Star-Topology Decoupled State-Space Search in AI Planning and Model Checking

A dissertation submitted towards the degree
Doctor of Natural Sciences
of the Faculty of Mathematics and Computer Science
of Saarland University

by

Daniel Gnad

Saarbrücken, 2021

Day of Colloquium	22.11.2021
Dean of Faculty	Prof. Dr. Thomas Schuster
Chair of the Committee	Prof. Dr. Jens Dittrich
Reviewers	Prof. Dr. Jörg Hoffmann
	Prof. Dr. Malte Helmert
	Prof. Bernd Finkbeiner, Ph.D.
Academic Assistant	Dr. Sophia Saller

Abstract

State-space search is a widely employed concept in many areas of computer science. The well-known state explosion problem, however, imposes a severe limitation to the effective implementation of search in state spaces that are exponential in the size of a compact system description, which captures the state-transition semantics.

Decoupled state-space search, decoupled search for short, is a novel approach to tackle the state explosion. It decomposes the system such that the dependencies between components take the form of a star topology with a center and several leaf components. Decoupled search exploits that the leaves in that topology are conditionally independent. Such independence naturally arises in many kinds of factored model representations, where the overall state space results from the product of several system components.

In this work, we introduce decoupled search in the context of artificial intelligence planning and formal verification using model checking. Building on common formalisms, we develop the concept of the decoupled state space and prove its correctness with respect to capturing reachability of the underlying model exactly. This allows us to connect decoupled search to any search algorithm, and, important for planning, adapt any heuristic function to the decoupled state representation. Such heuristics then guide the search towards states that satisfy a desired goal condition. In model checking, we address the problems of verifying safety properties, which express system states that must never occur, and liveness properties, that must hold in any infinite system execution.

Many approaches have been proposed in the past to tackle the state explosion problem. Most prominently partial-order reduction, symmetry breaking, Petri-net unfolding, and symbolic state representations. Like decoupled search, all of these are capable of exponentially reducing the search effort, either by pruning part of the state space (the former two), or by representing large state sets compactly (the latter two).

For all these techniques, we prove that decoupled search can be exponentially more efficient, confirming that it is indeed a novel concept that exploits model properties in a unique way. Given such orthogonality, we combine decoupled search with several complementary methods. Empirically, we show that decoupled search favourably compares to state-of-the-art planners in common algorithmic planning problems using standard benchmarks. In model checking, decoupled search outperforms well-established tools, both in the context of the verification of safety and liveness properties.

Zusammenfassung

Die Zustandsraumsuche ist ein weit verbreitetes Konzept in vielen Bereichen der Informatik, deren effektive Anwendung jedoch durch das Problem der Zustandsexplosion deutlich erschwert wird. Die Zustandsexplosion ist dadurch charakterisiert dass kompakte Systemmodelle exponentiell große Zustandsräume beschreiben.

Entkoppelte Zustandsraumsuche (entkoppelte Suche) beschreibt einen neuartigen Ansatz der Zustandsexplosion entgegenzuwirken indem die Struktur des Modells, insbesondere die bedingte Unabhängigkeit von Systemkomponenten in einer Sterntopologie, ausgenutzt wird. Diese Unabhängigkeit ergibt sich bei vielen faktorisierten Modellen deren Zustandsraum sich aus dem Produkt mehrerer Komponenten zusammensetzt.

In dieser Arbeit wird die entkoppelte Suche in der Planung, als Teil der Künstlichen Intelligenz, und der Verifikation mittels Modellprüfung eingeführt. In etablierten Formalismen wird das Konzept des entkoppelten Zustandsraums entwickelt und dessen Korrektheit bezüglich der exakten Erfassung der Erreichbarkeit von Modellzuständen bewiesen. Dies ermöglicht die Kombination der entkoppelten Suche mit beliebigen Suchalgorithmen. Wichtig für die Planung ist zudem die Nutzung von Heuristiken, die die Suche zu Zuständen führen, die eine gewünschte Zielbedingung erfüllen, mit der entkoppelten Zustandsdarstellung. Im Teil zur Modellprüfung wird die Verifikation von Sicherheits- sowie Lebendigkeitseigenschaften betrachtet, die unerwünschte Zustände, bzw. Eigenschaften, die bei unendlicher Systemausführung gelten müssen, beschreiben.

Es existieren diverse Ansätze um die Zustandsexplosion anzugehen. Am bekanntesten sind die Reduktion partieller Ordnung, Symmetriereduktion, Entfaltung von Petri-Netzen und symbolische Suche. Diese können, wie die entkoppelte Suche, den Suchaufwand exponentiell reduzieren. Dies geschieht durch Beschneidung eines Teils des Zustandsraums, oder durch die kompakte Darstellung großer Zustandsmengen.

Für diese Verfahren wird bewiesen, dass die entkoppelte Suche exponentiell effizienter sein kann. Dies belegt dass es sich um ein neuartiges Konzept handelt, das sich auf eigene Art der Modelleigenschaften bedient. Auf Basis dieser Beobachtung werden, mit Ausnahme der Entfaltung, Kombinationen mit entkoppelter Suche entwickelt. Empirisch kann die entkoppelte Suche im Vergleich zu modernen Planern zu deutlichen Vorteilen führen. In der Modellprüfung werden, sowohl bei der Überprüfung von Sicherheit-, als auch Lebendigkeitseigenschaften, etablierte Programme übertroffen.

Acknowledgements

First and foremost, I would like to thank Jörg Hoffmann for offering me the opportunity to enter the world of academia. I have always enjoyed (and still do) working with him. I am extremely grateful that he introduced me to the ICAPS community, which allowed me to meet so many great people in the research field. Thank you for all your support over the years, for pushing me when it was necessary, for creating such an amazing work environment in your group, and many fun activities outside work!

I also want to thank all my colleagues from the FAI group for the great time we were having: Álvaro, Dan, Daniel, Julia, Marcel, Marcel, Max, Rebecca, Thorsten. Special thanks go to Álvaro, Max, and Rebecca, for proofreading this thesis. I know it was a lot of work, and the schedule was tough. Thank you to Ankur, Endre, Felix, Joris, Marcel, Max, and Stefan, for uncountable TT matches giving us some distraction from work.

I want to take the opportunity to say thank you to my co-authors and the all nice people who hosted me for research visits in the last few years: Alberto, Alex, Alfonso, Carlos, Carmel, Dan, Erez, Facundo, Ivan, Malte, Martín, Martin.

A big thank you goes to my friends Stefan, Anne, Katharina, Patrick, Álvaro, and Siggi, as well as to my sports club TV Mettlach. Thank you for all the time we spent in various cool activities and for keeping me grounded, you are wonderful people!

I am deeply grateful to my family, my parents Monique and Wolfgang, my sisters Melanie and Hélène with Christian and Patrick, and Monika and Roland. Thank you so much for all the support throughout my life and for simply being a great family!

Finally, to the most important person in my life: Sabine, thanks for bearing with me!

Contents

Part I

Introduction and Planning Background

Chapter 1

Introduction

In the field of artificial intelligence (AI), automated planning is an area that aims at developing methods that allow machines to act strategically. It has a long tradition in AI, some of the first attempts to create intelligent machines dating back to the 1950's.[1]

"Planning is the art and practice of thinking before acting."
P@trik Haslum

While humans are arguably good at planning, i. e., predicting the consequences of their actions and acting accordingly, the task of teaching it to a machine is extremely challenging. Nevertheless, researchers have come a long way from the early attempts to today's sophisticated planning systems. Modern planners are able to outperform humans when solving huge models of planning tasks that are formulated in a restricted, yet expressive, formalism. The purpose of a general formalism is to allow for the development of *domain-independent* planners that can solve arbitrary planning tasks relying only on the model of the specific task at hand.

On a high level, automated planning deals with the problem of finding a sequence of actions that leads from the current state of the world to a state that satisfies certain desired goal conditions, if such a sequence exists. More concretely, the state of the world is typically described on an abstract level using *state variables* that capture the relevant characteristics that one is interested in. Variables could for example represent the location of objects in the world, their properties, or express facts that are true in the world. Actions describe an agent's possibilities to interact with the environment, like moving in it or manipulating objects. By performing actions, the agent changes the current world state, and transitions to a successor state. A sequence of actions that leads from the current state to a state in which all goal conditions are achieved is called a *plan*.

Various formalisms exist that allow to model planning tasks. In this work, we restrict ourselves to the most canonical such formalism, classical planning. Here, the state of the

[1]E.g., McCarthy, 1959; Newell and Simon, 1963; Ernst and Newell, 1969; Fikes and Nilsson, 1971.

world is always fully known to the agent, state variables have a finite discrete domain, and actions have unique outcomes that occur deterministically. Despite its simplicity, classical planning offers an expressive framework in which complex and interesting problems can be modeled.

Nowadays, the most popular way to solve classical planning tasks is *state-space search*, most often using a *heuristic function* for guidance towards the goal. The presented work, star-topology decoupled state-space search, indicated by its name, is a state-space search method as well. Decoupled search falls into the categories of state-space reformulation, such as factored planning, and partial-order reduction techniques, like strong stubborn sets pruning and Petri-net unfolding, as well as symbolic state representation, e. g., via binary decision diagrams. With factored planning, decoupled search shares the concept of not searching over full variable assignments (states), but partitioning the variables into separate factors, solving sub-tasks locally, and constructing global solutions by coordinating cross-factor interactions. Similar to partial-order reduction methods, decoupled search exploits the independence between actions that affect disjoint sets of variables, avoiding the enumeration of all interleavings of such actions. Decoupled search has in common with symbolic representations that search nodes do not correspond to single states, but compactly represent entire sets of states. Despite these similarities, we will show that decoupled search is a novel method that can provide exponential savings over all methods just mentioned.

Star-topology decoupled state-space search is a very general concept that applies to various kinds of factored state representations, i. e., representations where states are described in terms of several components. In Parts I–III of this work, we focus on the application of decoupled search to classical planning, formalizing states as assignments to a set of variables. In Part IV, we apply decoupled search to model checking, where the system components are non-deterministic automata. This generality, decoupled search being applicable to many formulations of state-space search problems, is shared with most of the aforementioned well-known reduction techniques, many of which are employed in the planning as well as the model-checking community, and other areas of computer science.

1.1 Illustrative Example

Throughout this work, we will use a logistics planning task as running example. It shall serve to illustrate the introduced concepts. Figure 1.1 shows a simple version of such a logistics task, with a truck T that has to transport several packages $p_1, \dots, p_4$ from their current to a goal location. The "world" here consists of four locations, $l_1, \dots l_4$, arranged in a line such that the truck can *drive* between any two adjacent locations. The truck can further *load* and *unload* a package if both are at the same location, respectively the package is in the truck. A state in the task is described in terms of five state variables

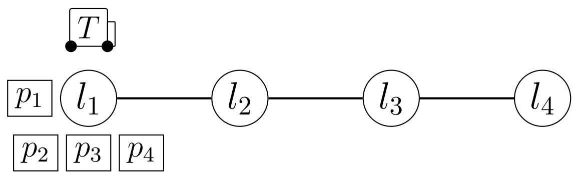


Figure 1.1: An illustration of the initial state of our running example.

$T, p_1, \dots p_4$, which indicate the positions of the five objects. For the truck T, these can be any one of $l_1, \dots, l_4$; the packages can additionally be placed in the truck.

Let's assume that the state depicted in the figure is the current state, denoted $\mathcal{I}$, and we want all packages to be transported to l_4. The way this planning task is usually solved via state-space search is to generate all successor states of $\mathcal{I}$ by applying all possible sequences of the actions drive, load, and unload, until we reach a state where all packages are in l_4. The *state space* constructed this way contains a total of 2500 states—one for each combination of truck and package locations. This indicates a major limitation to making any kind of state-space search method efficient, the **state explosion problem**—a compact representation of a planning task, here in terms of five state variables, describes a state space whose size is exponential in the number of variables.

Decoupled search is a novel concept and was specifically designed to tackle the state explosion problem. The key observation is that we do not have to enumerate all possible assignments to state variables explicitly if these assignments do not depend on each other. In our example, this holds for the four packages. Given a sequence of truck drives, the packages are *conditionally independent*, meaning that no matter what sequence of load/unload actions we choose for one package, we are free to use any other such sequence for the other packages. Hence, instead of enumerating all combinations of package positions, we can maintain the reachable locations for each package *separately* along with the current truck position.

Graphically, this kind of dependency can be illustrated using a structure known as the *causal graph*. The causal graph of our example is shown in Figure 1.2. The connections in the graph indicate the dependencies between the state variables: the packages depend on the truck (to be transported somewhere), the truck provides this service to the packages. Importantly, there is no direct connection between the packages, illustrating their conditional independence. The structure of the graph further shows where *star-topology* decoupled search got its name from. With the truck in the *center*, the packages form the *leaves* of a star topology.

In the state depicted in Figure 1.1, a package can be loaded into the truck independent of the other packages. Thus, we can compactly represent the 16 states where the truck is at l_1 and any combination of packages is either at l_1 or in the truck in a single so-called *decoupled state*. The set of states represented in the decoupled state corresponds to all states reachable from $\mathcal{I}$ via any sequence of only load/unload actions of

$$\begin{tikzcd}
\boxed{p_1} & & \boxed{p_2} \\
& \boxed{T} \arrow[ul] \arrow[ur] \arrow[dl, Circle-] \arrow[dr, Circle-] \arrow[loop right] & \\
\boxed{p_3} & & \boxed{p_4}
\end{tikzcd}$$

Figure 1.2: The causal graph of our running example.

the individual packages. More generally, all sequences of actions affecting only a leaf of the star topology can be enumerated, leaving the center actions to be branched over in the search. In our example, this means that decoupled search only branches over drive actions, which form the transitions in the *decoupled state space*. A decoupled state is then defined by the position of the truck and the possible locations of each package given the sequence of drive actions that leads to the decoupled state.

$$\begin{tikzcd}[column sep=small]
& & \mathcal{I} : T{=}l_1, p_x{=}l_1 \arrow[dll, "\text{load}"'] \arrow[dl, "\text{load}"'] \arrow[d, "\text{drive}"] \arrow[dr, "\text{load}"] \arrow[drr, "\text{load}"] & & \\
T{=}l_1, p_1{=}T, p_{2,3,4} = l_1 & T{=}l_1, p_2{=}T, p_{1,3,4} = l_1 & T{=}l_2, p_x{=}l_1 & T{=}l_1, p_3{=}T, p_{1,2,4} = l_1 & T{=}l_1, p_4{=}T, p_{1,2,3} = l_1
\end{tikzcd}$$

Figure 1.3: Part of the state space of our running example.

Consider Figures 1.3 and 1.4 for illustrations of part of the normal state space and the decoupled state space, respectively. In the normal state space, we see that the search constructs all five possible successors of $\mathcal{I}$, either driving the truck or loading any one package. In the next level, i. e., at a distance of two actions from $\mathcal{I}$, there would be 11 new states, out of which 6 encode states with the truck at l_1 and all combinations of any two packages loaded into the truck. We would see a similar branching at the subsequent level, where all combinations of any three packages are in the truck, and so on.

Explicitly enumerating all states where the truck is in l_1 that only differ in which packages are loaded, respectively still at l_1, results in 16 states. All these states are compactly represented in the initial decoupled state, which we denote by $\mathcal{I}^{\mathcal{F}}$. This is indicated by the notation $p_x = \{l_1, T\}$, stating that every package can be at l_1 or in the truck. We are free to choose any of the two positions for every package, independent of the others. In total, the decoupled state space only contains 20 decoupled states, greatly compressing the state space (with 2500 states), with no information loss.

While our running example is admittedly quite simplistic, we will prove that decoupled search can be exponentially more efficient than many existing methods. Comple-

$$\begin{tikzcd}
& \mathcal{I}^{\mathcal{F}} : T = l_1, p_x = \{l_1, T\} \arrow[d, "\text{drive}"] & \\
& T = l_2, p_x = \{l_1, l_2, T\} \arrow[dl, "\text{drive}"'] \arrow[dr, "\text{drive}"] & \\
T = l_1, p_x = \{l_1, l_2, T\} & & T = l_3, p_x = \{l_1, l_2, l_3, T\}
\end{tikzcd}$$

Figure 1.4: Part of the decoupled state space of our running example.

menting this theoretical analysis, our implementations of decoupled search in planning and model checking shows significant improvements over state-of-the-art systems on standard benchmarks.

1.2 Contributions

This thesis makes three core contributions:

1) We develop the basic concept of star-topology decoupled state-space search in the context of classical planning. This comprises the formal introduction of decoupled search in terms of the decoupled state space, proof of the correctness of the approach, and the development of the connection to heuristic search methods. We show that star-topology decoupled search is orthogonal to standard search algorithms, like breadth-first, A*, or greedy best-first search, and that in principle all classical planning heuristics can be used in decoupled search via a simple task compilation.

   Furthermore, we develop techniques to identify duplicate and (more importantly) *dominated* decoupled states, i. e., states that can be safely pruned during the search, without sacrificing completeness or optimality of the underlying search algorithm. Using dominance pruning, we prove that (1) the decoupled state space is finite, and (2) that the size of the decoupled state space is upper-bounded by the size of the standard state space when performing *hypercube pruning*, which involves solving a co-**NP**-complete subproblem for every decoupled state.

   We devise exponential separations of decoupled search to related methods to exemplify that no known method dominates decoupled search in terms of its reduction power. Such an exponential separation can be captured via a planning task family that is parametric in a natural number n that is scaled linearly. If the size of the state-space representation of method A is polynomial in n, whereas that of method B is exponential in n, we say that A is exponentially separated from B. We will give examples separating decoupled search from partial-order reduction, Petri-net unfolding, symmetry breaking, symbolic representations, and factored planning.

Moreover, we address the problem of how planning tasks can be decomposed to lead to a significant state-space reduction. This is an important question, since there are many—in fact exponentially many—ways to decompose a planning task. We propose several algorithms that partition the state variables into non-empty subsets, such that the dependencies between these components take the form of a star topology, and analyze the resulting reduction.

We provide a thorough experimental evaluation showing that decoupled search is able to outperform state-of-the-art planning systems on standard benchmarks that have a pronounced star topology. We distinguish three different types of experiments for our evaluation: (1) finding optimal solutions (in terms of sum of action costs), (2) finding any solution (satisficing planning), and (3) proving that no solution exists.

2) Looking into related state-space reduction methods, we observe that decoupled search is orthogonal to several existing techniques, namely partial-order reduction, symmetry breaking, symbolic representations, and dominance pruning, and can hence be combined with these methods. We develop such combinations for all these methods, prove their correctness, and show, both theoretically and empirically, that the combination can outperform its components. In particular, we (a) introduce strong stubborn sets pruning for decoupled search, avoiding unnecessary branching over center actions that leads to commutative parts in the decoupled state space. We (b) adapt symmetry-breaking techniques from planning to work in the decoupled-state setting, and (c) make use of the compact symbolic representation of leaf state spaces when the leaf components grow too large to be handled explicitly. Moreover, (d) we extend decoupled state dominance criteria by adapting methods known from optimal planning that can identify when a state s is "better" than another one s', implying that every plan for s' is also a plan for s. We can then discard s' if s has been seen before.

3) We apply decoupled search in the context of formal verification, in particular model checking. The aim of model checking is to check if a given system model satisfies certain desired properties. More specifically, we address the problem of model checking safety and liveness properties using state-space search. The former is conceptually close to planning in that both address the problem of checking if a state that satisfies certain properties, i. e., that satisfies the goal in planning, respectively that violates a desired property when verifying safety properties, is reachable from a given initial state. Both can be solved by search, constructing the state space of the specified model and checking if such a state is reachable.

Liveness checking is quite different, but can also be solved using state-space search. Here, instead of checking if a state is reachable, we look for an infinite sequence of states, a so-called *lasso*, that consists of a finite prefix and a cyclic part that contains an *accepting* state, i. e., a state that violates certain properties. Typically, liveness properties specify that "something good will eventually happen". The presence of a lasso then serves as a witness that there exists an infinite system execution in which nothing good ever happens. A possible approach to the formal verification of liveness properties is on-the-fly checking using nested depth-first search, which we will adapt to star-topology decoupled search.

Our contribution consists of (a) formalizing decoupled search in the setting of synchronized non-deterministic automata, (b) devising algorithms for safety and liveness checking, and proving their correctness. We (c) show exponential separations to related techniques, and (d) implemented prototypes of decoupled search for safety and liveness checking that we evaluate against established model checkers.

1.3 Publications

Most of the results of this thesis have been presented in the publications listed below. We have grouped the publications by relevance to the individual parts of this work.

The first three publications cover the main concepts of star-topology decoupled search in AI planning, formally introduce decoupled search, prove its correctness, discuss the relation to existing work, and prove the effectiveness of decoupled search in thorough experimental analyses. In the first work we introduce a restricted setting, namely fork instead of star topologies, where the leaf components can have dependencies on the center, but not vice versa. This is alleviated in the second publication. The third publication extends this further with a much more comprehensive evaluation, in particular adding theoretical results pertaining to decomposition methods, decoupled state dominance pruning, state-space size guarantees, and exponential separations from existing methods.

- Gnad, D. and Hoffmann, J. (2015a). Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Brafman, R., Domshlak, C., Haslum, P., and Zilberstein, S., editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 88–96. AAAI Press.

- Gnad, D., Hoffmann, J., and Domshlak, C. (2015). From fork decoupling to star-topology decoupling. In Lelis, L. and Stern, R., editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, pages 53–61. AAAI Press.

- Gnad, D. and Hoffmann, J. (2018). Star-topology decoupled state space search. *Artificial Intelligence*, 257:24 – 60.

The following seven publications establish further connections to related work, namely to dominance pruning, partial-order reduction, symbolic state representation, symmetry breaking, and Petri-net unfolding. These publications also provide combinations of decoupled search with all of these methods, except Petri-net unfolding. The last work introduces an advanced dominance relation over decoupled states.

- Torralba, Á., Gnad, D., Dubbert, P., and Hoffmann, J. (2016). On state-dominance criteria in fork-decoupled search. In Kambhampati, S., editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 3265–3271. AAAI Press/IJCAI.
- Gnad, D., Wehrle, M., and Hoffmann, J. (2016d). Decoupled strong stubborn sets. In Kambhampati, S., editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 3110–3116. AAAI Press/IJCAI.
- Gnad, D., Torralba, Á., and Hoffmann, J. (2017b). Symbolic leaf representation in decoupled search. In Fukunaga, A. and Kishimoto, A., editors, *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*. AAAI Press.
- Gnad, D., Torralba, Á., Shleyfman, A., and Hoffmann, J. (2017c). Symmetry breaking in star-topology decoupled search. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, pages 125–134. AAAI Press.
- Gnad, D., Hoffmann, J., and Wehrle, M. (2019a). Strong stubborn set pruning for star-topology decoupled state space search. *Journal of Artificial Intelligence Research*, 65:343–392.
- Gnad, D. and Hoffmann, J. (2019). On the relation between star-topology decoupling and petri net unfolding. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*, pages 172–180. AAAI Press.
- Gnad, D. (2021b). Revisiting dominance pruning in decoupled search. In Leyton-Brown, K. and Mausam, editors, *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21)*, pages 11809–11817. AAAI Press.

The next two publications analyze the problem of how to decompose a given planning task into center and leaf components, the *factoring* process. The first work introduces ways to compute an upper bound on the number of leaf factors by a reduction to the maximum independent set problem. It further describes a greedy factoring method that works very well in practice, but does not provide any guarantees. In the second work, we phrase the factoring process as an integer linear program (ILP). We prove that one of our ILP encodings guarantees to find a factoring with the maximum number of *mobile* leaf factors, if such a factoring exists. A factoring is mobile if for every leaf component L, there exists an action that only affects L.

- Gnad, D., Poser, V., and Hoffmann, J. (2017a). Beyond forks: Finding and ranking star factorings for decoupled search. In Sierra, C., editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 4310–4316. AAAI Press/IJCAI.

- Schmitt, F., Gnad, D., and Hoffmann, J. (2019). Advanced factoring strategies for decoupled search using linear programming. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*. AAAI Press.

The following two publications form the basis of our work in formal verification. We introduce star-topology decoupled state-space search in the context of model checking of safety properties in the SPIN model checker [Holzmann, 2004]. With the observation that safety checking is conceptually similar to classical planning, we develop a decoupled search extension to SPIN that shows strong experimental result on a set of standard Promela benchmarks [PromelaManual, 2020]. We extend the scope of decoupled search to model checking of liveness properties by adapting the well-known nested depth-first search algorithm that detects infinite accepting runs.

- Gnad, D., Dubbert, P., Lluch-Lafuente, A., and Hoffmann, J. (2018a). Star-topology decoupling in SPIN. In del Mar Gallardo, M. and Merino, P., editors, *Proceedings of the 25th International Symposium on Model Checking of Software (SPIN'18)*, Lecture Notes in Computer Science. Springer.

  Winner of the SPIN Best Paper Award.

- Gnad, D., Eisenhut, J., Lluch-Lafuente, A., and Hoffmann, J. (2021c). Model checking ω-regular properties with decoupled search. In Silva, A. and Leino, K. R. M., editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*, volume 12760 of *Lecture Notes in Computer Science*, pages 411–434. Springer.

The following works were published as part of the author's doctoral studies, but are not covered by this thesis.

The first five publications are in the context of red-black planning [Domshlak et al., 2015a], a relaxation technique that is based on delete-relaxation [Hoffmann and Nebel, 2001]. The first and third publication introduce new heuristic functions for satisficing planning. The second and fourth publication develop the concept of *red-black search*, where the state representation is changed to allow for a certain degree of delete-relaxation within the search, in contrast to the traditional approach of relaxing only in a heuristic that guides the search. In the second paper we introduce an algorithm that computes red-black plans that are fed into a tool for plan repair. The contribution of the fourth paper is to remove this last step and iteratively refine the relaxation until the relaxed plan is a correct plan. Although this is reminiscent of counter-example guided abstraction refinement techniques [Clarke et al., 2003; Seipp and Helmert, 2013], the red-black relaxation is not an abstraction in the usual sense. The last paper aims at optimizing the encoding of the planning task to fit to the tractable fragment required by the red-black heuristic used in the Mercury and MERWIN planners [Katz and Hoffmann, 2014; Katz et al., 2018].

- Gnad, D. and Hoffmann, J. (2015b). Red-black planning: A new tractability analysis and heuristic function. In Lelis, L. and Stern, R., editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*. AAAI Press.

- Gnad, D., Steinmetz, M., Jany, M., Hoffmann, J., Serina, I., and Gerevini, A. (2016b). Partial delete relaxation, unchained: On intractable red-black planning and its applications. In Baier, J. and Botea, A., editors, *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS'16)*. AAAI Press.

- Speicher, P., Steinmetz, M., Gnad, D., Hoffmann, J., and Gerevini, A. (2017). Beyond red-black planning: Limited-memory state variables. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, pages 269–273. AAAI Press.

- Fickert, M., Gnad, D., and Hoffmann, J. (2018a). Unchaining the power of partial delete relaxation, part II: finding plans with red-black state space search. In Lang, J., editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 4750–4756.

- Fišer, D., Gnad, D., Katz, M., and Hoffmann, J. (2021). Custom-design of FDR encodings: The case of red-black planning. In Zhou, Z.-H., editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 4054–4061.

The following publication is in the context of grounding planning tasks from a compact formulation on lifted PDDL level to a grounded finite-domain representation [McDermott et al., 1998; Helmert, 2009; Haslum et al., 2019]. The grounding phase can sometimes be the bottleneck of the overall planning process, namely if there exists a blow-up in size of the grounded compared to the lifted representation. In this work, we tackle this blow-up by developing methods that do not fully ground a planning task, but that aim at minimizing the size of the grounded representation without sacrificing the completeness of the overall planning algorithm. We do so by devising heuristic-style guidance mechanisms that steer the grounding process towards parts of the task that are deemed relevant to find a plan. We propose prioritization criteria that are based on (1) syntactic characteristics of the lifted task, computed on the fly during grounding, and (2) a learning approach that is trained on a set of small instances of a domain and applied to large problem instances in which grounding becomes an issue.

- Gnad, D., Torralba, Á., Domínguez, M., Areces, C., and Bustos, F. (2019c). Learning how to ground a plan – partial grounding in classical planning. In Hentenryck, P. V. and Zhou, Z.-H., editors, *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, pages 7602–7609. AAAI Press.

During his doctoral studies, the author participated with five planners in the International Planning Competitions 2016 and 2018, as well as in the Sparkle Planning Challenge 2019.

- Gnad, D., Torralba, Á., Hoffmann, J., and Wehrle, M. (2016c). Decoupled search for proving unsolvability. In *UIPC 2016 planner abstracts*, pages 16–18.

  The planner is based on decoupled search with a simple factoring strategy, and makes use of the combinations with strong stubborn sets and dominance pruning discussed in Part III.

  The planner achieved the 5th place out of 11 participants.

- Gnad, D., Steinmetz, M., and Hoffmann, J. (2016a). Django: Unchaining the power of red-black planning. In *UIPC 2016 planner abstracts*, pages 19–23.

  Our planner is based on red-black search, incrementally unrelaxing the red-black task transformation until it is proven unsolvable.

  The planner achieved the 8th place out of 11 participants.

- Gnad, D., Shleyfman, A., and Hoffmann, J. (2018b). DecStar - star-topology decoupled search at its best. In *IPC 2018 planner abstracts*.

  The planner competed in all tracks with different search configurations. It uses decoupled search whenever a task decomposition can be identified, and exploits all combinations of techniques from Part III, except symbolic leaf representations.

  The planner achieved the 6th place in the optimal track (out of 10), the 4th place in the satisficing track (out of 17), the 6th place in the agile track (out of 18), and the 7th place in the cost-bounded track (out of 10).

- Fickert, M., Gnad, D., Speicher, P., and Hoffmann, J. (2018b). Saarplan: Combining Saarland's greatest planning techniques. In *IPC 2018 planner abstracts*.

  The planner competed in all but the optimal track of the competition, with different search configurations. It uses a variety of techniques developed in the Foundations of Artificial Intelligence group at Saarland University. Among others, it has components that are based on decoupled search, and a component based on a red-black heuristic.

The planner was awarded as **Runner-Up** in the **agile** and **cost-bounded tracks**, as well as **recognized** for having the **highest coverage** in the **satisficing** and **agile tracks**. Furthermore, it finished **3rd** in the **satisficing track** (out of 17).

- Gnad, D., Torralba, Á., Domínguez, M., Areces, C., and Bustos, F. (2019b). IPA LAMA: Planner abstract. In *Sparkle Planning Challenge 2019*.

  Our planner, based on partial grounding, finished 8th among 10 participants.

1.4 Outline

The remainder of this work is organized as follows: Chapter 2 gives a formal background of classical planning, state spaces of planning tasks, and search methods, including heuristic search, that are used to solve classical planning tasks. We also formally define exponential separations in there.

Part II introduces star-topology decoupled search in the context of classical planning in Chapter 3, proving its correctness and investigating dominance pruning over decoupled states and its implications on the size of the decoupled state space. We develop the connection to heuristic search methods in Chapter 4. Chapter 5 looks more closely into how the problem decomposition for decoupled search can be done for arbitrary planning tasks, answering the question of how to automatically partition the state variables of a task into center and leaf components. In Chapter 6, we provide exponential separations, i. e., families of scalable planning task, leading to a polynomial increase in the search effort of one, vs. an exponential increase for the other method. Finally, Chapter 7 shows an empirical evaluation of decoupled search in classical planning. We summarize our findings from Part II in Chapter 8.

In Part III, we take a closer look at orthogonal search reduction methods, in particular partial-order reduction (Chapter 10), symmetry breaking (Chapter 11), symbolic state representations (Chapter 12), and dominance pruning (Chapter 13). We formally introduce all methods and develop combinations with decoupled search. We prove the correctness of the combined algorithms, and show that these can have advantages over the base techniques both theoretically and empirically. We summarize our findings from Part III in Chapter 14.

Part IV investigates the application of decoupled search to model checking of safety and liveness properties. First, we give the required model-checking background in Chapter 16, we then formalize decoupled search in the framework of synchronized nondeterministic automata in Chapter 17. For both safety (Chapter 18) and liveness (Chapter 19) checking, we show how to adapt existing algorithms to the context of decoupled search, and prove the correctness of our adaptations. We illustrate the effectiveness of decoupled search in a prototype implementation. Part IV ends with a discussion of related work in Chapter 20. We summarize our findings from Part IV in Chapter 21.

We conclude the thesis in Part V and give an outlook to possible research directions for future work.

Chapter 2

Background

We start by giving the required background about classical planning in terms of the finite-state representation (FDR), a standard framework to formalize planning tasks. We formally define the state space, a labeled transition system, induced by a planning task. In Chapter 2.2, we give a brief overview on how planning tasks are typically solved with state-space search methods. Last, we formally define the concept of exponential separations. Some of the definitions and descriptions in this chapter follow the Background section of Gnad and Hoffmann [2018].

2.1 Classical Planning

There exist many variants of AI planning, which differ in terms of the expressiveness of the planning models. This includes, for example, variants with continuous state variables, uncertainty about the initial state and/or action outcomes, or temporal aspects. For a comprehensive overview, please see Ghallab et al. [2004]. In this work, we consider *classical planning*, which assumes discrete state variables with finite domains, complete knowledge about the initial state, and deterministic actions, where effects occur instantaneously and atomically. In the literature, this classical planning formalism is referred to as *finite-domain representation* (FDR), or *SAS*+ [Bäckström and Nebel, 1995; Edelkamp and Helmert, 1999; Helmert, 2006b, 2009].

Definition 1 (Planning Task)**.** *A* planning task *is a tuple* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$*, where*

- $\mathcal{V}$ *is a finite set of* variables*, each* $v \in \mathcal{V}$ *is associated with a finite* domain $\mathcal{D}(v)$*. A complete assignment to* $\mathcal{V}$ *is called a* state*, partial assignments to* $\mathcal{V}$ *are called* partial states*;*
- $\mathcal{A}$ *is a finite set of* actions*. Each action* $a \in \mathcal{A}$ *is a pair* $\langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$*, where* $\mathsf{pre}(a)$ *is the* precondition *of* a*, and* $\mathsf{eff}(a)$ *is its* effect*, both partial variable assignments;*

- $\mathsf{cost} : \mathcal{A} \to \mathbb{R}^{0+}$ *is the* cost *function of* Π, *mapping each action* $a \in \mathcal{A}$ *to its non-negative cost* $\mathsf{cost}(a)$*;*
- $\mathcal{I}$ *is the* initial state*;*
- $\mathcal{G}$ *is the* goal*, a partial state.*

The assignment to a single variable $v \in \mathcal{V}$ is called a *fact*, denoted by a variable/value pair $\langle v, d\rangle$ (we will also use the notation $v = d$) with $d \in \mathcal{D}(v)$. For ease of notation, we will often interpret (partial) states as sets of facts. For a partial variable assignment p we denote the subset of variables defined in p by $\mathsf{vars}(p) \subseteq \mathcal{V}$. For a (partial) state s of Π, we denote the assignment to a variable $v \in \mathsf{vars}(s)$ by $s[v]$ and similarly for a set of variables V by $s[V] := \{s[v] \mid v \in \mathsf{vars}(s) \cap V\}$. We say that a (partial) assignment p *satisfies* another (partial) assignment q, denoted $p \models q$, if $\mathsf{vars}(q) \subseteq \mathsf{vars}(p)$ and for all $v \in \mathsf{vars}(q) : p[v] = q[v]$, or alternatively in set-notation $q \subseteq p$.

An action $a \in \mathcal{A}$ is *applicable* in a state s, if $s \models \mathsf{pre}(a)$. Applying a in s results in the successor state $s[\![a]\!] := s[\mathcal{V} \setminus \mathsf{vars}(\mathsf{eff}(a))] \cup \mathsf{eff}(a)$, i. e., we overwrite s with the effects of a where defined. We will also apply actions to partial states p; then, an action a is applicable in p if $p \models \mathsf{pre}(a)[\mathsf{vars}(p)]$, the resulting state is defined as $p[\![a]\!] := p[\mathsf{vars}(p) \setminus \mathsf{vars}(\mathsf{eff}(a))] \cup \mathsf{eff}(a)[\mathsf{vars}(p)]$. We say that an action a *affects* a variable v if it has an effect on v, i. e., $v \in \mathsf{vars}(\mathsf{eff}(a))$.

The semantics of a planning task are defined via its *state space*, a deterministic *labeled transition system*. The state space is defined directly on the states and actions of a planning task. We next define labeled transition systems, then show how the state space of a task is constructed.

Definition 2 (Labeled Transition System)**.** *A* labeled transition system *(LTS) is a tuple* $\Theta = \langle S, L, c, T, I, S_G\rangle$ *consisting of a finite set of* states S*, a finite set of transition* labels L*, a function* $c : L \mapsto \mathbb{R}^{0+}$ *associating each label with its non-negative* cost*, a set of* transitions $T \subseteq S \times L \times S$*, an* initial state $I \in S$*, and a set of* goal states $S_G \subseteq S$*.*

We will often write $s \xrightarrow{l} s'$ for $\langle s, l, s'\rangle \in T$, or $s \to s'$ if the label does not matter. A *path* in an LTS is a sequence of labels $\pi = \langle l_1, \dots, l_n\rangle$ such that there exist states $s_0, \dots, s_n$ with $s_i \xrightarrow{l_{i+1}} s_{i+1} \in T$ for all $0 \leq i < n$. Such a path π is a *solution* for a state s if $s_0 = s$ and $s_n \in S_G$. A solution for I is a *solution* for the LTS. The *cost* of a path π, denoted $c(\pi)$, is the sum of the costs of its labels, $c(\pi) := \sum_{l_i \in \pi} c(l_i)$. A solution π (for a state s) is *optimal* if its cost $c(\pi)$ is minimal among all solutions (for s).

We say that a state s' is *reachable from a state* s if there exists a path that starts in s and ends in s'. A state s is *reachable* in an LTS if it is reachable from I.

We assume that LTSs are *deterministic*, i. e., for every $s \in S$ and $l \in L$ there exists at most one $s' \in S$ such that $\langle s, l, s'\rangle \in T$.

Definition 3 (State Space)**.** *The* state space *of a planning task* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *is the labeled transition system* $\Theta_\Pi = \langle S, L, c, T, \mathcal{I}, S_\mathcal{G} \rangle$ *where*

- S *is the set of all states of* Π*,*
- *the labels* $L = \mathcal{A}$ *are the actions of* Π*,*
- *the cost function* $c = \mathsf{cost}$ *is the cost function of* Π*,*
- $\langle s, a, s' \rangle \in T$*, iff* $s, s' \in S$*,* $a \in \mathcal{A}$*,* $s \models \mathsf{pre}(a)$*, and* $s[\![a]\!] = s'$*,*
- $I = \mathcal{I}$ *is the initial state of* Π*, and*
- $S_\mathcal{G}$ *is set of goal states of* Π*, where* $s_G \in S_\mathcal{G}$ *iff* $s_G \models \mathcal{G}$*.*

An (optimal) solution π for a state s in Θ_Π is an (optimal) *plan* for s, i. e., a sequence of actions that leads from s to a state that satisfies the goal $\mathcal{G}$. An (optimal) plan for $\mathcal{I}$ is an (optimal) solution for Π.

The state space of a task is generally exponentially larger than the compact description of the task, which is known as the *state explosion problem*. Thus, techniques need to be developed to systematically explore the state space. In the next section, we will introduce heuristics functions, a popular means to draw the search exploration towards promising states, i. e., states close to the goal.

Given a planning task Π, deciding whether a plan exists is **PSPACE**-complete [Bylander, 1994]. At an algorithmic level, planning distinguishes mainly three different problems: *optimal planning*, where the objective is to find an optimal plan; *satisficing planning*, where it suffices to find any plan; and *proving unsolvability*, where the objective is to prove that no goal state is reachable.

2.2 Heuristic Search

In this work, we focus on what is currently the most popular approach to solve classical planning problems, namely (heuristic) state-space search [Bonet and Geffner, 2001]. Search algorithms *systematically* explore the state space Θ_Π of a planning task Π by following the transitions in the state space. We will only consider *forward* search methods, which, starting in the initial state $\mathcal{I}$, generate the successors of a state by applying the applicable actions. The states generated in the search can be ranked by a *heuristic*, which estimates the distance from a state to a goal state and thereby prioritizes states that are closer to the goal.

Definition 4 (Heuristic)**.** *A* heuristic $h : S \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ *is a function that maps each state* s *of a planning task* Π *to an estimate of the cost of a plan for* s*, or* ∞ *indicating that no plan exists. The* perfect heuristic h^* *maps each state* s *to the cost of an optimal* *plan for s, or ∞ if no such plan exists. A heuristic h is* admissible *if $h(s) \leq h^*(s)$ for all states s.*

Heuristics form an important component in planning systems based on search, since they guide the search towards states that are most promising [McDermott, 1999; Bonet and Geffner, 2001; Hoffmann and Nebel, 2001; Gerevini et al., 2003; Helmert, 2006b; Helmert and Domshlak, 2009; Richter and Westphal, 2010; Domshlak et al., 2015a].

We will adapt several popular planning heuristics for decoupled search and provide a compilation that enables the use of *any* heuristic. With these heuristics available, we will deploy decoupled search for best-first search algorithms, namely A^* [Hart et al., 1968; Pearl, 1984], which can be used for optimal planning when using an admissible heuristic, and greedy best-first search (GBFS) [Doran and Michie, 1966], which is typically used for satisficing planning. The concept of decoupled search is orthogonal to the use of search algorithms, and can, in principle, be combined with any search algorithm.

2.3 Problem Structure – The Causal Graph

Many works in planning have investigated the structure of planning tasks to facilitate the search (e. g. [Domshlak and Dinitz, 2001; Helmert, 2004; Haslum et al., 2007; Katz and Domshlak, 2008; Giménez and Jonsson, 2012; Katz and Keyder, 2012; Pommerening et al., 2013; Aghighi et al., 2015; Domshlak et al., 2015a]). An important notion in this analysis is the *causal graph* CG_Π of a planning task Π [Knoblock, 1994; Jonsson and Bäckström, 1995; Brafman and Domshlak, 2003; Helmert, 2006b; Giménez and Jonsson, 2008].

Definition 5 (Causal Graph)**.** *The* causal graph CG_Π *of a planning task Π is a digraph with vertices $\mathcal{V}$ and edges E, where $\langle v, v' \rangle \in E$ iff $v \neq v'$ and there exists an action a such that $\langle v, v' \rangle \in (\mathsf{vars}(\mathsf{pre}(a)) \cup \mathsf{vars}(\mathsf{eff}(a))) \times \mathsf{vars}(\mathsf{eff}(a))$.*

The causal graph captures direct state variable dependencies induced by the actions. It captures two forms of dependencies, (i) *precondition-effect* dependencies, if $v \in \mathsf{vars}(\mathsf{pre}(a))$ and $v' \in \mathsf{vars}(\mathsf{eff}(a))$, as well as (ii) *effect-effect* dependencies, if both $v, v' \in \mathsf{vars}(\mathsf{eff}(a))$. Intuitively, given a causal graph arc $v \rightarrow v'$, changing the value of v' may involve changing that of v as well, because either (i) v may need to provide a precondition, or (ii) v may be affected as a side effect of changing the value of v'.

Causal graphs will play a key role throughout this work, because they allow to reason about (sets of) variables that are *independent* of other variables of a planning task.

2.4 Exponential Separation

Throughout this work, we will compare the reduction power of decoupled search to that of several existing techniques, as well as different variants of decoupled search to each other. We do so via so-called *exponential separations*, families of planning tasks that can be scaled in a parameter n. This allows us to reason about the behaviour of different search methods when increasing the size of the planning tasks that are tackled.

Definition 6 (Exponential Separation)**.** *Let* $\{\Pi^n \mid n \in \mathbb{N}^+\}$ *be a family of planning tasks of size (number of facts and actions) polynomially related to* n*. Then search method* X *is* exponentially separated *from search method* Y *if (i) the representation size of the reachable state space under* X *is bounded by a polynomial in* n*, while (ii) the representation size of the reachable state space under* Y *is exponential in* n*.*

By "representation size", we typically refer to the number of states. While these are always compact for explicit-state search, decoupled states themselves can have a size that is exponential in the number of variables. In our exponential separations, the task families and decompositions will always ensure that decoupled states are compact, so there is no hidden blow-up that is not represented by the number of states reached.

Part II

Star-Topology Decoupled State-Space Search

Chapter 3

Decoupled State-Space Search

In this chapter, we will introduce decoupled search for classical planning. We will formalize the decomposition of a given planning task to obtain the required structure in Chapter 3.1, then formally define the decoupled state space and prove its correctness in Chapters 3.2 and 3.3. We then look into decoupled state dominance pruning and size guarantees for the decoupled state space in Chapter 3.4.

This chapter is for the largest part based on Gnad and Hoffmann [2018]. In Chapter 3.4, we extend the dominance pruning from Gnad and Hoffmann [2018] by a more sophisticated variant introduced in Gnad [2021b]. The idea behind the proof of Theorem 3 is due to Álvaro Torralba.

3.1 Problem Decomposition

The main difference between decoupled search and explicit-state search methods is that decoupled search does not operate on full states, but on sets of partial states. This is done by splitting the state variables $\mathcal{V}$ of a planning task Π into disjoint non-empty subsets, the *factors*. Such a partition of the state variables is called a *factoring* $\mathcal{F}$. The search explores sequences of actions that affect the so-called *center* factor, while enumerating sets of partial states that can be reached with actions that only affect one of the other factors, the *leaves*.

Definition 7 (Star Factoring)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task. A* factoring $\mathcal{F}$ *for* Π *is a partition of its variables* $\mathcal{V}$ *into disjoint non-empty subsets.*

A factoring $\mathcal{F}$ *is a* star factoring, *if* $|\mathcal{F}| > 1$ *and there exists a* center factor $C \in \mathcal{F}$ *such that, with* $\mathcal{L} := \mathcal{F} \setminus \{C\}$ *being the* leaf factors *of* $\mathcal{F}$*, for all actions* $a \in \mathcal{A}$ *it holds that either there exists an* $L \in \mathcal{L}$ *such that* $\mathsf{vars}(\mathsf{pre}(a)) \subseteq C \cup L$ *and* $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L$*, or* $\mathsf{vars}(\mathsf{eff}(a)) \cap C \neq \emptyset$*.*

In words, in a star factoring $\mathcal{F}$, there are no restrictions on actions affecting the center C, actions that do not affect C are limited to effects on a single leaf L and preconditions on L and C. From here on, we will assume that all factorings are star factorings and omit the “star” prefix. We will investigate different special types of factorings, and *how* factorings can be identified in Chapter 5.

Complete assignments to C are called *center states*, complete assignments to an $L \in \mathcal{L}$ are called *leaf states*. We denote the set of leaf states of a particular leaf $L \in \mathcal{L}$ by S^L and the set of all leaf states by $S^{\mathcal{L}} := \bigcup_{L \in \mathcal{L}} S^L$. As a notation convention, we will denote center states with a superscript C, e. g., s^C, and leaf states with a superscript L, e. g., s^L. A center state s^C is a *center goal state* iff $s^C \models \mathcal{G}[C]$, i. e., it satisfies all goals that are defined over center variables. Similarly, leaf state s^L of leaf L is a *leaf goal state* iff $s^L \models \mathcal{G}[L]$, i. e., it satisfies all goals that are defined over L.

For a factoring $\mathcal{F}$, we define the *center actions* $\mathcal{A}^C$ as the actions that affect C, formally $\mathcal{A}^C := \{a \in \mathcal{A} \mid \mathsf{vars}(\mathsf{eff}(a)) \cap C \neq \emptyset\}$, and the *leaf actions* $\mathcal{A}^L$ as the actions affecting a leaf L, formally $\mathcal{A}^L := \{a \in \mathcal{A} \mid \mathsf{vars}(\mathsf{eff}(a)) \cap L \neq \emptyset\}$. We denote the set of all leaf actions by $\mathcal{A}^{\mathcal{L}} := \bigcup_{L \in \mathcal{L}} \mathcal{A}^L$. We remark that $\mathcal{A}^C$ and $\mathcal{A}^{\mathcal{L}}$ are not necessarily disjoint, i. e., there can be actions that affect both center and, potentially several, leaves. An important type of actions are those leaf actions that are not at the same time center actions. We call these *leaf-only actions* and define the set of all leaf-only actions by $\mathcal{A}^{\mathcal{L}}_{\not C} := \mathcal{A}^{\mathcal{L}} \setminus \mathcal{A}^C$, and those of a particular leaf L by $\mathcal{A}^L_{\not C} := \mathcal{A}^L \setminus \mathcal{A}^C$.

We call the preconditions $\mathsf{pre}(a)[C]$, respectively effects $\mathsf{eff}(a)[C]$, of an action $a \in \mathcal{A}$ on the center factor C *center preconditions*, respectively *center effects*. Similarly, we call the preconditions/effects on a leaf factor $L \in \mathcal{L}$ its *leaf preconditions/effects*. A sequence of center actions applicable to $\mathcal{I}$ when ignoring all leaf preconditions is called a *center path*, notation convention π^C. A sequence of leaf actions of a leaf L applicable to $\mathcal{I}$ when ignoring the preconditions on $\mathcal{V} \setminus L$ is called a *leaf path*, notation convention π^L. For an arbitrary path π of actions in $\mathcal{A}$, by $\pi^C(\pi)$ we denote its projection onto the center actions $\mathcal{A}^C$.

We will illustrate these concepts on a variation of the example from the introduction with two trucks and three packages, which is shown in Figure 3.1:

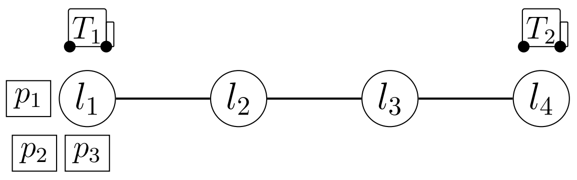


Figure 3.1: An illustration of the initial state of our running example. In the goal, all packages need to be in l_4.

The example can be formalized as an FDR task as follows:

- $\mathcal{V} = \{T_1, T_2, p_1, p_2, p_3\}$,
- $\mathcal{A} = \{\text{drive}(T_i, a, b) \mid i \in \{1,2\}, \{a,b\} \in \{\{l_1,l_2\},\{l_2,l_3\},\{l_3,l_4\}\}\} \cup$
  $\{\text{load}(T_i, p_j, l_k) \mid i \in \{1,2\}, j \in \{1,2,3\}, k \in \{1,2,3,4\}\} \cup$
  $\{\text{unload}(T_i, p_j, l_k) \mid i \in \{1,2\}, j \in \{1,2,3\}, k \in \{1,2,3,4\}\}$,

  where:

  $\mathsf{pre}(\text{drive}(T_i,a,b)) = \{T_i = a\}, \mathsf{eff}(\text{drive}(T_i,a,b)) = \{T_i = b\}$,
  $\mathsf{pre}(\text{load}(T_i,p_j,l_k)) = \{T_i = l_k, p_j = l_k\}, \mathsf{eff}(\text{load}(T_i,p_j,l_k)) = \{p_j = T_i\}$,
  $\mathsf{pre}(\text{unload}(T_i,p_j,l_k)) = \{T_i = l_k, p_j = T\}, \mathsf{eff}(\text{unload}(T_i,p_j,l_k)) = \{p_j = l_k\}$.
- $\mathcal{I} = \{T_1 = l_1, T_2 = l_4, p_1 = l_1, p_2 = l_1, p_3 = l_1\}$,
- $\mathcal{G} = \{p_1 = l_4, p_2 = l_4, p_3 = l_4\}$.

A possible factoring of this task puts the trucks together and each of the packages into an individual factor, so $\mathcal{F}_1 = \{\{T_1, T_2\}, \{p_1\}, \{p_2\}, \{p_3\}\}$. Taking the trucks as the center factor, $C = \{T_1, T_2\}$, and each package as a leaf $L_i = \{p_i\}$, this is indeed a star factoring, where the load/unload actions are leaf-only actions with preconditions on the truck and the respective package, and effects only on the package. The truck drives are the center actions with preconditions and effects only on a truck.

Alternatively, the variables can be partitioned such that all packages form the center factor $C = \{p_1, p_2, p_3\}$ and each of the trucks becomes a leaf, $\mathcal{L} = \{\{T_1\}, \{T_2\}\}$. Here, the drive actions are leaf-only actions and the load/unload actions are center actions with leaf preconditions. We will refer to this factoring by $\mathcal{F}_2$.

A center state is an assignment to the truck variables, e. g., $s^C = \{T_1 = l_1, T_2 = l_4\}$, a leaf state is an assignment to one of the package variables, e. g., $s^{L_1} = \{p_1 = l_2\}$ (both examples are for $\mathcal{F}_1$).

The causal graph of the task is illustrated in Figure 3.2:

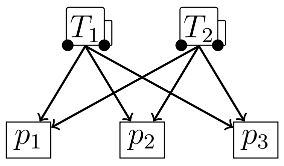


Figure 3.2: The causal graph of our running example.

The key observation for decoupled search is that the leaf factors, the packages in our example with $\mathcal{F}_1$, are conditionally independent. Given a sequence of drive actions π^C that move the trucks, for example:

$$\pi^C = \text{drive}(T_1, l_1, l_2), \text{drive}(T_1, l_2, l_3), \text{drive}(T_1, l_3, l_4),$$

we can choose an arbitrary sequence of *complying*[1] leaf paths, i. e., a sequence of leaf-only actions that can be scheduled along π^C such that all preconditions on the center are fulfilled. Some compliant leaf paths are for example:

$$\begin{aligned}
\pi_1^{L_1} &= \text{load}(T_1, p_1, l_1), \text{unload}(T_1, p_1, l_2)\\
\pi_2^{L_1} &= \text{load}(T_1, p_1, l_1), \text{unload}(T_1, p_1, l_3)\\
\pi_1^{L_3} &= \text{load}(T_1, p_3, l_1), \text{unload}(T_1, p_3, l_2)\\
\pi_2^{L_3} &= \langle\rangle
\end{aligned}$$

The leaf path $\pi_1^{L_1}$ can for example be scheduled along π^C as follows:

$$\text{load}(T_1, p_1, l_1), \text{drive}(T_1, l_1, l_2), \text{unload}(T_1, p_1, l_2), \text{drive}(T_1, l_2, l_3), \text{drive}(T_1, l_3, l_4)$$

We can also decide not to move a package at all as in $\pi_2^{L_3}$.

The important point is that independent of which of the paths $\pi_1^{L_1}, \pi_2^{L_1}$ we choose for leaf L_1, we are free to choose any compliant path for, e. g., leaf L_3. This is because leaf-only actions of a leaf L do not affect variables in another leaf $L' \in \mathcal{L} \setminus \{L\}$. This property is exploited by decoupled search by searching only over center paths π^C, maintaining for each leaf the set of leaf states reachable via leaf paths compliant with π^C. This avoids the enumeration of all global states that result from combinations of these leaf states across leaves, leading to exponential savings. More specifically, the reduction achieved by maintaining leaf paths separately is exponential in the number of leaves. It is, for each leaf factor, limited by the number of leaf-only actions. Thus, these actions play an important role for the reduction power of decoupled search. The more leaf-only actions are induced by a factoring, the higher the reduction because the search branches over fewer actions, and potentially more leaf paths are compliant.

We will formalize the concepts of compliant paths and the decoupled state space in the next sections.

3.2 Decoupled State Space

Decoupled search exploits the independence of the leaf factors by only searching over sequences of center actions, i. e., center paths π^C, and enumerating for each such center path the leaf paths π^L compliant with π^C for each leaf separately. We will introduce the concept of compliant paths next, and, building thereupon, define the compliant-path graph and the decoupled state space.

[1]We will formally define the notion of compliant paths in the next section.

3.2.1 Compliant-Path Graph

A leaf path π^L of leaf L *complies* with a center path π^C, π^L is π^C-*compliant*, if: (1) the subsequences of $\mathcal{A}^L \cap \mathcal{A}^C$ actions in π^L and π^C coincide; and (2) the leaf-only actions $\mathcal{A}^L_{\not C}$ in π^L can be scheduled alongside π^C so that (a) for the actions in π^L all preconditions on C are satisfied, and (b) for the actions in π^C all preconditions on L are satisfied. In the above example, (1) is moot because $\mathcal{A}^{\mathcal{L}} \cap \mathcal{A}^C$ is empty for both $\mathcal{F}_1$ and $\mathcal{F}_2$. Condition (2a) matters for $\mathcal{F}_1$ only because the package moves (load/unload) rely on center preconditions. Condition (2b) is moot for $\mathcal{F}_1$ because center actions do not have leaf preconditions, it is effective for $\mathcal{F}_2$, though, since the center actions load/unload have preconditions on the truck positions. The leaf paths complying with a given center path π^C can be easily maintained in the form of a layered compliant-path graph (with layers corresponding to the time steps along π^C).

Each center path π^C in the search ends in a *decoupled state* $s^{\mathcal{F}}$. We denote the center path on which a decoupled state is reached by $\pi^C(s^{\mathcal{F}})$. The decoupled state $s^{\mathcal{F}}$ contains a center state $\mathsf{center}(s^{\mathcal{F}})$. For the empty center path $\pi^C(s^{\mathcal{F}}) = \langle\rangle$, we have $\mathsf{center}(s^{\mathcal{F}}) = \mathcal{I}[C]$, in the example using $\mathcal{F}_1$ this is $\{T_1 = l_1, T_2 = l_4\}$. The decoupled state furthermore contains a *pricing function*, a mapping $\mathsf{prices}(s^{\mathcal{F}}) : S^{\mathcal{L}} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ from *leaf states* to non-negative numbers, or ∞ to indicate unreachable leaf states.

In the initial decoupled state $s^{\mathcal{F}}$ of the example with $\mathcal{F}_1$, the price of $\{p_1 = l_1\}$ is 0 because the empty leaf path $\pi^L = \langle\rangle$ complies with $\pi^C(s^{\mathcal{F}}) = \langle\rangle$. The price of $\{p_1 = T_1\}$ is 1 because the leaf path $\pi^L = \langle \text{load}(T_1, p_1, l_1)\rangle$ complies with $\pi^C(s^{\mathcal{F}})$: the only center precondition of π^L, $T_1 = l_1$, is satisfied in the center state $\mathsf{center}(s^{\mathcal{F}}) = \{T_1 = l_1, T_2 = l_4\}$. For the leaf state $\{p_1 = l_2\}$, however, there is no $\pi^C(s^{\mathcal{F}})$-compliant leaf path because we would need the center precondition $T_1 = l_2$, which is not true anywhere along $\pi^C(s^{\mathcal{F}})$. In particular, the path $\langle \text{load}(T_1, p_1, l_1), \text{unload}(T_1, p_1, l_2)\rangle$ is not $\pi^C(s^{\mathcal{F}})$-compliant for that reason. Similarly for the leaf states $\{p_i = l_2\}$, $\{p_i = l_3\}$, and $\{p_i = l_4\}$. These leaf states are not reachable given $\pi^C(s^{\mathcal{F}})$, so their price in $s^{\mathcal{F}}$ is ∞. The path $\langle \text{unload}(T_1, p_1, l_1)\rangle$ is π^C-compliant, but is not actually a leaf path because it is not applicable to $\mathcal{I}$: its precondition $p_1 = T_1$ on the leaf itself is not satisfied initially.

We next define the compliant-path graph, which captures all compliant paths along a center path and serves as a basis to obtain the pricing function.

Definition 8 (Compliant-Path Graph)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a star factoring with center* C *and leaves* $\mathcal{L}$*, and* $\pi^C = \langle a^C_1, \dots, a^C_n\rangle$ *a center path traversing center states* $\langle s^C_0, \dots, s^C_n\rangle$*. The* π^C-compliant-path graph *for a leaf* $L \in \mathcal{L}$*, denoted* $\mathrm{CPG}_\Pi(\pi^C, L)$*, is the arc-labeled weighted directed graph whose vertices are* $\{s^L_t \mid s^L \in S^L, 0 \leq t \leq n\}$*, and whose arcs are:*

(i) $s^L_t \xrightarrow{a^L} q^L_t$ *with weight* $\mathsf{cost}(a^L)$ *whenever* $s^L, q^L \in S^L$ *and* $a^L \in \mathcal{A}^L_{\not C}$ *are such that* $s^C_t \models \mathsf{pre}(a^L)[C]$*,* $s^L \models \mathsf{pre}(a^L)[L]$*, and* $s^L[\![a^L]\!] = q^L$*.*

(ii) $s^L_t \xrightarrow{0} q^L_{t+1}$ *with weight* 0 *whenever* $s^L, q^L \in S^L$ *are such that* $s^L \models \mathsf{pre}(a^C_{t+1})[L]$ *and* $s^L[\![a^C_{t+1}]\!] = q^L$*.*

Item (i) concerns transitions *within* each time step t, i. e., the graph captures how L—*only* L, not the center or anything else—can be moved given the center preconditions provided by s^C_t. Recall here that the center actions $\mathcal{A}^C$ are not disjoint from the leaf actions $\mathcal{A}^L$. Within time steps, we only consider the leaf-only actions, $\mathcal{A}^L_{\not C}$.

Item (ii) concerns transitions *across* time steps, from t to $t+1$, compliant with the center action a^C_t. Leaf states s^L_t at step t survive (have a transition to $t+1$) only if they comply with the leaf precondition required by a^C_t, and they get transformed to leaf states q^L_{t+1} at step $t+1$ through the effect of a^C_t on L. Note that, if a^C_t has no precondition on L (e. g., like in the example with $\mathcal{F}_1$), then all leaf states s^L_t survive. If a^C_t has no effect on L, then the surviving leaf states remain the same at $t+1$, i. e., the item (ii) transitions have the form $s^L_t \xrightarrow{0} s^L_{t+1}$.

The arc weights capture the cost incurred for L, i. e., the cost of the π^C-compliant leaf paths of L. Within time steps, for the actions moving only L, these are just the action costs. Across time steps, where the center moves (which may or may not move L as a side effect), the arc weight is 0 because these costs are accounted for on the center path itself.

In short, $\text{CPG}_\Pi(\pi^C, L)$ captures all ways in which leaf paths for L can be scheduled alongside π^C. The π^C-compliant paths correspond exactly to the paths from $\mathcal{I}[L]_0$, i. e., from the vertex representing L's initial state in $\text{CPG}_\Pi(\pi^C, L)$, to the vertices of the last layer n, where n is the length of π^C. Consequently, we define the pricing function for π^C and L through the *cheapest* such paths in $\text{CPG}_\Pi(\pi^C, L)$.

Keep in mind that a pricing function, and thereby the compliant-path graph, does not represent a commitment, but a set of options, one of which will be committed to later on. In the example with $\mathcal{F}_1$, *if* we later on choose to load p_1 into T_1, the cost for doing so will be 1. The commitments will only be made once we reach the goal. Namely, a *decoupled goal state* is one whose center state is a *center goal state*, and where every leaf has a finite-price *leaf goal state*. Given a center path π^C leading to a decoupled goal state, we can extract a (global) plan π for the input task by augmenting π^C with π^C-compliant leaf goal paths, i. e., compliant leaf paths that end in a leaf goal state. In fact, pricing functions allow to extract a plan π *optimal subject to using exactly the center action subsequence* π^C. This is so because every plan decomposes into a center path augmented with compliant leaf paths, and the pricing functions keep track of the *cheapest* compliant leaf paths.

A variant of decoupled search is obtained by replacing the pricing functions with *reachability functions*, that distinguish only if a leaf state is reachable ($\mathsf{prices}(s^\mathcal{F})[s^L] < \infty$), or not ($\mathsf{prices}(s^\mathcal{F})[s^L] = \infty$). This allows to preserve completeness, but does not allow to preserve optimality. It is of advantage in practice because reachability functions can be computed more efficiently, and as they make less distinctions so reduce the size of the decoupled state space. Reachability functions are equivalent to pricing functions in the modified task where all leaf action costs are set to 0, so we will specify the more general pricing functions only.

Example 1. *Consider again the example with $\mathcal{F}_1$. Denote the empty center path by π_0^C, and the corresponding decoupled state by $s_0^{\mathcal{F}}$. The outgoing transitions of a decoupled state are given by those center actions whose center precondition is satisfied, and whose precondition on each leaf has a finite price. In $s_0^{\mathcal{F}}$, these are drive(T_1, l_1, l_2) and drive(T_2, l_4, l_3).*

Denote the center path extended with drive(T_1, l_1, l_2) by π_1^C, and the outcome decoupled state by $s_1^{\mathcal{F}}$. Then $\mathsf{center}(s_1^{\mathcal{F}}) = \{T_1 = l_2, T_2 = l_4\}$*. The prices* $\mathsf{prices}(s_1^{\mathcal{F}})$ *are 0 for $\{p_i = l_1\}$ and 1 for $\{p_i = T_1\}$, as these were the prices in $s_0^{\mathcal{F}}$, and no cheaper compliant paths become available in $s_1^{\mathcal{F}}$. The only change in* $\mathsf{prices}(s_1^{\mathcal{F}})$ *is that the leaf states $\{p_i = l_2\}$ get price 2, accounting for the leaf paths $\langle$load(T_1, p_i, l_1), unload$(T_1, p_i, l_2)\rangle$, which are π_1^C-compliant because they can be scheduled along π_1^C in the form $\langle$load(T_1, p_i, l_1), drive(T_1, l_1, l_2), unload$(T_1, p_i, l_2)\rangle$. The compliant-path graph for $s_1^{\mathcal{F}}$ is shown in Figure 3.3.*

Now say we obtain π_2^C and $s_2^{\mathcal{F}}$ from π_1^C and $s_1^{\mathcal{F}}$ by moving T_1 to l_3. Then $\mathsf{center}(s_2^{\mathcal{F}}) = \{T_1 = l_3, T_2 = l_4\}$*, and we have the new finite prices* $\mathsf{prices}(s_2^{\mathcal{F}})[\{p_i = l_3\}] = 2$ *thanks to the π_2^C-compliant path $\pi_2^L := \langle$load(T_1, p_i, l_1), unload$(T_1, p_i, l_3)\rangle$. Say further that we obtain π_3^C and $s_3^{\mathcal{F}}$ from π_2^C and $s_2^{\mathcal{F}}$ by driving T_2 to l_3. Then* $\mathsf{prices}(s_3^{\mathcal{F}})[\{p_i = T_2\}] = 3$ *thanks to the π_3^C-compliant paths π_3^L where load(T_2, p_i, l_3) is appended to π_2^L. If next we choose the successor of $s_3^{\mathcal{F}}$ via drive(T_2, l_3, l_4), obtaining $s_4^{\mathcal{F}}$ with* $\mathsf{center}(s_2^{\mathcal{F}}) = \{T_1 = l_3, T_2 = l_4\}$ *and π_4^C by appending drive(T_2, l_3, l_4) to π_3^C, we have reached a decoupled goal state, where* $\mathsf{prices}(s_4^{\mathcal{F}})[\{p_i = l_4\}] = 4$*. We obtain a global plan by scheduling $\pi_4^L := \langle$load(T_1, p_i, l_1), unload(T_1, p_i, l_3), load(T_2, p_i, l_3), unload$(T_2, p_i, l_4)\rangle$ for each leaf alongside π_4^C, yielding the 16-step global plan that loads all packages onto T_1, drives T_1 to l_3 and unloads all packages; then drives T_2 to l_3, loads the packages into T_2, drives it back to l_4 and unloads all packages.*

Say finally that we continue on from this decoupled goal state, obtaining π_5^C and $s_5^{\mathcal{F}}$ from π_4^C and $s_4^{\mathcal{F}}$ by moving T_1 to l_4. Then $\mathsf{center}(s_5^{\mathcal{F}}) = \{T_1 = l_4, T_2 = l_4\}$*. The compliant leaf paths π_4^L supporting $\{p_i = l_4\}$ in $s_4^{\mathcal{F}}$, to yield the price of 4, are superseded by the new π_5^C-compliant (but not π_4^C-compliant) paths $\pi_5^L := \langle$load(T_1, p_i, l_1), unload$(T_1, p_i, l_4)\rangle$. This decreases the prices to* $\mathsf{prices}(s_5^{\mathcal{F}})[\{p_i = l_4\}] = 2$*. Observe that we now get a 11-step global plan, i. e., a plan* better *than that of the decoupled goal state $s_4^{\mathcal{F}}$ we passed through on the way to $s_5^{\mathcal{F}}$. Intuitively, decoupled goal states have* leaf-goal prices*, the cost of "buying" compliant leaf goal paths. The center paths account only for the center, not for the leaves, so the larger path costs of decoupled-goal-state descendants may be counteracted by cheaper leaf-goal prices, as in this example. We will show in Chapter 4 how optimal search algorithms can deal with this via a simple transformation.*

$$\begin{tikzcd}
(p_i = T)_1 \arrow[rr, bend left, leftrightarrow, "(\text{un})\text{load}(p_i, l_2);\, w = 1"] & (p_i = l_1)_1 & (p_i = l_2)_1 & (p_i = l_3)_1 & (p_i = T_2)_1 \\
(p_i = T_1)_0 \arrow[u, "0"'] \arrow[r, bend right, leftrightarrow, "(\text{un})\text{load}(p_i, l_1);\, w = 1"'] & (p_i = l_1)_0 \arrow[u, "0"'] & (p_i = l_2)_0 \arrow[u, "0"'] & (p_i = l_3)_0 \arrow[u, "0"'] & (p_i = T_2)_0 \arrow[u, "0"']
\end{tikzcd}$$

Figure 3.3: The compliant-path graphs for $\pi^C = \langle \text{drive}(T_1, l_1, l_2) \rangle$ in the example when using the factoring $\mathcal{F}_1$.

$$\begin{tikzcd}
(T_1 = l_1)_1 \arrow[r, leftrightarrow, "\text{drive};\, w = 1"] & (T_1 = l_2)_1 \arrow[r, leftrightarrow, "\text{drive};\, w = 1"] & (T_1 = l_3)_1 \arrow[r, leftrightarrow, "\text{drive};\, w = 1"] & (T_1 = l_4)_1 \\
(T_1 = l_1)_0 \arrow[u, "0"'] \arrow[r, leftrightarrow, "\text{drive};\, w = 1"'] & (T_1 = l_2)_0 \arrow[r, leftrightarrow, "\text{drive};\, w = 1"'] & (T_1 = l_3)_0 \arrow[r, leftrightarrow, "\text{drive};\, w = 1"'] & (T_1 = l_4)_0
\end{tikzcd}$$

Figure 3.4: The compliant-path graph of $L_1 = \{T_1\}$ for $\pi^C = \langle \text{load}(T_1, p_1, l_1) \rangle$ in the example when using the factoring $\mathcal{F}_2$.

Example 2. *Consider now our example with the alternative factoring $\mathcal{F}_2$. The initial decoupled state $s_0^{\mathcal{F}}$ has $\pi^C(s_0^{\mathcal{F}}) = \langle\rangle$,* center$(s_0^{\mathcal{F}}) = \{p_1 = l_1, p_2 = l_1, p_3 = l_1\}$*, and the following prices:* prices$(s_0^{\mathcal{F}})[\{T_1 = l_i\}] = i - 1$ *and* prices$(s_0^{\mathcal{F}})[\{T_2 = l_i\}] = 4 - i$*. All leaf states for both trucks are reachable, since the drive actions do not have a center precondition. The prices correspond to the distance of the respective truck to its initial position.*

Say we obtain $s_1^{\mathcal{F}}$ by loading p_1 into T_1 in $s_0^{\mathcal{F}}$. Then for $s_1^{\mathcal{F}}$, we obtain $\pi^C(s_1^{\mathcal{F}}) = \langle load(T_1, p_1, l_1) \rangle$, center$(s_1^{\mathcal{F}}) = \{p_1 = T_1, p_2 = l_1, p_3 = l_1\}$*, and the prices do not change. For T_2, this is because $load(T_1, p_1, l_1)$ does not have a leaf precondition on T_2, so the leaf paths compliant with $\pi^C(s_0^{\mathcal{F}})$ remain compliant and no cheaper paths become available. For T_1, observe that the load action restricts the set of cheapest compliant paths to only $\pi^L = \langle\rangle$ (although, e. g., the path $\langle drive(T_1, l_1, l_2), drive(T_1, l_2, l_1) \rangle$ is $\pi^C(s_1^{\mathcal{F}})$-compliant, it would lead to a price of 2 for $\{T_1 = l_1\}$, so it does not affect the prices). Thus, in $s_1^{\mathcal{F}}$ the leaf states of $L_1 = \{T_1\}$ need to be reached again from $(T_1 = l_1)_1$, leading to the same prices as before, because they extend the empty compliant leaf path π^L. The compliant-path graph for $s_1^{\mathcal{F}}$ and $\{T_1\}$ is shown in Figure 3.4.*

3.2.2 The Transition System

We are now ready to define the decoupled state space:

Definition 9 (Decoupled State Space)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π *with center* C *and leaves* $\mathcal{L}$*. A* decoupled state $s^\mathcal{F}$ *is a triple* $\langle \pi^C(s^\mathcal{F}), \mathsf{center}(s^\mathcal{F}), \mathsf{prices}(s^\mathcal{F})\rangle$ *where* $\pi^C(s^\mathcal{F})$ *is a center path,* $\mathsf{center}(s^\mathcal{F})$ *is a center state, and* $\mathsf{prices}(s^\mathcal{F})$ *is a* pricing function, $\mathsf{prices}(s^\mathcal{F}) : S^\mathcal{L} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$*, mapping each leaf state to a non-negative* price*. The* decoupled state space *is a labeled transition system* $\Theta^\mathcal{F}_\Pi = \langle \mathcal{S}^\mathcal{F}, \mathcal{A}^C, \mathsf{cost}|_{\mathcal{A}^C}, \mathcal{T}^\mathcal{F}, \mathcal{I}^\mathcal{F}, \mathcal{S}^\mathcal{F}_\mathcal{G}\rangle$ *as follows:*

(i) $\mathcal{S}^\mathcal{F}$ *is the set of all decoupled states.*

(ii) The transition labels are the center actions $\mathcal{A}^C$*.*

(iii) The cost function is that of Π*, restricted to* $\mathcal{A}^C$*.*

(iv) $\mathcal{T}^\mathcal{F}$ *contains a transition* $s^\mathcal{F} \xrightarrow{a^C} t^\mathcal{F} \in \mathcal{T}^\mathcal{F}$ *whenever* $a^C \in \mathcal{A}^C$ *and* $s^\mathcal{F}, t^\mathcal{F}$ *are such that:*

1. $\pi^C(s^\mathcal{F}) \circ \langle a^C \rangle = \pi^C(t^\mathcal{F})$*;*
2. $\mathsf{center}(s^\mathcal{F}) \models \mathsf{pre}(a^C)[C]$*;*
3. $\mathsf{center}(s^\mathcal{F})[\![a^C]\!] = \mathsf{center}(t^\mathcal{F})$*;*
4. *for every* $L \in \mathcal{L}$ *where* $\mathsf{vars}(\mathsf{pre}(a^C)) \cap L \neq \emptyset$*, there exists* $s^L \in S^L$ *s.t.* $s^L \models \mathsf{pre}(a^C)[L]$ *and* $\mathsf{prices}(s^\mathcal{F})[s^L] < \infty$*; and*
5. *for every leaf* $L \in \mathcal{L}$ *and leaf state* $s^L \in S^L$*,* $\mathsf{prices}(t^\mathcal{F})[s^L]$ *is the cost of a cheapest path from* $\mathcal{I}[L]_0$ *to* s^L_n *in* $\mathrm{CPG}_\Pi(\pi^C(t^\mathcal{F}), L)$*, where* $n := |\pi^C(t^\mathcal{F})|$*.*

(v) $\mathcal{I}^\mathcal{F}$ *is the* decoupled initial state*, where* $\mathsf{center}(\mathcal{I}^\mathcal{F}) := \mathcal{I}[C]$*,* $\pi^C(\mathcal{I}^\mathcal{F}) := \langle\rangle$*, and, for every leaf* $L \in \mathcal{L}$ *and leaf state* $s^L \in S^L$*,* $\mathsf{prices}(\mathcal{I}^\mathcal{F})[s^L]$ *is the cost of a cheapest path from* $\mathcal{I}[L]_0$ *to* s^L_0 *in* $\mathrm{CPG}_\Pi(\langle\rangle, L)$*.*

(vi) $\mathcal{S}^\mathcal{F}_\mathcal{G}$ *are the* decoupled goal states $s^\mathcal{F}_G$*, where* $\mathsf{center}(s^\mathcal{F}_G) \models \mathcal{G}[C]$ *is a center goal state and, for every* $L \in \mathcal{L}$*, there exists a reached leaf goal state* $s^L \in S^L$*, i. e.,* $s^L \models \mathcal{G}[L]$ *and* $\mathsf{prices}(s^\mathcal{F}_G)[s^L] < \infty$*.*

For a transition $s^\mathcal{F} \xrightarrow{a^C} t^\mathcal{F} \in \mathcal{T}^\mathcal{F}$, we denote the decoupled successor state $t^\mathcal{F}$ of $s^\mathcal{F}$ via a^C by $s^\mathcal{F}[\![a^C]\!]$. A decoupled state $s^\mathcal{F}$ satisfies a (partial) assignment p to $\mathcal{V}$, denoted $s^\mathcal{F} \models p$, iff $\mathsf{center}(s^\mathcal{F}) \models p[C]$ and for all $L \in \mathcal{L}$ there exists a leaf state s^L such that $s^L \models p[L]$ and $\mathsf{prices}(s^\mathcal{F})[s^L] < \infty$. A center action $a^C \in \mathcal{A}^C$ is applicable in a decoupled state $s^\mathcal{F}$ iff $s^\mathcal{F} \models \mathsf{pre}(a^C)$.

Note that $\Theta^\mathcal{F}_\Pi$ is infinite, for two reasons: (1) decoupled states contain center paths, of which there are infinitely many unless the center cannot move in circles; (2) there are infinitely many pricing functions. We show in Chapter 3.4 that finiteness can be attained by a simple *dominance pruning* method.

We refer to paths $\pi^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{F}}$ as *decoupled paths*. Such a path is a *decoupled plan* if it ends in a decoupled goal state. The notions of path cost and solutions, for decoupled states as well as for $\Theta_{\Pi}^{\mathcal{F}}$, are inherited from labeled transition systems. However, we also require a specialized notion of cost and optimality, *augmented cost/optimality*, different from the standard additive-cost notion. This is because, when applied to $\Theta_{\Pi}^{\mathcal{F}}$, the standard notion accounts only for the center-action costs. Augmented cost/optimality accounts also for the leaf-goal prices, and, as we shall see, corresponds exactly to optimality in the input planning task.

Definition 10 (Augmented Cost & Optimality)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. For a decoupled goal state* $s_G^{\mathcal{F}}$*, its* leaf-goal price*, denoted* $\mathsf{gprice}(s_G^{\mathcal{F}})$*, is the sum over its minimal leaf goal state prices:*

$$\mathsf{gprice}(s_G^{\mathcal{F}}) := \sum_{L \in \mathcal{L}} \min\{\mathsf{prices}(s_G^{\mathcal{F}})[s^L] \mid s^L \in S^L, s^L \models \mathcal{G}[L]\}$$

For a decoupled path $\pi^{\mathcal{F}}$ *ending in* $s_G^{\mathcal{F}}$*, its* augmented cost *is:*

$$\mathsf{augCost}(\pi^{\mathcal{F}}) := \mathsf{cost}(\pi^{\mathcal{F}}) + \mathsf{gprice}(s_G^{\mathcal{F}}).$$

A solution for a decoupled state $s^{\mathcal{F}}$ *is* augmented-optimal *if its augmented cost is minimal among all solutions for* $s^{\mathcal{F}}$*.*

In the definition of leaf-goal prices, recall that each leaf factor may contain several state variables, and hence may have more than one leaf goal state. So we need to minimize over these.

Intuitively, the augmented cost of a decoupled path is its own cost, plus that of the compliant leaf goal paths it needs to be augmented with to reach the global goal. When referring to optimality in decoupled search, from now on we will always mean augmented optimality. To avoid clumsy wording, we will sometimes drop the word "augmented" and simply talk about optimality.

From any decoupled plan $\pi^{\mathcal{F}}$ for a state $s^{\mathcal{F}}$ we can obtain a global plan for any goal member state s of $s^{\mathcal{F}}$ by extending $\pi^C(s^{\mathcal{F}}) \circ \pi^{\mathcal{F}}$ with the cheapest compliant leaf paths ending in $s[L]$ for all leaves. Herein, we are only interested in the cheapest global plan:

Definition 11 (Global Plan)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. Let* $\pi^{\mathcal{F}}$ *be a decoupled plan for a decoupled state* $s^{\mathcal{F}}$*. The* global plan $\pi^G(\pi^{\mathcal{F}}, s^{\mathcal{F}})$ *for* $\pi^{\mathcal{F}}$ *and* $s^{\mathcal{F}}$ *is a sequence of actions such that* $\pi^C(\pi^G(\pi^{\mathcal{F}}, s^{\mathcal{F}})) = \pi^C(s^{\mathcal{F}}) \circ \pi^{\mathcal{F}}$ *is the underlying center path, which is augmented with the cheapest compliant leaf paths for all* $L \in \mathcal{L}$*, such that* $\mathsf{cost}(\pi^G(\pi^{\mathcal{F}}, s^{\mathcal{F}})) = \mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{augCost}(\pi^{\mathcal{F}})$*. For decoupled goal states* $s_G^{\mathcal{F}}$*, we define the global plan as* $\pi^G(s_G^{\mathcal{F}}) := \pi^G(\langle\rangle, s_G^{\mathcal{F}})$*.*

The global plan can be computed efficiently by inserting, between every pair of consecutive center actions, the leaf actions that belong to the cheapest compliant leaf path on the respective layer of the compliant-path graph.

We will now illustrate the pruning power of decoupled search on a variant of our example in which we scale the number of packages.

Example 3. *Consider a scaling variant of our example, where one truck T and n packages p_i move along a line $l_1, \dots, l_m$ of length m, the truck and all packages starting in l_1, the goal being to transport all packages to l_m. The standard state space has $m(m+1)^n$ reachable states. By contrast, using the factoring with $C = \{T\}$ and $\mathcal{L} = \{\{p_1\}, \dots, \{p_n\}\}$, the decoupled state space has only $\frac{m(m+1)}{2}$ reachable decoupled states. This does not even depend on the number of packages.*[2]

Intuitively, the reason is that pricing function changes happen synchronously across all leaves, as a function of truck moves, to the effect that we need to keep track only of the subsequence of locations visited by the truck. In detail: the initial decoupled state allows each package to be loaded, so each has the price 0 for $\{p_i = l_1\}$ and 1 for $\{p_i = T\}$. After moving to l_2, each package gets the additional price 2 for $\{p_i = l_2\}$. Now there are two choices, moving back to l_1 which yields the same center state as before but with the additional $\{p_i = l_2\}$ prices, or moving ahead to l_3 which yields the new prices 2 for $\{p_i = l_3\}$. In this manner, for each location l_i, the new decoupled states are the single one where the truck reaches l_i for the first time, plus the $i-1$ ones reached by going back to $l_{i-1}, \dots, l_1$. This yields the overall count $\sum_{i=1}^{m} i = \frac{m(m+1)}{2}$.

Each decoupled state has size $1 + n(m+1)$ (truck position; n packages, each with $m+1$ leaf states), so the overall state-space representation size is $O(nm^3)$.

Remarkably, while the scaling example is trivial, as we will detail in Chapter 6, it exponentially separates decoupled search from most previous search reduction methods.

3.3 Correctness

We now prove soundness, completeness, and optimality of $\Theta_\Pi^\mathcal{F}$ relative to the input planning task Π. We do so via a characterization of decoupled states in terms of their *hypercubes*:

Definition 12 (Hypercube). *Let Π be a planning task and $\mathcal{F}$ a factoring for Π with center C and leaves $\mathcal{L}$. Then a state s of Π is a* member state *of a decoupled state $s^\mathcal{F}$, if $s[C] = \mathsf{center}(s^\mathcal{F})$ and, for all leaves $L \in \mathcal{L}$, $\mathsf{prices}(s^\mathcal{F})[s[L]] < \infty$. The* price *of s in $s^\mathcal{F}$ is $\mathsf{price}(s^\mathcal{F}, s) := \sum_{L \in \mathcal{L}} \mathsf{prices}(s^\mathcal{F})[s[L]]$. The* hypercube *of $s^\mathcal{F}$, denoted $[s^\mathcal{F}]$, is the set of all member states of $s^\mathcal{F}$.*

[2]We assume here a form of duplicate state pruning, which is only formally introduced in Section 3.4, where a new decoupled state is pruned if there exists an already visited state with the same center state and pricing function. Thus, the center path reaching the states is ignored for pruning.

In other words, $[s^{\mathcal{F}}]$ is the product of the reached leaf states across leaf factors. This is naturally viewed as a hypercube whose dimensions are the leaf factors L.

Observe that $s^{\mathcal{F}}$ is a decoupled goal state if and only if its hypercube $[s^{\mathcal{F}}]$ contains a goal state of Θ_Π. Furthermore, it is not difficult to prove that $[s^{\mathcal{F}}]$ captures exactly those states of Θ_Π reachable using the center path $\pi^C(s^{\mathcal{F}})$:

Lemma 1. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Let $s^{\mathcal{F}}$ be a reachable decoupled state in $\Theta_\Pi^{\mathcal{F}}$. Then:*

(i) $[s^{\mathcal{F}}]$ is exactly the set of states s for which there exists a path π, from $\mathcal{I}$ to s in Θ_Π, where $\pi^C(\pi) = \pi^C(s^{\mathcal{F}})$.

(ii) For every $s \in [s^{\mathcal{F}}]$, the cost of a cheapest such path π is $\mathsf{cost}(\pi^C(s^{\mathcal{F}}))+\mathsf{price}(s^{\mathcal{F}}, s)$.

Our proof, given in Appendix A.1, is via notions of *embedded states*, and *embedded transitions*, linking member states, and member-state transitions, to decoupled states. Embedded states $\widehat{s}$ are in one-to-one correspondence with member states s, but replace the value assignment to each L with the respective vertex in the compliant-path graph for L. Embedded transitions $\widehat{s} \xrightarrow{a} \widehat{t}$ capture member state transitions $s \xrightarrow{a} t$ in the compliant-path graphs, and hence in the decoupled state space. One can think about this as capturing the decoupled state space in terms of atomic transition-addition steps (which relates to unfolding-prefix extensions, discussed in Chapter 6.1).

Lemma 1 follows directly from the correspondence between member state transitions, embedded state transitions, and the decoupled state space. The same correspondence also shows that:

Lemma 2. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Let s be a reachable state in Π, and let π be a path reaching s. Then there exists a reachable decoupled state $s^{\mathcal{F}}$ in $\Theta_\Pi^{\mathcal{F}}$ so that $s \in [s^{\mathcal{F}}]$, and $\pi^C(\pi) = \pi^C(s^{\mathcal{F}})$.*

The hypercubes in $\Theta_\Pi^{\mathcal{F}}$ capture reachability exactly, in the sense stated by Lemmas 1 and 2. Note that the hypercubes of distinct decoupled states may overlap.

Correctness relative to the input planning task Π now follows directly:

Theorem 1 (Soundness, Optimality Subject to Center Path)**.** *Let Π be a planning task, $\mathcal{F}$ a factoring, and $\pi^{\mathcal{F}}$ a solution for $\Theta_\Pi^{\mathcal{F}}$ ending in $s^{\mathcal{F}}$. Then there is a plan π for Π where $\mathsf{cost}(\pi) = \mathsf{augCost}(\pi^{\mathcal{F}})$, where $\pi^C(\pi) = \pi^C(s^{\mathcal{F}})$, and where π is cheapest among all plans for Π sharing that same center-action subsequence.*

Proof. As $s^{\mathcal{F}}$ is a decoupled goal state, there exists a member goal state in $[s^{\mathcal{F}}]$. Let s be a cheapest such state, i. e., where $s = \text{argmin}_{t \in [s^{\mathcal{F}}]} \mathsf{price}(s^{\mathcal{F}}, t)$. Using Lemma 1, we can obtain a plan π ending in s, where $\mathsf{cost}(\pi) = \mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{price}(s^{\mathcal{F}}, s)$. By construction, this cost is equal to $\mathsf{augCost}(\pi^{\mathcal{F}})$.

Let now π' be any plan using $\pi^C(s^{\mathcal{F}})$, ending in goal state s'. By Lemma 1 (i), $s' \in [s^{\mathcal{F}}]$. So, by construction, $\mathsf{price}(s^{\mathcal{F}}, s') \geq \mathsf{price}(s^{\mathcal{F}}, s)$. By Lemma 1 (ii) applied to s', $\mathsf{cost}(\pi') \geq \mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{price}(s^{\mathcal{F}}, s')$. Hence $\mathsf{cost}(\pi') \geq \mathsf{cost}(\pi)$, concluding the argument. □

Theorem 2 (Completeness, Global Optimality)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. If* Π *is solvable, then so is* $\Theta^{\mathcal{F}}_{\Pi}$*. If* $\pi^{\mathcal{F}}$ *is an augmented-optimal solution to* $\Theta^{\mathcal{F}}_{\Pi}$*, then* $\mathsf{augCost}(\pi^{\mathcal{F}})$ *equals the cost of an optimal plan for* Π*.*

Proof. Assume that Π is solvable. Let π be an optimal plan for Π, ending in goal state s. With Lemma 2, there exists a reachable decoupled state $s^{\mathcal{F}}$ s.t. $s \in [s^{\mathcal{F}}]$. Let $\pi^{\mathcal{F}}$ be the decoupled path reaching $s^{\mathcal{F}}$ from $\mathcal{I}^{\mathcal{F}}$. As s is a goal state, $s^{\mathcal{F}}$ is a decoupled goal state, so $\pi^{\mathcal{F}}$ is a solution for $\Theta^{\mathcal{F}}_{\Pi}$.

With Lemma 1 (ii), we have that $\mathsf{cost}(\pi) \geq \mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{price}(s^{\mathcal{F}}, s)$. By definition, $\mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{price}(s^{\mathcal{F}}, s) \geq \mathsf{augCost}(\pi^{\mathcal{F}})$. Thus $\mathsf{cost}(\pi) \geq \mathsf{augCost}(\pi^{\mathcal{F}})$, i. e., $\Theta^{\mathcal{F}}_{\Pi}$ has a solution whose augmented cost is at most that of π. The second part of the claim now follows directly with Theorem 1. □

Together, Theorems 1 and 2 show that (augmented-optimal) search in $\Theta^{\mathcal{F}}_{\Pi}$ is a form of (optimal) planning for Π. Furthermore, given a solution $\pi^{\mathcal{F}}$ for $\Theta^{\mathcal{F}}_{\Pi}$, the corresponding plan π for Π as per Theorem 1 can be constructed by selecting, for every leaf factor, a cheapest leaf goal state s^L, and a cheapest path to s^L in the respective compliant-path graph. This construction is low-order polynomial-time in the size of Π and the size of the compliant path graphs along $\pi^{\mathcal{F}}$.

We have already seen, in our scaling example (Example 3), that the decoupled state space can be exponentially smaller than the standard state space. We next examine possible blow-ups and how to tackle them; then we show how to design (augmented-optimal) search algorithms for $\Theta^{\mathcal{F}}_{\Pi}$. We then show that our method is exponentially separated from all previous search reduction techniques.

3.4 Decoupled State-Space Size and Pruning

As pointed out before, the decoupled state space is, in general, infinite due to (1) different center paths, and due to (2) different pricing functions. Of these, (1) would be easy to tackle by considering decoupled states to be equivalent if they disagree only on the center path. However, as we now show, due to (2) there can be infinitely many reachable non-equivalent decoupled states. Furthermore, even in finite cases, the number of reachable non-equivalent decoupled states can blow up exponentially relative to the standard state space.

Fortunately, both can be tackled using pruning techniques. Finiteness can be achieved through a simple *dominance pruning* technique that we introduce in Chapter 3.4.1.

Blow-ups can be avoided through a more powerful *hypercube pruning* technique that we introduce in Chapter 3.4.2. The latter guarantees that *there can never be more reachable decoupled states than reachable standard states*. The downside of hypercube pruning is that it incurs solving a co-**NP**-complete subproblem for every reached decoupled state. Empirically, as we will show in Chapter 7.4 on standard planning benchmarks, hypercube pruning incurs a large computational overhead; reduces search space size only marginally relative to dominance pruning; and even without hypercube pruning, the number of generated decoupled states is typically significantly smaller than that of standard states. We introduce hypercube pruning mostly for its theoretical merit.

3.4.1 Finiteness and Dominance Pruning

Non-reachable states are not of interest in practice, so we focus on reachable states only. We denote by $\Theta_{\Pi}^{\mathcal{RF}}$ the sub-space of $\Theta_{\Pi}^{\mathcal{F}}$ containing only those decoupled states reachable from $\mathcal{I}^{\mathcal{F}}$.

For reachability functions, as their number is finite, so is the number of non-equivalent decoupled states in $\Theta_{\Pi}^{\mathcal{RF}}$ (and even in $\Theta_{\Pi}^{\mathcal{F}}$). With pricing functions, however, the center may commit the leaves to provide preconditions, causing the leaf state prices to increase. Repeated commitments can cause the prices to diverge:

Example 4. *Consider a variant of our example with one package p, two trucks T_1, T_2, initial positions as before, and a factoring with $C = \{p\}$ and $\mathcal{L} = \{\{T_1\}, \{T_2\}\}$. Say we continue the decoupled path $\mathcal{I}^{\mathcal{F}} \xrightarrow{load(T_2,p,l_1)} s_1^{\mathcal{F}}$ with the center action unload(T_2, p, l_1). Then the outcome decoupled state $s_2^{\mathcal{F}}$ has the same center state as $\mathcal{I}^{\mathcal{F}}$, $\{p = l_1\}$, but the pricing function is different. The prices for T_2 in $\mathcal{I}^{\mathcal{F}}$ are 0 for $\{T_2 = l_4\}$, 1 for $\{T_2 = l_3\}$, 2 for $\{T_2 = l_2\}$, and 3 for $\{T_2 = l_1\}$. In $s_2^{\mathcal{F}}$, they are 3 for $\{T_2 = l_1\}$, 4 for $\{T_2 = l_2\}$, 5 for $\{T_2 = l_3\}$, and 6 for $\{T_2 = l_4\}$. If we continue loading/unloading the package in alternating locations on the map, then the prices for truck positions will keep increasing ad infinitum.*

Intuitively, the decoupled states along load/unload sequences as outlined get *worse*, as the prices get higher. This leads to the following simple definition of dominance:

Definition 13 (Dominance)**.** *Let Π be a planning task, $\mathcal{F}$ a factoring for Π. Let $s^{\mathcal{F}}, t^{\mathcal{F}}$ be decoupled states. We say that $t^{\mathcal{F}}$* dominates *$s^{\mathcal{F}}$, denoted $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$, iff $\mathsf{center}(t^{\mathcal{F}}) = \mathsf{center}(s^{\mathcal{F}})$ and, for every leaf state $s^L \in S^{\mathcal{L}}$, $\mathsf{prices}(t^{\mathcal{F}})[s^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$.*

Note that equivalence is a special case of dominance, in that equivalent $s^{\mathcal{F}}$ and $t^{\mathcal{F}}$ dominate each other. Informally, if $t^{\mathcal{F}}$ dominates $s^{\mathcal{F}}$, then anything one can do in $s^{\mathcal{F}}$, one can do at least as well in $t^{\mathcal{F}}$. Formally, dominance is a simulation relation (e. g. [Henzinger et al., 1995]):

Proposition 1 (Correctness of Dominance Pruning)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a factoring, and* $s^{\mathcal{F}}, t^{\mathcal{F}}$ *decoupled states, where* $t^{\mathcal{F}}$ *dominates* $s^{\mathcal{F}}$*. Then, for every transition* $s^{\mathcal{F}} \xrightarrow{a^C} s_i^{\mathcal{F}}$ *in* $\Theta_{\Pi}^{\mathcal{F}}$*,* $t^{\mathcal{F}} \xrightarrow{a^C} t_i^{\mathcal{F}}$ *also is a transition in* $\Theta_{\Pi}^{\mathcal{F}}$*, and* $t_i^{\mathcal{F}}$ *dominates* $s_i^{\mathcal{F}}$*.*

Proof. The center precondition of a^C is trivially true in $t^{\mathcal{F}}$, and its leaf preconditions have finite prices in $t^{\mathcal{F}}$ because that is so already in $s^{\mathcal{F}}$. Hence $t^{\mathcal{F}} \xrightarrow{a^C} t_i^{\mathcal{F}}$ is a transition in $\Theta_{\Pi}^{\mathcal{F}}$. To see that $t_i^{\mathcal{F}}$ dominates $s_i^{\mathcal{F}}$, note that the compliant-path graph is extended with the same last time-step on both sides, so the cheaper prices in $t^{\mathcal{F}}$ can only lead to cheaper prices in $t_i^{\mathcal{F}}$. □

Note that dominance only considers the leaf state prices, but not the cost of the center path on which a decoupled state is reached. We next define augmented-cost dominance, that takes all costs into account:

Definition 14 (Augmented-Cost Dominance)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a factoring for* Π*, and* $s^{\mathcal{F}}, t^{\mathcal{F}}$ *be decoupled states. We say that* $t^{\mathcal{F}}$ augmented-cost dominates $s^{\mathcal{F}}$*, denoted* $s^{\mathcal{F}} \preceq_{\text{aug}} t^{\mathcal{F}}$*, iff* $\forall s \in [s^{\mathcal{F}}] : \mathsf{cost}(\pi^C(t^{\mathcal{F}})) + \mathsf{price}(t^{\mathcal{F}}, s) \leq \mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{price}(s^{\mathcal{F}}, s)$*.*

Like basic dominance, augmented-cost dominance generalizes equivalence and is a simulation relation:

Proposition 2 (Correctness of Augmented-Cost Dominance Pruning)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a factoring, and* $s^{\mathcal{F}}, t^{\mathcal{F}}$ *decoupled states, where* $t^{\mathcal{F}}$ *augmented-cost dominates* $s^{\mathcal{F}}$*. Then, for every transition* $s^{\mathcal{F}} \xrightarrow{a^C} s_i^{\mathcal{F}}$ *in* $\Theta_{\Pi}^{\mathcal{F}}$*,* $t^{\mathcal{F}} \xrightarrow{a^C} t_i^{\mathcal{F}}$ *also is a transition in* $\Theta_{\Pi}^{\mathcal{F}}$*, and* $s_i^{\mathcal{F}} \preceq_{\text{aug}} t_i^{\mathcal{F}}$*.*

Proof. Note that $s^{\mathcal{F}} \preceq_{\text{aug}} t^{\mathcal{F}}$ implies that $[s^{\mathcal{F}}] \subseteq [t^{\mathcal{F}}]$ and hence necessarily $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$ and for all $s^L \in S^{\mathcal{L}}$ where $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$ it holds that $\mathsf{prices}(t^{\mathcal{F}})[s^L] < \infty$. Otherwise, there would exist a member state s of $s^{\mathcal{F}}$ that is not reached in $t^{\mathcal{F}}$, so $\mathsf{price}(t^{\mathcal{F}}, s) = \infty$, in contradiction to $s^{\mathcal{F}} \preceq_{\text{aug}} t^{\mathcal{F}}$. Therefore, for every transition $s^{\mathcal{F}} \xrightarrow{a^C} s_i^{\mathcal{F}}$ there also is a transition $t^{\mathcal{F}} \xrightarrow{a^C} t_i^{\mathcal{F}}$.

It remains to show that $t_i^{\mathcal{F}}$ augmented-cost dominates $s_i^{\mathcal{F}}$. The cost of the decoupled paths reaching $s_i^{\mathcal{F}}$ and $t_i^{\mathcal{F}}$ is incremented by the same amount, namely $\mathsf{cost}(a^C)$. The leaf state prices in $t_i^{\mathcal{F}}$ can only be cheaper than the ones in $s_i^{\mathcal{F}}$ for the same reason stated in the proof of Proposition 1. Thus, it holds that $\forall s \in [s_i^{\mathcal{F}}] : \mathsf{cost}(\pi^C(t_i^{\mathcal{F}})) + \mathsf{price}(t_i^{\mathcal{F}}, s) \leq \mathsf{cost}(\pi^C(s_i^{\mathcal{F}})) + \mathsf{price}(s_i^{\mathcal{F}}, s)$. □

Simulation relations have previously been used for dominance pruning in standard state-space search (e. g. Hall et al., 2013; Torralba and Hoffmann, 2015). In particular, they can be used for optimality-preserving pruning when used only against previously visited states with equal or smaller *path costs*, i. e., reached with action sequences of equal or smaller cost. For the purpose of our discussion in this work, we distinguish the following three forms of pruning:

1. **Ancestor:** Compare any new state $s^{\mathcal{F}}$ to its ancestor states $t^{\mathcal{F}}$.

2. **Cheaper-Visited:** Compare any new state $s^{\mathcal{F}}$ to all previously visited states $t^{\mathcal{F}}$ whose path cost is at most that of $s^{\mathcal{F}}$.

3. **All-Visited:** Compare any new state $s^{\mathcal{F}}$ to all previously visited states $t^{\mathcal{F}}$.

In all cases, if $s^{\mathcal{F}}$ is (augmented-cost) dominated by one of the states $t^{\mathcal{F}}$ it is compared to, then $s^{\mathcal{F}}$ is pruned. As (augmented-cost) dominance generalizes equivalence, such pruning generalizes duplicate pruning over equivalent states.

Ancestor states are a special case of cheaper-visited states, and cheaper-visited states are a special case of visited states. Thus, the pruning methods become stronger in the order listed. Cheaper-visited pruning preserves optimality [Torralba and Hoffmann, 2015] for both types of dominance; all-visited pruning does only do so for augmented-cost dominance. In the latter, basic dominance is not sufficient because it ignores the cost of the path reaching a decoupled state, so the search might first visit a dominating but more costly state, pruning the optimal solutions. With augmented-cost dominance, the path cost is incorporated, so even checking against *all* visited states preserves optimality.

It turns out that, to avoid infinite reachable decoupled state spaces $\Theta_{\Pi}^{\mathcal{RF}}$, the weakest form of pruning considered herein—ancestor dominance pruning—already suffices.

For cases like in Example 4, i. e., where center actions do not affect leaves, this is trivial: as the prices increase monotonically, with ancestor dominance pruning every search path must stop as soon as the same center state is visited a second time. For example, the search path $\mathcal{I}^{\mathcal{F}} \xrightarrow{\text{load}(T_2,p,l_1)} s_1^{\mathcal{F}} \xrightarrow{\text{unload}(T_2,p,l_1)} s_2^{\mathcal{F}}$ is pruned, because $s_2^{\mathcal{F}}$ is dominated by $\mathcal{I}^{\mathcal{F}}$.

For more general factorings, with bidirectional dependencies between leaves and center, where the prices neither increase nor decrease monotonically, the proof is more involved:

Theorem 3 (Finiteness under Dominance Pruning). *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. Under ancestor dominance pruning,* $\Theta_{\Pi}^{\mathcal{RF}}$ *is finite.*

Proof (sketch). Consider the non-pruned paths $\pi^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{RF}}$. We prove that, under ancestor dominance pruning, every $\pi^{\mathcal{F}}$ is finite. As the branching factor is finite, this proves the claim.

Observe that the non-pruned paths necessarily are *descending*: intuitively, some prices along $\pi^{\mathcal{F}}$ must descend each time we encounter the same center state as otherwise the new state would be dominated by some previous state. Formally, consider the pricing functions along $\pi^{\mathcal{F}}$ as vectors v over $\mathbb{R}^{0+} \cup \{\infty\}$. Define a relation $\succ$ over such vectors by $v \succ v'$ iff there exists a vector position k so that $v[k] > v'[k]$. Say that a vector sequence $\vec{v} = v_0, v_1, v_2, \ldots$ is *descending* if, whenever $i < j$, $v_i \succ v_j$.

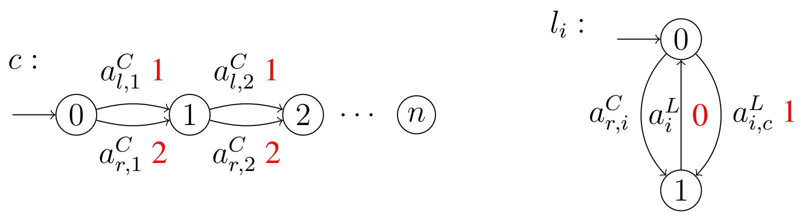


Figure 3.5: Illustration of the planning task used in the proof of Proposition 3.

Assume to the contrary that there is an infinite descending path $\pi^{\mathcal{F}}$. Then from an infinite sub-path with identical center states we can extract an infinite descending vector sequence $\vec{p}$ over $\mathbb{R}^{0+} \cup \{\infty\}$. More precisely, $\vec{p}$ is over $R \cup \{\infty\}$ where R contains all possible finite action-cost sums in Π.

It is easy to see (Proposition 14 in Appendix A.2) that R has no infinite descending sequence of 1-vectors (within each finite cost value, only a finite number of non-0 cost actions can be used). But what about N-vectors? Observe that, for $N > 1$, the relation $\succ$ is not a partial order. It is neither transitive, e. g., $(5, 5) \succ (4, 10)$ and $(4, 10) \succ (5, 9)$, but $(5, 5) \not\succ (5, 9)$; nor asymmetric, e. g., $(4, 5) \succ (5, 4)$ and $(5, 4) \succ (4, 5)$. Furthermore, $v \succ v'$ as soon as v' is strictly smaller anywhere; all other positions can increase by arbitrary amounts, e. g., $v \succ v'$ for $v = (5, 5)$ and $v' = (4, 1000)$. Nonetheless (Lemma 20 in Appendix A.2), if there is no infinite descending sequence of 1-vectors over some set $R \subseteq \mathbb{R}$, then there is no infinite descending sequence of N-vectors over $R \cup \{\infty\}$. Hence $\vec{p}$ must be finite, in contradiction, showing the claim. $\square$

Regarding the difference between basic $\preceq$ and augmented-cost dominance $\preceq_{\text{aug}}$, we next show that checking against all visited states with $\preceq_{\text{aug}}$ is strictly more powerful than checking only against cheaper-visited states with $\preceq$, and that it can actually lead to exponentially smaller reachable state spaces:

Proposition 3 (Pruning Power of Augmented-Cost Dominance)**.** *All-visited augmented-cost dominance pruning generalizes cheaper-visited dominance pruning and is exponentially separated from it.*

Proof. For the first part, observe that if there are two decoupled states $s^{\mathcal{F}}, t^{\mathcal{F}}$ such that $\text{cost}(\pi^C(s^{\mathcal{F}})) \geq \text{cost}(\pi^C(t^{\mathcal{F}}))$ and $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$ (which corresponds to cheaper visited pruning with $\preceq$), then it trivially follows that $s^{\mathcal{F}} \preceq_{\text{aug}} t^{\mathcal{F}}$.

For the second part, we give an example where pruning with $\preceq_{\text{aug}}$ against all visited states is exponentially separated from pruning with $\preceq$ against cheaper-visited states.

The task is illustrated in Figure 3.5 and employs the factoring $\mathcal{F}$ with center $C = \{c\}$ and n leaves $L_i = \{l_i\}$, for $1 \leq i \leq n$. There are two types of center actions, $a^C_{l,i}$ and $a^C_{r,i}$, both of which increment c by one. Additionally, the $a^C_{r,i}$ actions have a leaf effect $\{l_i = 1\}$. There are also two types of leaf-only actions, $a^L_i, a^L_{i,c}$, where a^L_i decrements l_i

without center precondition and $a^L_{i,c}$ increments it with precondition $\{c = i\}$. The action costs are highlighted in red in the illustration.

Say the decoupled state space is explored using a uniform-cost search. In $\mathcal{I}^\mathcal{F}$, the center state is $\{c = 0\}$, and the prices are $\mathsf{prices}(\mathcal{I}^\mathcal{F})[\{l_i = 0\}] = 0$ and $\mathsf{prices}(\mathcal{I}^\mathcal{F})[\{l_i = 1\}] = \infty$. In the $a^C_{l,1}$-successor $s^\mathcal{F}_{l,1} := \mathcal{I}^\mathcal{F}[\![a^C_{l,1}]\!]$, only the price of $\{l_1 = 1\}$ is updated to 1, all other prices remain the same. For the $a^C_{r,1}$-successor $s^\mathcal{F}_{r,1} := \mathcal{I}^\mathcal{F}[\![a^C_{r,1}]\!]$ in contrast, the price of $\{l_1 = 1\}$ becomes 0, because $a^C_{r,1}$ is a center action, so does not affect leaf state prices, and the only L_1-path compliant with $\pi^C(s^\mathcal{F}_{r,1})$ is $\langle\rangle$ with cost 0.

With $\preceq$-pruning against cheaper-visited states, because the path reaching $s^\mathcal{F}_{r,1}$ has higher cost, we can only check if $s^\mathcal{F}_{r,1} \preceq s^\mathcal{F}_{l,1}$, which does not hold. Thus, both states are kept and the search proceeds in the same manner, reaching 2^n decoupled states.

With augmented-cost dominance $\preceq_{\text{aug}}$ against all visited states, however, we have $s^\mathcal{F}_{l,1} \preceq_{\text{aug}} s^\mathcal{F}_{r,1}$, so $s^\mathcal{F}_{l,1}$ can be pruned. We obtain a similar pruning at all later choice points, so only n decoupled states are reached. □

An important question is how to compute the dominance check efficiently, i. e., without explicitly enumerating the costs of all member states. For $\preceq$, this can be done by simply comparing the prices of all reached leaf states directly, so the check is linear in the number of such states. We next show that $\preceq_{\text{aug}}$ can similarly be checked component-wise by only considering the leaf state with the highest price difference per leaf:

Proposition 4 (Augmented-Cost Dominance Check)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. Let* $s^\mathcal{F}$ *and* $t^\mathcal{F}$ *be two decoupled states. Then* $s^\mathcal{F} \preceq_{\text{aug}} t^\mathcal{F}$ *iff* $\mathsf{cost}(\pi^C(s^\mathcal{F})) - \mathsf{cost}(\pi^C(t^\mathcal{F})) \geq \sum_{L \in \mathcal{L}} \max_{s^L \in S^L_R}(\mathsf{prices}(t^\mathcal{F})[s^L] - \mathsf{prices}(s^\mathcal{F})[s^L])$*, where* $S^L_R = \{s^L \in S^L \mid \mathsf{prices}(s^\mathcal{F})[s^L] < \infty\}$*.*

Proof. Let $s \in [s^\mathcal{F}]$ be a member state of $s^\mathcal{F}$ where $\mathsf{prices}(t^\mathcal{F})[s[L]] - \mathsf{prices}(s^\mathcal{F})[s[L]]$ is maximal for all $L \in \mathcal{L}$. Then, if $\mathsf{price}(t^\mathcal{F}, s) - \mathsf{price}(s^\mathcal{F}, s) \leq \mathsf{cost}(\pi^C(s^\mathcal{F})) - \mathsf{cost}(\pi^C(t^\mathcal{F})$, this also holds for all other $s' \in [s^\mathcal{F}]$. Thus, for all $s' \in [s^\mathcal{F}]$ we get that $\mathsf{cost}(\pi^C(t^\mathcal{F})) + \mathsf{price}(t^\mathcal{F}, s') \leq \mathsf{cost}(\pi^C(s^\mathcal{F})) + \mathsf{price}(s^\mathcal{F}, s')$. □

If $t^\mathcal{F}$ has a higher path cost than $s^\mathcal{F}$, but has leaf states with lower prices, then the disadvantage in path cost can be traded against the advantage in leaf state prices. More concretely, it suffices to sum-up only the maximal price-difference of any leaf state over the leaves. Thereby, we essentially compare only the member state $s \in [s^\mathcal{F}]$ for which the price-advantage is maximal. This can be done component-wise, so is efficient to compute.

Augmented-cost dominance also tackles more subtle cases, where prices differ in several leaf factors. We can then *distribute* the difference in path costs *across* the leaf factors, i. e., we cannot use the full difference for each factor. However, we can even trade lower prices in one leaf by higher prices in another, setting these different prices off against the difference in path costs. We can combine the price advantage in one leaf for $s^{\mathcal{F}}$ with its path-cost advantage to make up for a higher price disadvantage in other leaves, where $t^{\mathcal{F}}$ might have lower prices. Assuming that $t^{\mathcal{F}}$ is visited before $s^{\mathcal{F}}$, $s^{\mathcal{F}}$ can be pruned although the prices of its leaf states are neither lower-equal, nor higher-equal than the prices of $t^{\mathcal{F}}$, strictly generalizing dominance via $\preceq$.

3.4.2 Size Blow-Up and Hypercube Pruning

Even in finite cases, the number of non-equivalent decoupled states can blow-up relative to the size of the standard state space. This is because the maintenance of pricing functions can result in a form of “memory of the center path taken”, similar to the example in the proof of Proposition 3, but for reachability, not different price values. Different paths may have provided center preconditions enabling different leaf moves, thus resulting in different sets of reached leaf states. This happens, indeed, even in very simple examples akin to the logistics examples we have been using all along:

Example 5. *Consider the following* BlowUp *example, with one truck T, one package p, and n locations $l_1, \dots, l_n$, where the truck can drive between any pair of locations; both T and p are initially at l_1, the goal is for p to be at l_n. In short, this is like the scaling example except with only a single package, and with locations arranged as a fully-connected road map instead of a line. We consider the factoring with $C = \{T\}$ and $\mathcal{L} = \{\{p\}\}$.*

Obviously, the standard state space in this example is small ($n(n+1)$ reachable states). But the reachable decoupled state space has size exponential in n. Through the pricing functions, the decoupled states “remember” the locations visited by T in the past. For example, the decoupled state reached through $\langle drive(T, l_1, l_2), drive(T, l_2, l_3)\rangle$ has finite prices for $\{p = l_1\}$, $\{p = T\}$, $\{p = l_2\}$, and $\{p = l_3\}$; while the decoupled state reached through $\langle drive(T, l_1, l_4), drive(T, l_4, l_5)\rangle$ has finite prices for $\{p = l_1\}$, $\{p = T\}$, $\{p = l_4\}$, and $\{p = l_5\}$. Hence the decoupled state space enumerates pricing functions corresponding to every subset of visited locations from $\{l_2, \dots, l_n\}$.

Note the simplicity of this example. Blow-ups may occur even when maintaining reachability functions only (as the pricing functions here disagree even on reachability). This also implies that dominance pruning does not prevent the blow-up.

It turns out that blow-ups can be avoided based on the previously introduced hypercube characterization. The core observation in this context is that solvability of a decoupled state is equivalent to solvability of at least one member state:

Lemma 3. *Let Π be a planning task, $\mathcal{F}$ a factoring for Π, and $s^{\mathcal{F}}$ a decoupled state. Then $s^{\mathcal{F}}$ is solvable if and only if at least one $s \in [s^{\mathcal{F}}]$ is solvable.*

This follows from the same correspondence between member state transitions, embedded state transitions, and the decoupled state space, as used to prove correctness in Chapter 3.3. The proof is stated in Appendix A.2.

The idea now is to prune a decoupled state $s^{\mathcal{F}}$ when its hypercube does not contribute any new member states, i. e., if $s^{\mathcal{F}}$ is *covered* by previously visited decoupled states:

Definition 15 (Hypercube Covering)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a factoring for* Π*, and* $s^{\mathcal{F}}, t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ *decoupled states. We say that* $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ cover $s^{\mathcal{F}}$ *if* $\bigcup_{i=1}^{n}[t_i^{\mathcal{F}}] \supseteq [s^{\mathcal{F}}]$.

From Lemma 3, we immediately get:

Theorem 4 (Correctness of Hypercube Pruning)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. Let* $s^{\mathcal{F}}, t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ *be decoupled states, where* $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ *cover* $s^{\mathcal{F}}$*. If* $s^{\mathcal{F}}$ *is solvable, then so is at least one* $t_i^{\mathcal{F}}$*.*

Proof. Say that $s^{\mathcal{F}}$ is solvable. By Lemma 3 applied to $s^{\mathcal{F}}$, some $s \in [s^{\mathcal{F}}]$ is solvable. By prerequisite, $s \in \bigcup_{i=1}^{n}[t_i^{\mathcal{F}}]$. So there is at least one $t_i^{\mathcal{F}}$ such that $s \in [t_i^{\mathcal{F}}]$. By Lemma 3 applied to $t_i^{\mathcal{F}}$, $t_i^{\mathcal{F}}$ is solvable as desired. □

Therefore, hypercube pruning is correct: if $s^{\mathcal{F}}$ is covered by previously visited decoupled states $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$, then pruning $s^{\mathcal{F}}$ does not forego completeness. This applies to the same three forms of pruning as above, i. e., ancestor pruning, cheaper-visited pruning, and all-visited pruning. In each case, the respective collection of $t^{\mathcal{F}}$ becomes the set $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ in the hypercube-covering check.

On the other hand, hypercube pruning does, as stated, not preserve optimality. The hypercube disregards leaf state *prices*, and so does the notion of hypercube covering.

Given this, cheaper-visited pruning does not make much sense. Ancestor pruning is not enough to prevent exponential blow-ups. However, all-visited pruning provides the desired guarantee:

Theorem 5 (Size Guarantee under Hypercube Pruning)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. Under all-visited hypercube pruning, the number of decoupled states in* $\Theta_{\Pi}^{\mathcal{RF}}$ *is bounded by the number of reachable states in* Θ_{Π}*.*

Proof. Consider the non-pruned visited decoupled states $s^{\mathcal{F}}$ under all-visited hypercube pruning. With $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ being the decoupled states visited prior to $s^{\mathcal{F}}$ sharing the same center state, denote by $\overline{[s^{\mathcal{F}}]} := [s^{\mathcal{F}}] \setminus \bigcup_{i=1}^{n}[t_i^{\mathcal{F}}]$ the hypercube remaining of $s^{\mathcal{F}}$ when eliminating $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$. Clearly, $\overline{[s^{\mathcal{F}}]} \neq \emptyset$ as otherwise $s^{\mathcal{F}}$ would be pruned. Furthermore, for any other non-pruned visited decoupled state $r^{\mathcal{F}}$, $\overline{[s^{\mathcal{F}}]} \cap \overline{[r^{\mathcal{F}}]} = \emptyset$: if $r^{\mathcal{F}}$ was visited before $s^{\mathcal{F}}$, this is because $[r^{\mathcal{F}}]$ was eliminated from $\overline{[s^{\mathcal{F}}]}$; if $r^{\mathcal{F}}$ was visited after $s^{\mathcal{F}}$, this is because $[s^{\mathcal{F}}]$ was eliminated from $\overline{[r^{\mathcal{F}}]}$. So the non-pruned decoupled states are associated with non-empty and disjoint sets $\overline{[s^{\mathcal{F}}]}$ of states in Θ_{Π}. The claim now follows together with Lemma 1, which implies that, if $s^{\mathcal{F}}$ is reachable in $\Theta_{\Pi}^{\mathcal{F}}$, then every $s \in [s^{\mathcal{F}}]$ is reachable in Θ_{Π}. □

The bad news is that testing Definition 15 is, in general, hard, even for the hypercubes that may actually occur during decoupled search:

Proposition 5. *Given a planning task* Π *and a factoring* $\mathcal{F}$ *for* Π*, it is co-***NP***-complete to decide whether reachable decoupled states* $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ *cover a reachable decoupled state* $s^{\mathcal{F}}$*.*

Proof (sketch). Membership follows directly from the results by Hoffmann and Kupferschmid [2005] for general hypercube covering problems. Hardness follows by reduction from the complement of SAT, extending Hoffmann and Kupferschmid's argument by a simple construction of Π and $\mathcal{F}$.

Given any CNF formula ϕ with clauses $C_1, \dots, C_m$, the construction includes, for each clause C_j, a center action a_j^C which is applicable to the initial state, and which allows to generate a hypercube $t_j^{\mathcal{F}}$ corresponding to the truth-value assignments disallowed by C_j. Another center action a_0^C allows to generate the hypercube $s^{\mathcal{F}}$ corresponding to the space of all truth-value assignments. Consider the time point in search where search has explored each of the alternatives $a_1^C, \dots, a_m^C$ (applied each of these actions to the initial state separately), and now explores the alternative a_0^C. Then all-visited hypercube pruning checks whether $t_1^{\mathcal{F}}, \dots, t_m^{\mathcal{F}}$ cover $s^{\mathcal{F}}$. The latter is the case iff ϕ is unsatisfiable. □

We remark that in $\mathcal{F}$ in the proof construction the dependencies between the factors are of the simple form $C \rightarrow L$, for $L \in \mathcal{L}$, so co-**NP**-completeness holds for this restricted case already. The full proof is stated in Appendix A.2.

Exploring optimality-preserving forms of hypercube pruning remains a topic for future work. One simple option to adapt Definition 15 would be to restrict the "cube edge" in each dimension L to only those leaf states whose price is at most the price in $s^{\mathcal{F}}$. That is, for the purpose of the hypercube cover test we could set $[t_i^{\mathcal{F}}] := \{s \mid s[C] = \mathsf{center}(s_i^{\mathcal{F}}), \forall L \in \mathcal{L} : \mathsf{prices}(t_i^{\mathcal{F}})[s[L]] \leq \mathsf{prices}(s^{\mathcal{F}})[s[L]] < \infty\}$. Then the states covered by $[t_i^{\mathcal{F}}]$ are reached at equal or cheaper cost than in $s^{\mathcal{F}}$. Cheaper-visited pruning with optimal search algorithms like A* should then preserve optimality.

Chapter 4

Heuristic Search

Heuristic search, where the search is guided by a heuristic, a function that maps states to estimates of *remaining cost* to reach a goal state, has been highly successful in AI planning for the last two decades (e. g. McDermott, 1999; Bonet and Geffner, 2001; Hoffmann and Nebel, 2001; Gerevini et al., 2003; Helmert, 2006b; Helmert and Domshlak, 2009; Richter and Westphal, 2010; Domshlak et al., 2015a).

In this chapter, we show that decoupled search combines gracefully with standard heuristic search methods. Chapter 4.1 discusses heuristic functions, in Chapter 4.2 we look into search algorithms, Chapter 4.3 introduces a cost transformation, which can significantly improve the tie-breaking behaviour of an A^* search.

This chapter is mostly based on Gnad and Hoffmann [2018]. Center heuristics and anytime decoupled search have been introduced in Gnad and Hoffmann [2015a]. The g-value adaptation has first been described in Gnad [2021b].

4.1 Heuristic Functions

Our definition of heuristic functions for decoupled search follows the standard concepts. We distinguish between two types of heuristics, *star heuristics*, which are based on the notion of augmented optimality, taking all costs into account, and *center heuristics* that consider only the remaining cost of the center actions:

Definition 16 (Decoupled Heuristic Function)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. A* decoupled heuristic function*,* heuristic *for short, is a function* $h : \mathcal{S}^{\mathcal{F}} \mapsto \mathbb{R}^{0+} \cup \{\infty\}$.

The augmented-perfect *heuristic* $h^*_{\mathcal{F}}$ *assigns each* $s^{\mathcal{F}} \in \mathcal{S}^{\mathcal{F}}$ *the cost of an augmented-optimal solution for* $s^{\mathcal{F}}$*. We say that a heuristic* $h_{\mathcal{F}}$ *is* augmented-admissible *if* $h_{\mathcal{F}} \leq h^*_{\mathcal{F}}$.

The center-perfect *heuristic* h^*_C *assigns each* $s^{\mathcal{F}} \in \mathcal{S}^{\mathcal{F}}$ *the cost of an optimal solution for* $s^{\mathcal{F}}$*. We say that a heuristic* h_C *is* center-admissible *if* $h_C \leq h^*_C$.

We call heuristics that approximate $h^*_{\mathcal{F}}$ *star heuristics*, notation convention $h_{\mathcal{F}}$, and heuristics that approximate h^*_C *center heuristics*, notation convention h_C.

The possibility to return ∞, not a numeric estimate, is intended to allow the heuristic to identify unsolvable states, a capability many classical-planning heuristic functions have (e. g. Haslum and Geffner, 2000; Helmert et al., 2014; Hoffmann et al., 2014).

Per the definition of augmented optimality (Definition 10), the augmented-perfect heuristic $h^*_{\mathcal{F}}$ accounts not only for standard path cost from a decoupled state $s^{\mathcal{F}}$, but also for the cost of compliant leaf goal paths that a solution for $s^{\mathcal{F}}$ needs to be augmented with. A subtlety here is that, given the least-commitment strategy for leaf factors, selecting the *entire* leaf path only at the end, part of the associated price lies "in the past", before $s^{\mathcal{F}}$. Specifically, let $\pi^{\mathcal{F}}$ be a solution for $s^{\mathcal{F}}$, ending in $s^{\mathcal{F}}_G$. $h^*_{\mathcal{F}}$ accounts for (1) the cost of $\pi^{\mathcal{F}}$, i. e., of the center action sequence π^C underlying $\pi^{\mathcal{F}}$; plus (2) the cost of $\pi^C(s^{\mathcal{F}}_G)$-compliant leaf goal paths π^L. In (2), $\pi^C(s^{\mathcal{F}}_G) = \pi^C(s^{\mathcal{F}}) \circ \pi^C$, so part of the paths π^L will be scheduled alongside the path $\pi^C(s^{\mathcal{F}})$ leading to $s^{\mathcal{F}}$. In particular, while heuristic functions usually return 0 on goal states, that is not so for $h^*_{\mathcal{F}}$: we still have to pay the leaf-goal prices, moving the leaves into place.

In contrast, the center-perfect heuristic h^*_C accounts only for the costs of the center actions underlying a solution $\pi^{\mathcal{F}}$ for a decoupled state $s^{\mathcal{F}}$, ignoring its leaf-state prices. Thereby, center and star heuristics may *disagree*. Using our example, say that there are two alternative kinds of plans, (a) ones that pass the packages through several trucks, loading/unloading every time, vs. (b) ones that make more truck moves but have to load/unload each package only once and thus are better globally. Then h^*_C will draw search towards plans (a), whereas $h^*_{\mathcal{F}}$ will draw search towards plans (b).

Pricing functions capture everything relevant about the past, so that it suffices to consider the possible futures *starting from the current pricing function*. We formulate this in terms of a compilation into classical planning, allowing to estimate $h^*_{\mathcal{F}}$ and h^*_C through standard classical-planning heuristics. The idea is to force the plan to "buy" exactly one leaf state from each leaf factor:

Definition 17 ($h^*_{\mathcal{F}}$ and h^*_C as Classical Planning)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task,* $\mathcal{F}$ *a factoring for* Π *with center* C *and leaves* $\mathcal{L}$*, and* $s^{\mathcal{F}}$ *a decoupled state. The* buy-leaves compilation *is the planning task* $\Pi_{L\$} = \langle \mathcal{V}_{L\$}, \mathcal{A}_{L\$}, \mathsf{cost}_{L\$}, s^{\mathcal{F}}_{L\$}, \mathcal{G}_{L\$} \rangle$*, obtained from* Π *and* $s^{\mathcal{F}}$ *as follows:*

1. *The variables* $\mathcal{V}_{L\$}$ *add a new Boolean variable* $\mathsf{bought}[L]$ *for every leaf* L*,* $\mathcal{V}_{L\$} := \mathcal{V} \cup \{\mathsf{bought}[L] | L \in \mathcal{L}\}$*. For all variables* $v \notin C$*, we add the new value* none *into* $\mathcal{D}(v)$*.*
2. *The initial state is* $s^{\mathcal{F}}_{L\$} := \mathsf{center}(s^{\mathcal{F}}) \cup \{v{=}\mathsf{none} \mid v \notin C\} \cup \{\mathsf{bought}[L]{=}\bot \mid L \in \mathcal{L}\}$*.*
3. *The goal is* $\mathcal{G}_{L\$} := \mathcal{G} \cup \{\mathsf{bought}[L] = \top \mid L \in \mathcal{L}\}$*.*
4. *The actions* $\mathcal{A}_{L\$}$ *are the previous ones* $\mathcal{A}$*, adding precondition* $\mathsf{bought}[L] = \top$ *to* a *whenever* $(\mathsf{vars}(\mathsf{pre}(a)) \cup \mathsf{vars}(\mathsf{eff}(a))) \cap L \neq \emptyset$*. We furthermore add, for every*

leaf L, and for every leaf state $s^L \in S^L$ where $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$*, a new action* $a[s^L]$ *with precondition* $\mathsf{pre}(a[s^L]) := \{\mathsf{bought}[L] = \bot\}$ *and effect* $\mathsf{eff}(a[s^L]) := s^L \cup \{\mathsf{bought}[L] = \top\}$.

5. (a) *For the augmented-perfect heuristic $h^*_{\mathcal{F}}$, the cost function* $\mathsf{cost}_{L\$}$ *extends the previous one* cost *by setting* $\mathsf{cost}_{L\$}(a[s^L]) := \mathsf{prices}(s^{\mathcal{F}})[s^L]$ *for each new action* $a[s^L]$.

   (b) *For the center-perfect heuristic h^*_C, the cost function* $\mathsf{cost}_{L\$}$ *extends the previous one* cost *by setting* $\mathsf{cost}_{L\$}(a[s^L]) := 0$ *for each new action* $a[s^L]$*. Furthermore, we adapt* cost *by setting* $\mathsf{cost}_{L\$}(a) := 0$ *for all leaf-only actions* $a \in \mathcal{A}^{\mathcal{L}}_{\not{C}}$.

Center heuristics will be employed mostly for non-optimal planning where only reachability is maintained for the leaf states. We will, however, also introduce an optimal planning algorithm that is based on center heuristics, see Chapter 4.2.1.

We will use the buy-leaves compilation to design heuristic functions for decoupled search. We explain this below; let us first explain the compilation. The leaf factors are assigned the value none initially to indicate that they "do not have a state yet". Before we can do anything relying on a leaf factor L, we have to buy (exactly) one of its states, at the price specified in the decoupled state $s^{\mathcal{F}}$ at hand (or 0 if we want a center heuristic). For star heuristics, the price we pay in doing so accounts for L's compliant path before $s^{\mathcal{F}}$; the classical plan obtained afterwards accounts for L's compliant path behind $s^{\mathcal{F}}$.

Note that the goal in $\Pi_{L\$}$ forces the plan to buy a leaf state from *every* L, even if L has no goal and would otherwise not be touched by any actions in the plan for $\Pi_{L\$}$. This is necessary because L may have had to move *before* $s^{\mathcal{F}}$: we need to account for any costs incurred in L in order to enable (to comply with) the center path $\pi^C(s^{\mathcal{F}})$ leading to $s^{\mathcal{F}}$ in the first place. For a center heuristic, we still need to "buy" the leaf states (select one per leaf for cost 0), because the center might need to provide different preconditions for required leaf actions depending on which leaf states are reached.

Example 6. *Consider our example with one package p and two trucks T_1, T_2 moving along the line l_1, l_2, l_3, l_4. Initially, $T_1 = l_1$, $T_2 = l_4$, and $p = l_1$. The goal is $p = l_4$. Consider the factoring with $C = \{T_1, T_2\}$ and $\mathcal{L} = \{\{p\}\}$, and consider the decoupled state $s^{\mathcal{F}}$ reached by applying drive(T_1, l_1, l_2). We have $h^*_{\mathcal{F}}(s^{\mathcal{F}}) = 4$ due to the optimal solution induced by the center path $\langle$drive(T_1, l_2, l_3), drive$(T_1, l_3, l_4)\rangle$, augmented with the leaf goal path $\langle$load(T_1, p, l_1), unload$(T_1, p, l_4)\rangle$. Observe that load(T_1, p, l_1) is scheduled before $s^{\mathcal{F}}$, while unload(T_1, p, l_4) is scheduled behind $s^{\mathcal{F}}$. The center-perfect heuristic for $s^{\mathcal{F}}$ is $h^*_C(s^{\mathcal{F}}) = 2$, accounting only for the two center actions.*

The finite prices in $s^{\mathcal{F}}$ are 0 for $\{p = l_1\}$, 1 for $\{p = T_1\}$, and 2 for $\{p = l_2\}$. In the buy-leaves compilation $\Pi_{L\$}$, we can pay one of these prices to obtain a leaf state other than none*. The cheapest plan for $\Pi_{L\$}$ results from buying $\{p = T_1\}$, yielding the plan $\langle a[\{p = T_1\}]$, drive(T_1, l_2, l_3), drive(T_1, l_3, l_4), unload$(T_1, p, l_4)\rangle$ which corresponds to* *the part behind $s^{\mathcal{F}}$ in the optimal solution above. (If we buy $p = l_1$ instead then additional truck moves are needed; if we buy $p = l_2$ instead then an additional load action is needed.)*

Consider now a variant, which is the same except that truck moves require p to be in the truck, i. e., $\mathsf{pre}(\mathit{drive}(T_i, l_j, l_k)) = \{T_i = l_j, p = T_i\}$. *Say, however, that the goal is $T_1 = l_4$. Consider again the decoupled state $s^{\mathcal{F}}$ reached by applying $\mathit{drive}(T_1, l_1, l_2)$. The finite prices in $s^{\mathcal{F}}$ now are 1 for $\{p = T_1\}$ and 2 for $\{p = l_2\}$; $\{p = l_1\}$ is no longer reachable as the truck drive committed p to $\{p = T_1\}$. Observe that $h^*_{\mathcal{F}}(s^{\mathcal{F}}) = 3$ because the optimal solution is $\langle \mathit{drive}(T_1, l_2, l_3), \mathit{drive}(T_1, l_3, l_4)\rangle$ augmented with the compliant leaf goal path $\langle \mathit{load}(T_1, p, l_1)\rangle$. Now, if $\Pi_{L\$}$ did* not *have the goal* $\mathsf{bought}[\{p\}] = \top$, *then $\langle \mathit{drive}(T_1, l_2, l_3), \mathit{drive}(T_1, l_3, l_4)\rangle$ would be a plan for $\Pi_{L\$}$, of cost $2 < h^*_{\mathcal{F}}(s^{\mathcal{F}})$. The goal* $\mathsf{bought}[\{p\}] = \top$ *forces the plan to pay for any services p may have needed to provide before $s^{\mathcal{F}}$. The center-perfect heuristic for $s^{\mathcal{F}}$ is still $h^*_C(s^{\mathcal{F}}) = 2$.*

Given a classical-planning heuristic function h, we obtain the *buy-leaves star heuristic function* $h^{\mathcal{F}}_{L\$}$ for a decoupled state $s^{\mathcal{F}}$ by setting the value of $h^{\mathcal{F}}_{L\$}$ on $s^{\mathcal{F}}$ to the value of h on $s^{\mathcal{F}}_{L\$}$ in $\Pi_{L\$}$ when using the cost function from Definition 17 part 5(a). We obtain the *buy-leaves center heuristic function* $h^{C}_{L\$}$ for a decoupled state $s^{\mathcal{F}}$ by setting the value of $h^{C}_{L\$}$ on $s^{\mathcal{F}}$ to the value of h on $s^{\mathcal{F}}_{L\$}$ in $\Pi_{L\$}$ when using the cost function from Definition 17 part 5(b). In other words, both heuristics result from h's estimate of initial-state remaining cost in the buy-leaves compilation for $s^{\mathcal{F}}$. This construction guarantees two very desirable properties: *if h is admissible, then $h^{\mathcal{F}}_{L\$}$ ($h^{C}_{L\$}$) is augmented-admissible (center-admissible)*; and *if h is perfect, then $h^{\mathcal{F}}_{L\$}$ ($h^{C}_{L\$}$) is augmented-perfect (center-perfect)*. The former is of course crucial for optimal planning. The latter curbs information loss: if the heuristic information given by h is perfect, then no loss is incurred. Both properties follow directly from the fact that $\Pi_{L\$}$ indeed captures $h^*_{\mathcal{F}}$ and h^*_C:

Lemma 4. *Let Π be a planning task, $\mathcal{F}$ a factoring for Π, and $s^{\mathcal{F}}$ a decoupled state. Let $\Pi_{L\$} = (\mathcal{V}_{L\$}, \mathcal{A}_{L\$}, \mathsf{cost}_{L\$}, s^{\mathcal{F}}_{L\$}, \mathcal{G}_{L\$})$ be the buy-leaves compilation. Then $h^*(s^{\mathcal{F}}_{L\$}) = h^*_{\mathcal{F}}(s^{\mathcal{F}})$ for the cost function in Definition 17 part 5(a) and $h^*(s^{\mathcal{F}}_{L\$}) = h^*_C(s^{\mathcal{F}})$ for the cost function in Definition 17 part 5(b).*

Proof. "$\leq$": Say $\pi^{\mathcal{F}}$ is any solution for $s^{\mathcal{F}}$, ending in decoupled state $s^{\mathcal{F}}_G$. Let π^C be the sequence of center actions underlying $\pi^{\mathcal{F}}$. We can construct a plan π for $\Pi_{L\$}$ by augmenting π^C with a cheapest π^C-compliant sequence of leaf actions for each leaf factor L, starting from a finite-price leaf state $s^L \in S^L$ in $s^{\mathcal{F}}$; and buying that leaf state via the action $a[s^L]$. By construction, the cost of π in $\Pi_{L\$}$ is $\mathsf{cost}(\pi^C) + \mathsf{gprice}(s^{\mathcal{F}}_G) = \mathsf{augCost}(\pi^{\mathcal{F}})$. Hence $h^*(s^{\mathcal{F}}_{L\$}) \leq h^*_{\mathcal{F}}(s^{\mathcal{F}})$. For h^*_C, we can construct a plan π in the same way, where, because all leaf actions have cost 0, the cost of π is $\mathsf{cost}(\pi^C)$, and thus $h^*(s^{\mathcal{F}}_{L\$}) \leq h^*_C(s^{\mathcal{F}})$.

"$\geq$": Say π is any plan for $\Pi_{L\$}$. Let π^C be the subsequence of center actions. As the construction of $\Pi_{L\$}$ forces the plan to buy, for each leaf factor, exactly one leaf state $s^L \in S^L$, π^C must be augmentable with π^C-compliant leaf paths achieving the leaf goals starting from these s^L. So π^C induces a solution for $s^{\mathcal{F}}$, ending in some $s^{\mathcal{F}}_G$. If the leaf paths used by π are the cheapest ones, then $\mathsf{cost}(\pi) = \mathsf{cost}(\pi^C) + \mathsf{gprice}(s^{\mathcal{F}}_G) = \mathsf{augCost}(\pi^{\mathcal{F}})$, respectively $\mathsf{cost}(\pi) = \mathsf{cost}(\pi^C)$ if leaf-action costs are 0. Thus $h^*(s^{\mathcal{F}}_{L\$}) \geq h^*_C(s^{\mathcal{F}})$ for the cost function in Definition 17 part 5(b), as desired. For arbitrary leaf paths, we have that $\mathsf{cost}(\pi) \geq \mathsf{augCost}(\pi^{\mathcal{F}})$, and hence that $h^*(s^{\mathcal{F}}_{L\$}) \geq h^*_{\mathcal{F}}(s^{\mathcal{F}})$ for the cost function in Definition 17 part 5(a), again as desired. □

Theorem 6. *Let Π a planning task, and $\mathcal{F}$ a factoring for Π. Let h be a heuristic function. If h is admissible, then $h^{\mathcal{F}}_{L\$}$ ($h^C_{L\$}$) is augmented-admissible (center-admissible). If $h = h^*$, then $h^{\mathcal{F}}_{L\$} = h^*_{\mathcal{F}}$ and $h^C_{L\$} = h^*_C$.*

Proof. Direct from Lemma 4. □

It should be noted that, while our construction can in principle be used with any heuristic function h, doing so efficiently may be challenging. In particular, the subset of artificial actions $a[s^L]$ present, as well as their cost, depend on the decoupled state $s^{\mathcal{F}}$. This is problematic for heuristic functions relying crucially on precomputations prior to search, like abstraction heuristics. We have so far realized the buy-leaves compilation for a number of canonical heuristic functions not relying on precomputation (namely for $h^{\max}$, $h^{\text{LM-cut}}$, and h^{FF}; see more details in Chapter 7).

4.2 Heuristic Search Algorithms

Disregarding solution quality, one can run any search algorithm on the decoupled state space $\Theta^{\mathcal{F}}_{\Pi}$, treating it like an arbitrary transition system. When taking solution quality into account, in particular for optimality, matters are a little more subtle as we need to tackle augmented-optimality in $\Theta^{\mathcal{F}}_{\Pi}$, in difference to the standard additive-cost notion for which heuristic search algorithms are defined. In particular, as we illustrated in Example 1, a solution for $\Theta^{\mathcal{F}}_{\Pi}$ may have a worse solution as a prefix. The issue is easy to resolve though, by making the leaf-goal prices explicit, encoding them as additional transitions to a new goal state:

Definition 18 (Explicit Leaf-Goal Price)**.** *Let Π be a planning task, $\mathcal{F}$ a factoring for Π, and $\Theta^{\mathcal{F}}_{\Pi} = \langle \mathcal{S}^{\mathcal{F}}, \mathcal{A}^C, \mathsf{cost}|_{\mathcal{A}^C}, \mathcal{T}^{\mathcal{F}}, \mathcal{I}^{\mathcal{F}}, \mathcal{S}^{\mathcal{F}}_G \rangle$ the decoupled state space. The* explicit leaf-goal price state space $\Theta^{LG\mathcal{F}}_{\Pi}$ *is like $\Theta^{\mathcal{F}}_{\Pi}$, but with a new state $s^{\mathcal{F}}_*$ set to be the only goal state; and adding, for every $s^{\mathcal{F}}_G \in \mathcal{S}^{\mathcal{F}}_G$, a new transition $s^{\mathcal{F}}_G \to s^{\mathcal{F}}_*$ with a new label whose cost is* $\mathsf{gprice}(s^{\mathcal{F}}_G)$.

In other words, $\Theta_{\Pi}^{LG\mathcal{F}}$ requires to pay the leaf-goal prices at the end of any solution, in the form of a transition having that cost. Obviously, the solutions for $\Theta_{\Pi}^{LG\mathcal{F}}$ are in one-to-one correspondence with those for $\Theta_{\Pi}^{\mathcal{F}}$, where additive cost in the former corresponds to augmented cost in the latter.

Consider now any optimal search algorithm X for additive cost in labeled transition systems, and consider a star heuristic function h defined on $\mathcal{S}^{\mathcal{F}}$. Define a heuristic function $h^{\mathsf{LG}\mathcal{F}}$ for $\Theta_{\Pi}^{LG\mathcal{F}}$ simply by extending h with $h^{\mathsf{LG}\mathcal{F}}(s_{*}^{\mathcal{F}}) := 0$. Define the algorithm *Decoupled-X (D-X)* as running X with $h^{\mathsf{LG}\mathcal{F}}$ on $\Theta_{\Pi}^{LG\mathcal{F}}$, returning $\pi^{\mathcal{F}}$ when X returns $\pi^{\mathcal{F}} \circ \langle s_{G}^{\mathcal{F}} \rightarrow s_{*}^{\mathcal{F}} \rangle$. With the above, we have:

Proposition 6. *If X is complete, then D-X is complete. If X is optimal for admissible heuristic functions, then D-X is augmented-optimal for augmented-admissible heuristic functions.*

With Theorem 6 and Proposition 6 together, we can (in principle) take any complete/optimal heuristic search algorithm X, and any classical-planning heuristic function h, and turn them into a complete/augmented-optimal search algorithm for $\Theta_{\Pi}^{\mathcal{F}}$, searching with $h_{L\$}^{\mathsf{LG}\mathcal{F}}$ on $\Theta_{\Pi}^{LG\mathcal{F}}$. By Theorems 1 and 2, this yields a complete/optimal planning algorithm.

We remark that center-admissible heuristics with an optimal search algorithm do not result in an augmented-optimal search algorithm for $\Theta_{\Pi}^{\mathcal{F}}$. This is because of the aforementioned "disagreement" of center and star heuristics. The center heuristic might draw the search to path with low center costs, which need to be augmented with high leaf costs. We will introduce a specialized version of the A^{*} algorithm in the next section that uses center heuristics for optimal planning.

4.2.1 Anytime Decoupled A^{*}

The above construction is simple and canonical and is hence applicable to any search algorithm and heuristic. In this section, we introduce a heuristic search algorithm specialized to decoupled search which uses a center heuristic for guidance, enabling the algorithm to explore decoupled states by estimated remaining center cost. As mentioned before, this does not guarantee augmented optimality, and hence global optimality with respect to the given planning task. Our algorithm is based on the A^{*} algorithm [Hart et al., 1968], which is widely employed in combination with an admissible heuristic as a basis for optimal planning approaches.

The modifications needed when adapting A^{*} are detailed in Figure 4.1. We baptize our new algorithm *Anytime D-*A^{*}. The search nodes are denoted by $N[s^{\mathcal{F}}]$, where $s^{\mathcal{F}}$ is the corresponding state and N is the node itself. By $g(N)$ we denote A^{*}'s *g-value*, i. e., the cost $\mathsf{cost}(\pi^{\mathcal{F}})$ of the decoupled path $\pi^{\mathcal{F}}$ reaching N. To guarantee optimality, (1. and 4.) we simply do not stop the search once a goal state is selected for expansion,

```
Anytime D-A$^*$($\Pi$, $\mathcal{F}$, $h_C$, $h_\mathcal{F}$):
$U \leftarrow \infty$ /* best known upper bound */
$s^\mathcal{F}_U \leftarrow \mathcal{I}^\mathcal{F}$ /* corresponding state */
Run A$^*$ with center heuristic $h_C$ on $\Theta^\mathcal{F}_\Pi$, with these modifications:
    1. Continue search until the open list is empty;
    2. Whenever a goal vertex node $N[s^\mathcal{F}_G]$ is expanded:
        if $\mathsf{cost}(\pi^C(s^\mathcal{F}_G)) + \mathsf{gprice}(s^\mathcal{F}_G) < U$ then
            $U \leftarrow \mathsf{cost}(\pi^C(s^\mathcal{F}_G)) + \mathsf{gprice}(s^\mathcal{F}_G)$
            $s^\mathcal{F}_U \leftarrow s^\mathcal{F}_G$
        if $\mathsf{gprice}(s^\mathcal{F}_G) = \mathsf{min\text{-}gprice}$ then
            return $\pi^G(s^\mathcal{F}_U)$ /* early termination */
    3. Whenever a node $N[s^\mathcal{F}]$ is generated, and $U \neq \infty$:
        if $\mathsf{cost}(\pi^C(s^\mathcal{F})) + h_\mathcal{F}(s^\mathcal{F}) \geq U$ then
            discard $N$ /* upper-bound pruning with star heuristic */
    4. When open list is empty:
        if $s^\mathcal{F}_U \neq \mathcal{I}^\mathcal{F}$ then return $\pi^G(s^\mathcal{F}_U)$
        else return "unsolvable"
```

Figure 4.1: The Anytime D-A*algorithm. Search nodes are notated $N[s^\mathcal{F}]$ where $s^\mathcal{F}$ is the state and N the node itself. min-gprice is the sum, over the leaf factors $L \in \mathcal{L}$, of optimal plan cost for the *projection* of Π onto L.

but, when the open list is exhausted, return the best solution found. Additionally, (2.) the search employs an *early termination* mechanism that triggers if the found solution is guaranteed to be optimal. Early termination is based on a global lower bound on the leaf goal price for all leaves, computed by optimally solving, for every leaf L, the projection of Π onto L. The sum of these estimates, the *minimum leaf-goal price*, is denoted min-gprice. Without early termination, the new algorithm would be dominated by D-A* because Anytime D-A* would then have to expand at least all nodes $N[s^\mathcal{F}]$ where $g(N) + h_\mathcal{F}(s^\mathcal{F})$ is less than augmented-optimal solution cost. *With* early termination, that is not so because in the best case we have to exhaust only those $N[s^\mathcal{F}]$ where $g(N) + h_\mathcal{F}(s^\mathcal{F})$ is less than optimal *center* solution cost. Lastly, (3.) the search prunes states against the best known global solution so far, using a star heuristic. We employ a star heuristic here to get an admissible estimate of the total (augmented) cost that still needs to be spent.

We next prove that Anytime D-A* is indeed complete and optimal:

Theorem 7 (Optimality of Anytime D-A*)**.** *Anytime D-*A* *is complete, and is augmented-optimal for center-admissible* h_C *and augmented-admissible* $h_\mathcal{F}$*.*

Proof. Completeness of Anytime D-A* is obvious with Theorem 2. Towards proving optimality, observe: (1) Without early termination and upper-bound pruning, Anytime D-A* generates (in particular) every augmented-optimal decoupled plan $\pi^{\mathcal{F}}$. (2) With center-admissible h_C, if Anytime D-A* generates $\pi_1^{\mathcal{F}}$ before it generates $\pi_2^{\mathcal{F}}$, then $\mathsf{cost}(\pi_1^{\mathcal{F}}) \leq \mathsf{cost}(\pi_2^{\mathcal{F}})$. (3) Upper-bound pruning trivially preserves (2), and it preserves (1) with augmented-admissible $h_{\mathcal{F}}$ because the pruned nodes N cannot lead to augmented-optimal solutions. (4) If early termination fires for $N[s_G^{\mathcal{F}}]$, then $\mathsf{gprice}(s_G^{\mathcal{F}})$ is minimal in $\mathcal{S}_{\mathcal{G}}^{\mathcal{F}}$.

Optimality of Anytime D-A* without early termination holds by (1) and (3) with Theorem 2. Say that early termination fires for $N[s_G^{\mathcal{F}}]$, let $\pi_1^{\mathcal{F}}$ be the decoupled plan leading to N, and let $\pi_2^{\mathcal{F}}$ be any decoupled plan that would be generated later if we continued the search. By (4) $\pi_2^{\mathcal{F}}$ pays at least as high a goal price as $\pi_1^{\mathcal{F}}$, and by (2) $\mathsf{cost}(\pi_1^{\mathcal{F}}) \leq cost(\pi_2^{\mathcal{F}})$, so $\pi_2^{\mathcal{F}}$ cannot have better augmented cost than $\pi_1^{\mathcal{F}}$. With (1), Anytime D-A* already found a augmented-optimal plan to a goal state, which must be stored in $s_U^{\mathcal{F}}$. □

As we will see in Chapter 7, there are cases where Anytime D-A* indeed empirically outperforms D-A*. This occurs, however, only in certain restricted topologies where only the leaves depend on the center, but not vice versa. This is because ignoring the leaf costs can tremendously underestimate the total costs in other topologies, and thereby mislead the search.

4.3 g-Value Adaptation

Closing this chapter, we introduce a cost transformation that is designed to improve the tie-breaking behaviour of A* for optimal planning, and the search guidance in the presence of a weak heuristic.

In a typical A* implementation, where search nodes are expanded in increasing order of their $f = g + h$ value, the algorithm breaks ties, i. e., selects among all nodes with the same f-value, the ones with minimal h. These nodes are expected to be closer to the goal than states with higher h, but lower g. An issue with this regarding D-A* is that due to the pricing function, the split between cost already spent and estimated cost to the goal is not so clear, any more. This is because the prices capture the cost spent to the current state, but this cost is not part of the state's g-value, but is taken into account by the heuristic. In combination with a weak heuristic, i. e., a heuristic that heavily underestimates h^*, A* is not aware of a certain part of the cost, which can cause poor search guidance.

To tackle this issue, we next introduce a cost transformation which moves as much of the leaf-state prices into the g-value of a decoupled state as possible. Assume that in a decoupled state $s^{\mathcal{F}}$ there exists a leaf L such that all leaf states s^L have a minimum non-zero price p_{min}^L, so $\forall s^L \in S^L : \mathsf{prices}(s^{\mathcal{F}})[s^L] \geq p_{min}^L$. Then we can reduce the

$$\boxed{\begin{array}{l} s^{\mathcal{F}} : g(s^{\mathcal{F}}) = 5 \\ s_1^L \rightarrow 3 \ s_1^{L'} \rightarrow 2 \\ s_2^L \rightarrow 1 \ s_2^{L'} \rightarrow 3 \end{array}} \rightarrow \boxed{\begin{array}{l} s_{gA}^{\mathcal{F}} : g(s_{gA}^{\mathcal{F}}) = 8 \\ s_1^L \rightarrow 2 \ s_1^{L'} \rightarrow 0 \\ s_2^L \rightarrow 0 \ s_2^{L'} \rightarrow 1 \end{array}}$$

Figure 4.2: Illustrating example showing the decoupled state $s^{\mathcal{F}}$ and its g-adapted representative $s_{gA}^{\mathcal{F}}$.

prices of all these leaf states by p_{min}^L and increase $g(s^{\mathcal{F}})$ by p_{min}^L without affecting the total cost $\mathsf{cost}(\pi^C(s^{\mathcal{F}})) + \mathsf{price}(s^{\mathcal{F}}, s)$ of the member states s of $s^{\mathcal{F}}$. Intuitively, the transformation moves the price that has to be spent to reach the cheapest member state of $s^{\mathcal{F}}$ into its g-value, reducing the price of all leaf states accordingly, so that in every leaf L there exists at least one leaf state with price 0. See Figure 4.2 for an example of a decoupled state $s^{\mathcal{F}}$ and its cost-transformed representative $s_{gA}^{\mathcal{F}}$.

Formally, we capture the g-value adaption as a transformation of the transition costs of the decoupled state space and the pricing functions of its states:

Definition 19 (g-value Adaptation)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a factoring for* Π*, and* $\Theta_\Pi^{\mathcal{RF}} = \langle \mathcal{S}^{\mathcal{F}}, \mathcal{A}^C, \mathsf{cost}|_{\mathcal{A}^C}, \mathcal{T}^{\mathcal{F}}, \mathcal{I}^{\mathcal{F}}, \mathcal{S}_{\mathcal{G}}^{\mathcal{F}} \rangle$ *the reachable decoupled state space. The* g-value adapted decoupled state space $\Theta_\Pi^{\mathcal{RF}gA} = \langle \mathcal{S}_{gA}^{\mathcal{F}}, \mathcal{A}^C, \mathsf{cost}_{gA}, \mathcal{T}_{gA}^{\mathcal{F}}, \mathcal{I}^{\mathcal{F}}, \mathcal{S}_{\mathcal{G}gA}^{\mathcal{F}} \rangle$ *is like* $\Theta_\Pi^{\mathcal{RF}}$*, but adapted as follows:*

For a decoupled state $s^{\mathcal{F}}$ *in* $\Theta_\Pi^{\mathcal{RF}}$*, let* $p_{min}^L(s^{\mathcal{F}})$ *be the minimum price of an* L*-state reached in* $s^{\mathcal{F}}$*, i. e.,* $p_{min}^L(s^{\mathcal{F}}) := \min_{s^L \in S^L} \mathsf{prices}(s^{\mathcal{F}})[s^L]$.

1. $gA(s^{\mathcal{F}}) \in \mathcal{S}_{gA}^{\mathcal{F}}$ *if* $s^{\mathcal{F}} \in \mathcal{S}^{\mathcal{F}}$*, where* $\mathsf{center}(gA(s^{\mathcal{F}})) := \mathsf{center}(s^{\mathcal{F}})$, $\pi^C(gA(s^{\mathcal{F}})) := \pi^C(s^{\mathcal{F}})$*, and for all leaf states* $s^L \in S^{\mathcal{L}}$: $\mathsf{prices}(gA(s^{\mathcal{F}}))[s^L] := \mathsf{prices}(s^{\mathcal{F}})[s^L] - p_{min}^L(s^{\mathcal{F}})$.

2. $gA(s^{\mathcal{F}}) \in \mathcal{S}_{\mathcal{G}gA}^{\mathcal{F}}$ *if* $s^{\mathcal{F}} \in \mathcal{S}_{\mathcal{G}}^{\mathcal{F}}$.

3. $gA(s^{\mathcal{F}}) \xrightarrow{a^C} gA(t^{\mathcal{F}}) \in \mathcal{T}_{gA}^{\mathcal{F}}$ *with cost* $\mathsf{cost}(a^C) + \sum_{L \in \mathcal{L}} p_{min}^L(t^{\mathcal{F}})$ *if* $s^{\mathcal{F}} \xrightarrow{a^C} t^{\mathcal{F}} \in \mathcal{T}^{\mathcal{F}}$ *with cost* $\mathsf{cost}(a^C)$.

The g-value adaptation can be maintained easily by constructing the reachable decoupled state space from $\mathcal{I}^{\mathcal{F}}$ and, for every successor state, removing the minimum leaf state price from the price of all reached leaf states and adding the sum over all leaves of these prices to the cost of the decoupled transition.

It is easy to see that the g-value adaptation does not affect optimality, but only shifts costs from the pricing function into the transition costs:

Proposition 7 (g-Adaptation preserves Optimality)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. Let* $s^{\mathcal{F}}$ *be a state reached in* $\Theta_\Pi^{\mathcal{RF}}$ *via* $\pi^{\mathcal{F}}$*, and* $gA(s^{\mathcal{F}})$ *its* g*-adapted representative in* $\Theta_\Pi^{\mathcal{RF}gA}$*. Then* $\mathsf{cost}(\pi^{\mathcal{F}}) + h_{\mathcal{F}}^*(s^{\mathcal{F}}) = \mathsf{cost}_{gA}(\pi^{\mathcal{F}}) + h_{\mathcal{F}}^*(gA(s^{\mathcal{F}}))$.

Proof. Note first that $\pi^{\mathcal{F}}$ indeed ends in $gA(s^{\mathcal{F}})$ in $\Theta_{\Pi}^{\mathcal{R}\mathcal{F}gA}$, because the cost adaptation does not effect decoupled paths, but only the transition costs. By construction, we have $\mathsf{cost}_{gA}(\pi^{\mathcal{F}}) = \mathsf{cost}(\pi^{\mathcal{F}}) + \sum_{L \in \mathcal{L}} p^L_{min}(s^{\mathcal{F}})$.

For the heuristic $h^*_{\mathcal{F}}(s^{\mathcal{F}})$, let s^L be the leaf state that it "bought" for leaf L by $h^*_{\mathcal{F}}$ in the buy-leaves compilation for $s^{\mathcal{F}}$, and let $\pi^L = \langle a[s^L], a^L_1, \dots, a^L_n \rangle$ be the sequence of L-actions underlying the optimal solution of cost h^* of $\Pi_{L\$}$. Then π^L is also a valid leaf path in the buy-leaves compilation for $gA(s^{\mathcal{F}})$, only that the cost of $a[s^L]$ is now $\mathsf{prices}(s^{\mathcal{F}})[s^L] - p^L_{min}(s^{\mathcal{F}})$. Since the price of all L-states in $gA(s^{\mathcal{F}})$ results from deducting $p^L_{min}(s^{\mathcal{F}})$ from their price in $s^{\mathcal{F}}$, the cost of every leaf path in the compilation is reduced by the same amount, so π^L has still the minimum cost among all leaf goal paths. Thus, $h^*_{\mathcal{F}}(gA(s^{\mathcal{F}})) = h^*_{\mathcal{F}}(s^{\mathcal{F}}) - \sum_{L \in \mathcal{L}} p^L_{min}(s^{\mathcal{F}})$, concluding the proof. □

By moving cost from the pricing function into the g-value we achieve that the heuristic of a decoupled state (which takes into account the pricing function) can only get lower, aiding A* to focus on more promising states. A second important effect is that the part of the prices moved into the g-value will always be considered entirely by the search, whereas heuristics are lossy (in the extreme case may ignore costs completely) and will typically not be able to capture all the cost represented in the leaf-state prices.

Chapter 5

Problem Decomposition – Factoring Strategies

In this chapter, we propose methods to identify star factorings, *factoring strategies*, in a given planning task. First, we characterize several special cases, namely fork, inverted-fork, and strict-star factorings, and present important properties of factorings. We then analyse the complexity of computing a factoring. While fork and inverted-fork factorings can be easily obtained via the strongly connected components of the causal graph, it is in general **NP**-hard to compute strict-star factorings that maximize the number of leaf components [Gnad and Hoffmann, 2018]. We introduce methods that are able to identify such factorings [Gnad et al., 2017a]. As an alternative, we formulate the factoring process as integer linear programming (ILP) and use an off-the-shelf ILP solver to find general star factorings [Schmitt et al., 2019].

5.1 Factoring Characteristics

Throughout this chapter we assume that the causal graph CG_Π of a planning task Π is always weakly connected. This is without loss of generality, because otherwise the task can equivalently be split into sub-tasks that can be solved independently.

Building on the causal graph, we capture the dependencies between factors as the quotient graph of the causal graph given the factoring. This allows us to identify important special cases regarding the interaction between factors. These often enable a more efficient handling, or specialized algorithmic optimizations. We introduce the interaction graph as a way to analyse cross-factor dependencies:

Definition 20 (Interaction Graph)**.** *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. The* interaction graph $\mathsf{IG}_\Pi^\mathcal{F}$ *of* Π *and* $\mathcal{F}$ *is a digraph with vertices* $\mathcal{F}$ *and edges* E*, where* $\langle F, F' \rangle \in E$ *iff* $F \neq F'$ *and there exist variables* $v \in F$ *and* $v' \in F'$ *such that* $\langle v, v' \rangle$ *is an edge in the causal graph* CG_Π *of* Π.

With the interaction graph, we can define special types of factorings, namely fork, inverted-fork, and strict-star factorings.

Definition 21 (Special-case Factorings)**.** *Let Π be a planning task and $\mathcal{F} = \{C\} \cup \mathcal{L}$ a factoring for Π with center factor C and leaves $\mathcal{L}$. Then $\mathcal{F}$ is:*

- *a* fork factoring*, iff all edges in $\mathsf{IG}^{\mathcal{F}}_{\Pi}$ are of the form $C \rightarrow L$, for $L \in \mathcal{L}$,*
- *an* inverted-fork factoring*, iff all edges in $\mathsf{IG}^{\mathcal{F}}_{\Pi}$ are of the form $L \rightarrow C$, for $L \in \mathcal{L}$,*
- *a* strict-star factoring*, iff there does not exist an edge $F \rightarrow F'$ in $\mathsf{IG}^{\mathcal{F}}_{\Pi}$ such that $F \in \mathcal{L}$ and $F' \in \mathcal{L}$.*

Intuitively, in fork factorings only the leaves depend on the center, but not vice versa; in inverted-fork factorings it is exactly the other way around. This leads to very specific behaviour of the pricing functions, which are *monotonous* in these cases. With fork factorings, leaf-state prices can only decrease along any decoupled path, because leaf paths that where compliant at some point will remain compliant forever (center actions do not impose any restriction because they have no leaf preconditions). In inverted-fork factorings, initially all leaf paths are compliant with the empty center path. Along a sequence of center actions, the set of compliant paths can only decrease, so leaf-state prices increase monotonously along any search path. Strict-star factorings do not lead to such monotonous behaviour. The difference to general-star factorings is that direct dependencies between leaves are forbidden, so a center action can at most affect one leaf. The presence of strict-star factorings indicates that the given planning task is less tightly coupled, so indicates higher potential for reduction.

In a general-star factoring $\mathcal{F}$, we call a leaf $L \in \mathcal{L}$ where the only connection in $\mathsf{IG}^{\mathcal{F}}_{\Pi}$ of L is of the form $C \rightarrow L$ *fork leaf*, and similarly if the connection is only $L \rightarrow C$ inverted-fork leaf.

Besides the cross-factor dependencies, there are two key properties that affect the state-space size reduction achieved by decoupled search, the number of leaf factors and the number of leaf-only actions. The reduction can be estimated by the number of explicit states that can be contained in a decoupled state $s^{\mathcal{F}}$, formally $|[s^{\mathcal{F}}]| = \prod_{L \in \mathcal{L}} |S^L_R|$, where S^L_R is the number of leaf states of L that are reached in $s^{\mathcal{F}}$. Thus, the number of leaf factors plays a crucial role in the potential for reduction, because the number of explicit states that is contained in a decoupled state is exponential in the number of leaves. We will therefore always aim at maximizing the number of leaf factors.

The second important factor is the number of *simultaneously* reached leaf states. A simple upper bound is given by the number of states of a leaf L, namely $\prod_{v \in L} |\mathcal{D}(v)|$. So it appears beneficial to maximize the size of the leaves to maximize the reduction. From a practical perspective, though, there is an important trade-off between the potential reduction and the efficiency of decoupled-state generation, which involves updating the pricing function. Since even when a good reduction is achieved, we expect that the search is required to explore many decoupled states, the size of the leaf state spaces needs to be kept reasonably small.[1] Moreover, the bound provides little knowledge about the number of leaf states that are reached at the same time in a decoupled state. We try to capture this with the notion of *mobility*, instead, which is based on the number of leaf-only actions of a leaf.

Definition 22 (Mobility)**.** *Let* Π *be a planning task and* $\mathcal{F}$ *a factoring for* Π*. A leaf factor* $L \in \mathcal{L}$ *is* mobile *if there exists a leaf-only action affecting* L*, i. e.,* $|\mathcal{A}^L_{\not{C}}| > 0$*. A factoring is mobile if there exists at least one center action and all its leaves are mobile. The* mobility *of a leaf factor* L *is defined as the number of its leaf-only actions, i. e.,* $|\mathcal{A}^L_{\not{C}}|$*. The* mobility *of a factoring* $\mathcal{F}$ *is the sum of the mobility of its leaves, i. e.,* $\sum_{L \in \mathcal{L}} |\mathcal{A}^L_{\not{C}}|$*.*

We require that at least one center action exists to avoid pathologic cases where all actions are leaf-only and there is no actual search.

Intuitively, the mobility captures the amount of "work" a leaf can do on its own, i. e., how much can be delegated from the main search to the leaves. Note that fork and inverted-fork leaves are always mobile, unless there are static variables in Π, i. e., variables that are not affected by any action. This is because there are no center actions that affect such leaves. Thus, for every (inverted-)fork leaf L, we have $\mathcal{A}^L = \mathcal{A}^L_{\not{C}}$. A leaf factor that is not mobile does not contribute to the state-space reduction.

Proposition 8 (Non-mobile Leaves)**.** *Let* Π *be a planning task and* $\mathcal{F}$ *a factoring for* Π *with leaves* $\mathcal{L}$*. If a leaf* $L \in \mathcal{L}$ *is not mobile, then in all decoupled states* $s^\mathcal{F}$ *reachable from the initial decoupled state* $\mathcal{I}^\mathcal{F}$ *there exists exactly one* L*-state* s^L *that is reached in* $s^\mathcal{F}$*, i. e., where* $\mathsf{prices}(s^\mathcal{F})[s^L] < \infty$*.*

Proof. The claim is true in the initial state, since there exists no leaf-only action for L, so $\mathsf{prices}(\mathcal{I}^\mathcal{F})[\mathcal{I}[L]] = 0$ and $\mathsf{prices}(\mathcal{I}^\mathcal{F})[s^L] = \infty$ for all $s^L \neq \mathcal{I}[L]$. Let $s^\mathcal{F}$ be a successor of $\mathcal{I}^\mathcal{F}$ via center action a^C. Then the leaf state $s^L := \mathcal{I}[L][\![a^C]\!]$ has a price of 0 in $s^\mathcal{F}$. Again, because there are no leaf-only actions of L, no other leaf state of L is reached in $s^\mathcal{F}$. This argument applies inductively to all decoupled states reachable from $\mathcal{I}^\mathcal{F}$. □

From Proposition 8, we get that in the extreme case where no leaf is mobile, every decoupled state represents exactly one explicit state and the search needs to branch over all actions since there are no leaf-only actions. As a consequence, decoupled search degrades to explicit-state search and there is no reduction.

In the methods presented in the next sections, we aim at obtaining mobile factorings with a maximum number of leaves. While this optimization condition may not sound

[1]Empirically, leaf factors with up to 10.000 reachable states can be handled quite well.

too complicated, it turns out that even maximizing the number of leaves in a strict-star factoring is **NP**-hard.

5.2 Complexity

We next take a look at the conditions under which factorings exist. First, we show that there is an easy-to-check property that implies that no mobile factoring exists. We then provide a construction to determine if there exists a mobile factoring with at least two leaves. Finally, we analyse the complexity of computing a factoring with the maximum number of leaves.

Regarding the existence of a mobile factoring, observe that for a leaf to be mobile we need an action that only affects variables of that leaf. The other way around, if there exists a variable v that is affected by *all* actions, then no mobile factoring exists.

To ensure that there also exists a center action, we need to forbid *static* variables, i. e., variables that are not affected by any action. More formally, a variable $v \in \mathcal{V}$ is *static* if there does not exist an action $a \in \mathcal{A}$ such that $v \in \mathsf{vars}(\mathsf{eff}(a))$.

Theorem 8 (Existence of Mobile Factoring)**.** *Let* Π *be a planning task without static variables. There exists a mobile factoring* $\mathcal{F} = \{C\} \cup \mathcal{L}$ *for* Π *where* $|\mathcal{L}| \geq 1$ *iff there exist a variable* $v \in \mathcal{V}$ *and an action* $a \in \mathcal{A}$ *such that* $v \notin \mathsf{vars}(\mathsf{eff}(a))$.

Proof. Assume $\mathcal{F} = \{C\} \cup \mathcal{L}$ is a mobile factoring with center C and at least one leaf $L \in \mathcal{L}$. Since $\mathcal{F}$ is mobile, there exists a leaf-only action $a \in \mathcal{A}^L_{\not{C}}$ for L where $\mathsf{vars}(\mathsf{pre}(a)) \subseteq C \cup L$ and $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L$. Thus, there exists a variable $v \in C$ such that $v \notin \mathsf{vars}(\mathsf{eff}(a))$.

Let Π be a task with a variable $v \in \mathcal{V}$ and an action $a \in \mathcal{A}$ such that $v \notin \mathsf{vars}(\mathsf{eff}(a))$. We construct a mobile factoring $\mathcal{F} = \{C, L\}$ with a leaf $L = \mathsf{vars}(\mathsf{eff}(a))$ and center $C = \mathcal{V} \setminus L$. First, because $v \notin \mathsf{vars}(\mathsf{eff}(a))$ $\mathcal{F}$ is a partition of the variables $\mathcal{V}$ with non-empty center $C \supseteq \{v\}$. All actions $a' \in \mathcal{A}$ with an effect only on L are leaf-only actions of L, where trivially $\mathsf{vars}(\mathsf{pre}(a')) \subseteq C \cup L$. All other actions are center actions. The leaf L is mobile because a is a leaf-only action for L. Because Π does not contain static variables, so v appears in the effect of some action, there also exists a center action. Thus, $\mathcal{F}$ is a mobile star factoring. □

With this result, we can efficiently check a sufficient condition for the existence of a mobile factoring with at least one leaf. If there exists a variable that appears in the effect of all actions, then no mobile factoring exists.

As pointed out in the previous section, we will always aim at maximizing the number of leaf factors, because the state-space reduction of decoupled search is exponential in that number. Therefore, we will only even start decoupled search if there are at least *two* mobile leaves. We next show how to efficiently check if a factoring with at least two mobile leaves exists. To do so, we introduce the concept of *action schemas*, which group together actions that are preconditioned by and affect the same sets of variables.

Definition 23 (Action Schema)**.** *Let* Π *be a planning task. An* action schema A *of* Π *is a pair of variable subsets* $\langle \mathsf{pre}(A), \mathsf{eff}(A) \rangle$ *such that there exists an action* $a \in \mathcal{A}$ *where* $\mathsf{vars}(\mathsf{pre}(a)) = \mathsf{pre}(A)$ *and* $\mathsf{vars}(\mathsf{eff}(a)) = \mathsf{eff}(A)$*.*

An important observation is that to obtain a mobile factoring each leaf must be a superset of the effect variables of an action. Thus, mobile factorings can be computed at the granularity of the action schema effects, and there is no benefit in considering individual variables. This leads to the result that for a mobile factoring with at least two leaves, there need to be two action schemas with non-overlapping effects. To ensure a proper star factoring, we additionally need that there is a non-empty center, and that leaf-only actions do not have preconditions on another leaf:

Theorem 9 (Existence of Mobile 2-Leaf Factoring)**.** *Let* Π *be a planning task without static variables. We can construct a mobile factoring* $\mathcal{F} = \{C\} \cup \mathcal{L}$ *for* Π *with* $|\mathcal{L}| \geq 2$ *iff there exist action schemas* A_1, A_2 *such that* $\mathsf{eff}(A_1) \cup \mathsf{eff}(A_2) \subset \mathcal{V}$*,* $\mathsf{eff}(A_1) \cap \mathsf{eff}(A_2) = \emptyset$*,* $\mathsf{eff}(A_1) \cap \mathsf{pre}(A_2) = \emptyset$*, and* $\mathsf{pre}(A_1) \cap \mathsf{eff}(A_2) = \emptyset$*.*

Proof. Let A_1, A_2 be action schemas such that $\mathsf{eff}(A_1) \cup \mathsf{eff}(A_2) \subset \mathcal{V}$, $\mathsf{eff}(A_1) \cap \mathsf{eff}(A_2) = \emptyset$, $\mathsf{eff}(A_1) \cap \mathsf{pre}(A_2) = \emptyset$, and $\mathsf{pre}(A_1) \cap \mathsf{eff}(A_2) = \emptyset$. We construct a factoring $\mathcal{F}$ with center C and two leaves L_1, L_2 as follows: $L_1 = \mathsf{eff}(A_1)$, $L_2 = \mathsf{eff}(A_2)$, and $C = \mathcal{V} \setminus (L_1 \cup L_2)$. We next show that $\mathcal{F}$ is a proper factoring with mobile leaves.

(i) By construction $\mathcal{F}$ is a partition of $\mathcal{V}$; all factors are non-empty. The latter is true for the center C because $\mathsf{eff}(A_1) \cup \mathsf{eff}(A_2) \subset \mathcal{V}$.

(ii) If there exists an action without effects on C, but (a) with effects on both L_1 and L_2, or (b) with effects on L_1 and precondition on L_2, or vice versa, then we add an auxiliary variable v_{aux} with $\mathcal{D}(v_{aux}) = \{0\}$ to Π. We set $\mathcal{I}[v_{aux}] = 0$ and add a redundant effect $\mathsf{eff}(a)[v_{aux}] = 0$ to all actions as of (a) and (b). Then all actions are either leaf-only actions of one of the leaves, or have a center effect, so $\mathcal{F} = \{C, L_1, L_2\}$ is a proper star factoring.

(iii) Each leaf factor L_i is mobile because (a) by construction there exists an action $a \in \mathcal{A}$ that affects only L_i, and because (b) $\mathsf{pre}(A_1) \cap \mathsf{eff}(A_2) = \emptyset$ there exists an action such that $\mathsf{vars}(\mathsf{pre}(a)) \subseteq C \cup L_1$. The same argument also holds for L_2.

(iv) Since there are no static variables, there exists an action affecting C.

By (i) and (ii), $\mathcal{F}$ is a star factoring, from (iii) and (iv) we get that $\mathcal{F}$ is mobile. The auxiliary variable introduced in (ii) is redundant and does not have any effect on Π.

Let now $\mathcal{F} = \{C\} \cup \mathcal{L}$ be a mobile factoring with center C and at least two leaves $L_1 \neq L_2 \in \mathcal{L}$. Because $\mathcal{F}$ is mobile, there exists a leaf-only action a_i for each of the leaves L_i where $\mathsf{vars}(\mathsf{pre}(a_i)) \subseteq C \cup L_i$ and $\mathsf{vars}(\mathsf{eff}(a_i)) \subseteq L_i$. Thus, such a_1 and a_2 are part of action schemas A_1 and A_2 where $\mathsf{eff}(A_1) \cap \mathsf{eff}(A_2) = \emptyset$, and where $\mathsf{eff}(A_i) \cap \mathsf{pre}(A_j) = \emptyset$ for $\{i, j\} = \{1, 2\}$. Furthermore, we have $\mathsf{eff}(A_1) \cup \mathsf{eff}(A_2) \subset \mathcal{V}$ because C is non-empty. □

The construction from point (ii) in the proof serves to meet the requirement that center actions have an effect on a center variable. The factoring methods we propose later on do not make use of this construction. Still, the conditions named in Theorem 9 are necessary for the existence of mobile two-leaf factorings. Thus, as they can be checked efficiently, we will implement them to decide if a factoring is possible before actually attempting to find a factoring.

Concluding this section, we prove that it is **NP**-complete to compute a, not necessarily mobile, strict-star factoring with the maximum number of leaves. Hardness follows from a construction based on computing maximum independent sets of the causal graph of a planning task. First, we prove that this construction is correct:

Lemma 5 (Factoring from Independent Set). *Let Π be a planning task. From a strict-star factoring $\mathcal{F}$ for Π with n leaves we can construct an independent set of size n of CG_Π, and vice versa.*

Proof. Let $\mathcal{F} = \{C, L_1, \dots, L_n\}$ be a strict-star factoring. Since it is a strict-star factoring, there are no edges between any pair of leaves in $\mathsf{IG}_\Pi^\mathcal{F}$. Thus, for any pair of variables v_i, v_j, where $v_i \in L_i$, $v_j \in L_j$, and $i \neq j$, there is no edge $\langle v_i, v_j \rangle$ in CG_Π. We can therefore select an arbitrary variable from each leaf, and the set of these variables forms an independent set in CG_Π.

Let $\{v_1, \dots, v_n\}$ be an independent set of CG_Π. We can construct a factoring $\mathcal{F} = \{C, L_1, \dots, L_n\}$ with n leaves as follows: $L_i = \{v_i\}$, $C = \mathcal{V} \setminus \{v_1, \dots, v_n\}$. Since the variables v_i are not connected in the causal graph, there are no edges between any leaves in $\mathsf{IG}_\Pi^\mathcal{F}$, so $\mathcal{F}$ is a strict-star factoring. □

While we can construct strict-star factoring with this approach, it is not reasonable to obtain factorings, because there is no guarantee that the leaves are mobile. Using the construction, we next show **NP**-completeness for maximizing the number of leaves:

Theorem 10 (Maximize Number of Leaves). *Let Π be a planning task. The maximum number of leaves in a strict-star factoring equals the size of a maximum independent set in CG_Π. Given $n \in \mathbb{N}$, it is **NP**-complete to decide if the maximum number of leaves in a strict-star factoring is n.*

Proof. Follows immediately from Lemma 5 and the fact that computing a Maximum Independent Set is **NP**-complete [Garey and Johnson, 1979]. □

This result allows to compute an upper bound on the number of leaves in strict-star factorings, and thereby a bound for more restricted factorings, too. It does not have implications for general-star factorings, though, nor on the number of *mobile* leaves in strict-star factorings.

5.3 Factoring Based on the Causal Graph

In this section, we introduce several factoring strategies that identify strict-star factorings. The strategies are based on an analysis of the causal graph CG_Π. First, we show how fork and inverted-fork can be obtained from the strongly connected components (SCCs) of CG_Π [Gnad and Hoffmann, 2018]. We then propose two greedy algorithms, one based on computing a maximum independent set (MIS) of the causal graph, another based on the connectivity of variables in CG_Π [Gnad et al., 2017a]. A base variant of the algorithm in Figure 5.1, as well as a first version of its implementation, is due to Valerie Poser. She also contributed an extended variant of the algorithm in Figure 5.2, which is not presented in this work. The author of this work contributed the underlying ideas of both algorithms, and refined and simplified the algorithm in Figure 5.1.

For fork and inverted-fork factorings, observe that each causal graph SCC must be subsumed by a factor. If an SCC is split across multiple factors, this implies a bidirectional connection between these factors in the interaction graph. Thus, (inverted-)fork factorings need to be computed at the level of SCCs. Let $\mathtt{SCCs}(\mathcal{G})$ be a function that returns the strongly connected components of a graph $\mathcal{G}$. Interpreting $\mathtt{SCCs}(\mathsf{CG}_\Pi)$ as a factoring (it is not guaranteed to be a star factoring), we can construct the interaction graph $\mathsf{IG}_\Pi^{\mathtt{SCCs}(\mathsf{CG}_\Pi)}$ as before. From this graph, we can obtain a (inverted-)fork factoring by selecting the leaf (root) vertices of $\mathsf{IG}_\Pi^{\mathtt{SCCs}(\mathsf{CG}_\Pi)}$ to be leaf factors and moving the remaining variables to the center. In fact, this maximizes the number of leaves in any (inverted-)fork factoring:

Theorem 11 (Fork & Inverted-Fork Factorings). *Let Π be a planning task. Then fork and inverted-fork factorings exist iff $|\mathtt{SCCs}(\mathsf{CG}_\Pi)| > 1$. The maximum number of leaves in a fork (inverted-fork) factoring equals the number of leaf (root) vertices in $\mathsf{IG}_\Pi^{\mathtt{SCCs}(\mathsf{CG}_\Pi)}$.*

Proof. The "only if" direction in the first part of the claim holds because any fork or inverted-fork factoring must be coarser than $\mathtt{SCCs}(\mathsf{CG}_\Pi)$. The "if" direction follows from the second part of the claim.

We prove the second part of the claim for fork factorings; the argument for inverted forks is symmetric because those are equivalent to forks in the modification of $\mathsf{IG}_\Pi^{\mathtt{SCCs}(\mathsf{CG}_\Pi)}$ where the direction of each arc is inverted.

Denote by K the maximum number of fork leaves. Denote by K' the number of leaves in $\mathsf{IG}_\Pi^{\mathtt{SCCs}(\mathsf{CG}_\Pi)}$.

$K \geq K'$ holds because we obtain a fork factoring by setting the leaves in $\mathsf{IG}_{\Pi}^{\mathtt{SCCs}(\mathsf{CG}_{\Pi})}$ to be the leaf factors, and taking the remaining state variables to form the center.

$K \leq K'$ holds because any fork leaf must be closed under following arcs in the causal graph, and hence every fork leaf must contain at least one leaf in $\mathsf{IG}_{\Pi}^{\mathtt{SCCs}(\mathsf{CG}_{\Pi})}$. Precisely, let $\mathcal{F}$ be any fork factoring, and let L be any leaf factor of $\mathcal{F}$. Let F, F' be components in $\mathtt{SCCs}(\mathsf{CG}_{\Pi})$ such that $F \rightarrow F'$ is an arc in $\mathsf{IG}_{\Pi}^{\mathtt{SCCs}(\mathsf{CG}_{\Pi})}$. If $F \subseteq L$, then we must have $F' \subseteq L$: otherwise, either $F' \subseteq C$ in contradiction, or $F' \subseteq L_2$ for some other leaf factor L_2 in contradiction. As $\mathcal{F}$ is coarser than $\mathtt{SCCs}(\mathsf{CG}_{\Pi})$, there exists at least one F where $F \subseteq L$. Applying the argument transitively starting from F, we obtain a leaf F'' in $\mathsf{IG}_{\Pi}^{\mathtt{SCCs}(\mathsf{CG}_{\Pi})}$ so that $F'' \subseteq L$. But then, as the leaf factors in $\mathcal{F}$ are disjoint, there cannot be more leaf factors than leaves in $\mathsf{IG}_{\Pi}^{\mathtt{SCCs}(\mathsf{CG}_{\Pi})}$, concluding the argument. $\square$

This result immediately provides us with an efficient algorithm to compute fork and inverted-fork factorings with maximum number of leaf factors. We also experimented with variants that (1) combine several SCCs into one leaf, and (2) combine fork with inverted-fork leaves into what is called an X-shape factoring. For (1), there are some planning domains in which this proved useful. In general, however, it is not clear when it is beneficial to combine leaves, since, as mentioned before, there is always a trade-off between potential reduction and the efficiency of handling large leaf state spaces. For (2), say we have a fork factoring $\mathcal{F}$ with leaves $\mathcal{L}$ and center C, where C contains causal-graph root SCCs $\mathcal{L}_R$. We can then make these SCCs additional leaves, unless this introduces an edge $L_R \rightarrow L$ in the interaction graph between any $L_R \in \mathcal{L}_R$ and an $L \in \mathcal{L}$.

Factoring strategies that are based on the causal graph SCCs are restricted to planning tasks where the causal graph is not strongly connected. We next introduce two strategies that lift this restriction, identifying strict-star factorings even if the causal graph is a single SCC.

Our first algorithm is presented in Figure 5.1 and is based on the construction in the proof of Lemma 5, namely the computation of a maximum independent set of the causal graph. Let $\mathtt{MIS}(\mathcal{G})$ be a function that returns a maximum independent set of a graph. Then we obtain an initial factoring $\mathcal{F}$ by making every variable v in $\mathtt{MIS}(\mathsf{CG}_{\Pi})$ a separate leaf factor. This, however, does not necessarily result in a *mobile* factoring. Therefore, we post-process $\mathcal{F}$ by identifying center variables v_L that are only connected to one of the leaves L. We move such v_L into the corresponding leaf L, since this can only lead to a higher mobility of L and ensures that $\mathcal{F}$ is still a strict-star factoring.

After considering all such variables, we remove leaves from $\mathcal{L}$ that are still not mobile. We do so because there is no easy way to make these leaves mobile, since this would introduce a dependency to another leaf. One could attempt to merge several leaves if there exists a shared center variable with causal graph connection to all of them. We experimented with several strategies for doing so, but did not obtain any conclusive results. Having removed the non-mobile leaves, we try to increase the mobility

```
1  **IndependentSetFactoring**($\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$):
2      $\mathcal{L} \leftarrow \{\{v\} \mid v \in \mathtt{MIS}(\mathsf{CG}_\Pi)\}$
3      $\mathcal{L} \leftarrow$ **maximizeMobility**($\mathcal{L}$)
4      $\mathcal{L} \leftarrow \{L \in \mathcal{L} \mid L$ is mobile$\}$
5      $\mathcal{L} \leftarrow$ **maximizeMobility**($\mathcal{L}$)
6      $C \leftarrow \{\mathcal{V} \setminus \bigcup_{L \in \mathcal{L}} L\}$
7      **return** $\mathcal{F} \leftarrow \{C\} \cup \mathcal{L}$
8  **maximizeMobility** ($\mathcal{L}_0$):
9      $C \leftarrow \{\mathcal{V} \setminus \bigcup_{L \in \mathcal{L}_0} L\}$
10     $V' \leftarrow \{v \in C \mid |$**neighbourLeaves**$(v, \mathcal{L}_0)| = 1\}$
11     $\mathcal{L}_{\max} \leftarrow \mathcal{L}_0$
12     $i \leftarrow 1$
13     **for** $v \in V'$ **do**
14         $\{L_v\} \leftarrow$ **neighbourLeaves**$(v, \mathcal{L}_{\max})$
15         $\mathcal{L}_i \leftarrow \mathcal{L}_{\max} \cup \{L_v \cup \{v'\}\} \setminus \{L_v\}$
16         $i \leftarrow i + 1$
17     **end**
18     **return** $\mathcal{L}_j$ with max # of mobile leaves
19 **neighbourLeaves** ($v, \mathcal{L}$):
20     **return** $N \leftarrow \{L \in \mathcal{L} \mid \exists v' \in L \; s.t. \; v \rightarrow v' \in \mathsf{CG}_\Pi \vee v' \rightarrow v \in \mathsf{CG}_\Pi\}$
```

Figure 5.1: Factoring strategy based on maximum independent sets of the causal graph.

of the remaining leaves again by extending them with center variables. This is possible because a variable that was part of a leaf that has been removed may have put a restriction on the choice of variables with only a single neighbouring leaf in the first iteration in line 10 of the algorithm.

The resulting algorithm, **IndependentSetFactoring**, by construction always returns strict-star factorings. It is not complete, though, neither guaranteeing to find a factoring if one exists, nor returning the factoring with the maximum number of leaves (in case a leaf has been removed in line 4).

Last in this section, we include another greedy algorithm that identifies strict-star factorings. The **IncidentArcsFactoring** algorithm is shown in Figure 5.2. It is based on the connectivity of variables in the causal graph, namely the number of *incident arcs* of a variable. Formally, the number of incident arcs of a variable v in CG_Π is the number of edges it is connected to, $|\{(v_1, v_2) \in \mathsf{CG}_\Pi \mid v \in \{v_1, v_2\}\}|$. We sort the variables $\mathcal{V}$ by decreasing number of incident arcs and start by moving the most densely connected variables to the center C. The rationale behind this is that having such variables in a leaf factor will introduce inter-leaf dependencies. Consequently, the key idea is that after moving densely connected variables to the center, the causal graph *without* these

```
1  **IncidentArcsFactoring**($\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$):
2      $C \leftarrow \emptyset$
3      $i \leftarrow 1$
4      for $v \in \mathcal{V}$ do
           // $\mathcal{V}$ sorted by decreasing # of incident arcs in $\mathsf{CG}_\Pi$
5          $C \leftarrow C \cup \{v\}$
6          $\mathcal{L}_i \leftarrow \mathtt{WCCs}(\mathsf{CG}_{\Pi'})$, where $\Pi' := \langle \mathcal{V} \setminus C, \mathcal{A}|_{\mathcal{V} \setminus C}, \mathsf{cost}, \mathcal{I}[\mathcal{V} \setminus C], \mathcal{G}[\mathcal{V} \setminus C] \rangle$
7          $i \leftarrow i + 1$
8      end
9      $\mathcal{L} \leftarrow$ select $\mathcal{L}_i$ with max # of mobile leaves
10     $\mathcal{L} \leftarrow \{L \in \mathcal{L} \mid L$ is mobile$\}$
11     $C \leftarrow \{\mathcal{V} \setminus \bigcup_{L \in \mathcal{L}} L\}$
12     return $\mathcal{F} \leftarrow \{C\} \cup \mathcal{L}$
```

Figure 5.2: A greedy factoring strategy based on the number of incident arcs of a variable in the causal graph.

variables will be disconnected, separated into several non-connected components that will become leaves.

We move variables into C one by one, every time generating a potential factoring with leaves $\mathcal{L}_i$. For each iteration, we compute the weakly connected components in the causal graph projected on the remaining variables. Concretely, let C_i be the center factor after moving the i most densely connected variables to the center. Then we project the task onto $\mathcal{V} \setminus C$, removing all occurrences of the respective variables from Π, obtaining a projected task Π'. Let $\mathtt{WCCs}(\mathcal{G})$ be a function that returns the weakly connected components of a graph $\mathcal{G}$. On the projected task, we make each component L in $\mathtt{WCCs}(\mathsf{CG}_{\Pi'})$ a leaf factor. Note that these leaves are not necessarily mobile, since for a leaf L there might only be actions that affect L *and* a center variable at the same time. Therefore, we need to remove the non-mobile leaves.

Upon termination, we pick the factoring with the maximum number of mobile leaves in $\mathcal{L}_i$. Similar to the previously introduced MIS-based algorithm, **IncidentArcsFactoring** does not provide any guarantees regarding the quality of the factoring, nor for finding a factoring if one exists. Still, empirically, both algorithms provide good results. As we will see in Chapter 7.3, many standard benchmark domains can be decomposed into strict-star factorings. There is also a significant number of domains that allow for (inverted-)fork factorings.

5.4 Factoring via Integer Linear Programming

All contributions in this section are due to Schmitt [2018]. The author of this work only acted as an advisor for the Bachelors thesis and came up with the idea *that* the factoring process could be formulated as an integer linear program (ILP). We include the approach here because it presents the only complete and optimal algorithm to obtain strict- and general-star factorings with the maximum number of leaves (maximum mobility). The idea of using action schemas (introduced in the previous section) is based on the effect schemas from Schmitt [2018].

In this section, we view the factoring process as an optimization problem that assigns variables to leaf factors while maximizing an objective function. We formulate the problem as an ILP, optimizing either the number of mobile leaf factors, or the mobility of the factoring. Based on the observation that a mobile leaf must subsume the effect variables of at least one action, we construct the ILP such that only factorings on the granularity of *effect schemas* are considered, which we introduce next.

We also experimented with an ILP encoding of factorings that searches over all possible combinations of variables across any number of leaf factors. While this is interesting conceptually, possibly resulting in factorings that are different from the ones we obtain with the methods introduced next, it is infeasible to solve the generated ILPs for non-trivial planning instances, due to the high number of constraints. Furthermore, the method never resulted in better factorings when given reasonable runtime limits. Therefore, we herein only introduce the encoding that builds on effect schemas. This avoids the complete search of the naive approach while preserving the guarantee that we can construct a factoring that maximizes the number of mobile leaf factors. So, in fact, a complete search over all possible assignments of variables to leaves is not required to maximize that number.

We next define the effect schemas of a planning task and based on these the *potential leaves* for a factoring $\mathcal{F}$.

Definition 24 (Effect Schema)**.** *Let* Π *be a planning task. An* effect schema $E \subseteq \mathcal{V}$ *is a subset of the variables of* Π*, such that there exists an action* $a \in \mathcal{A}$ *with* $\mathsf{vars}(\mathsf{eff}(a)) = E$*. We denote the set of all effect schemas of a task by* $\mathbf{ES}_\Pi$.

As explained before, if we want to obtain a mobile factoring $\mathcal{F}$, then every leaf $L \in \mathcal{L}$ must be the superset of at least one effect schema $E \in \mathbf{ES}_\Pi$. Every leaf L for which there does not exist an effect schema $E \subseteq L$ cannot be mobile. Based on this observation, we consider each effect schema $E \in \mathbf{ES}^*_\Pi := \mathbf{ES}_\Pi \setminus \{\mathcal{V}\}$ as a *potential leaf*, and construct a graph with nodes $\mathbf{ES}^*_\Pi$, such that any independent set of the graph forms a proper factoring. These graphs are different for strict and general-star factorings, so we introduce them separately. We will then encode the computation of a maximum independent set of these graphs as an ILP and formulate the objective function accordingly.

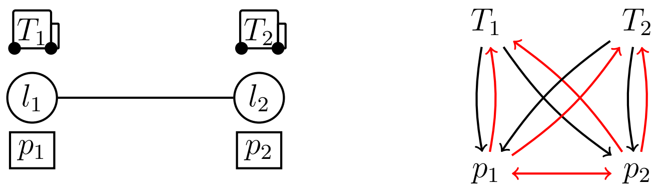


Figure 5.3: Illustration and causal graph of the running example.

Example 7. *As example, we will consider a variant of our previous logistics task* Π *with two trucks* T_1, T_2 *and two packages* p_1, p_2 *on a map with two locations. The example is illustrated in the left of Figure 5.3; in the right, there is the causal graph* CG_Π*. The task is defined as usual, with an additional action loadAllDrive*(T_x, l_y, l_z) *that loads all packages into a truck and drives the truck at once:* $\mathsf{pre}(\textit{loadAllDrive}(T_x, l_y, l_z)) = \{T_x = l_y, p_1 = l_y, p_2 = l_y\}$, $\mathsf{eff}(\textit{loadAllDrive}(T_x, l_y, l_z)) = \{T_x = l_z, p_1 = T_x, p_2 = T_x\}$. *The initial and goal state are not of interest to us, as they do not affect the factoring.*

Note that the red edges in the causal graph are only induced by the loadAll$()$ *actions. Without these, fork and inverted-fork factorings would be possible with (a subset of) the trucks or packages as leaves. With these actions, no such factorings are possible, and only the two algorithms from the previous section are able to find strict-star factorings where, e. g., the trucks are the leaves.*

In the next two sections, we introduce the graph structures that can be employed to compute strict- and general-star factorings based on a maximum independent set. For both types of graphs, we prove the correctness and optimality of the approach for obtaining mobile factorings. In Chapter 5.4.3, we describe the ILP encoding and the objective functions we want to maximize.

5.4.1 Strict-Star Factorings

As a basis for the ILP encoding, we next introduce the *potential strict-leaf graph*, which captures the overlap and dependencies between the potential leaves.

Definition 25 (Potential Strict-Leaf Graph)**.** *Let* Π *be a planning task and* CG_Π *its causal graph. The* potential strict-leaf graph $\mathbf{PSLG}_\Pi(\mathcal{S})$ *is an undirected graph with vertices* $\mathcal{S} \subseteq \mathcal{P}(\mathcal{V}) \setminus \{\emptyset, \mathcal{V}\}$*, where* $\mathcal{P}(\mathcal{V})$ *denotes the powerset of* $\mathcal{V}$*, and edges (i)* (S, S') *if* $S \neq S'$ *and* $S \cap S' \neq \emptyset$*, and (ii)* (S, S') *if* $S \neq S'$ *and there exist* $v \in S$ *and* $v' \in S'$*, such that* (v, v') *is an edge in* CG_Π.

The potential strict-leaf graph $\mathbf{PSLG}_\Pi(\mathcal{S})$ has an edge between (i) every pair of overlapping variable sets, and (ii) variable sets that are connected via their variables in the causal graph. Note that this exactly captures the requirements of a strict-star factoring, in that two connected sets cannot both be leaves.

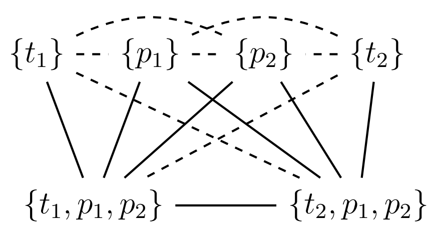


Figure 5.4: The potential strict-leaf graph of the example.

Example 8. *Figure 5.4 shows the potential strict-leaf graph $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$ of our example, where type-(ii)-only edges are dashed, and all solid edges are both from type (i) and type (ii). The (only) mobile strict-star factoring with the maximum number of two leaves has the packages in the center $C = \{p_1, p_2\}$ and each truck as a leaf $\mathcal{L} = \{\{T_1\}, \{T_2\}\}$. All other factorings have only a single leaf and result from $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$ by picking any one node in the graph as leaf and putting the remaining variables into the center.*

We next show that independent sets of the potential strict-leaf graph indeed are in one-to-one correspondence with strict-star factorings:

Lemma 6 (Strict-Star Factoring from Independent Set)**.** *Let Π be a planning task and let $\mathcal{S} = \mathcal{P}(\mathcal{V}) \setminus \{\emptyset, \mathcal{V}\}$. Then $\mathcal{F} = \{L_1, \dots, L_n, C\}$ is a strict-star factoring for Π, iff $I = \{L_1, \dots, L_n\}$ forms an independent set of $\mathbf{PSLG}_\Pi(\mathcal{S})$, where $C = \mathcal{V} \setminus \bigcup_{i=1}^n L_i$.*

Proof. From left to right, let $\mathcal{F} = \{L_1, \dots, L_n, C\}$, with all $L_i \in \mathcal{S}$, be a strict-star factoring. We prove that $I = \{L_1, \dots, L_n\}$ forms an independent set in $\mathbf{PSLG}_\Pi(\mathcal{S})$. Assume for contradiction that there exist $L_i, L_j \in I$, with $L_i \neq L_j$, such that (L_i, L_j) is an arc in $\mathbf{PSLG}_\Pi(\mathcal{S})$. Such an arc exists if (i) $L_i \cap L_j \neq \emptyset$, or (ii) there exist variables $v_i \in L_i$ and $v_j \in L_j$ such that (v_i, v_j) is an arc in CG_Π. Reason (i) contradicts the assumption that $\mathcal{F}$ is a valid factoring, i. e., a partitioning of $\mathcal{V}$; (ii) is in contradiction to $\mathcal{F}$ forming a strict-star factoring, as it implies a causal-graph arc between variables in L_i and L_j.

From right to left, let $I = \{L_1, \dots, L_n\}$ be an independent set of $\mathbf{PSLG}_\Pi(\mathcal{S})$. First, we prove that $\mathcal{F} = \{L_1, \dots, L_n, C\}$ with $C = \mathcal{V} \setminus L_1 \cup \dots \cup L_n$ is a valid factoring, i. e., a partitioning of $\mathcal{V}$. By definition, every element of I is a non-empty proper subset of $\mathcal{V}$. From (i) in the definition of $\mathbf{PSLG}_\Pi(\mathcal{S})$ it follows that all elements of $\mathcal{F}$ are pairwise disjoint. Furthermore, from the definition of C it follows that $L_1 \cup \dots \cup L_n \cup C = \mathcal{V}$. It remains to show that C is non-empty. If $|I| = 1$, then $C \neq \emptyset$, because $\mathcal{V} \notin \mathcal{S}$. Otherwise, assume for contradiction that $C = \emptyset$. Then I forms a partitioning of $\mathcal{V}$ such that for all $L_i, L_j \in I$, with $L_i \neq L_j$, it holds that there exist no variables $v_i \in L_i$ and $v_j \in L_j$ such that (v_i, v_j) is an arc in CG_Π. This contradicts our general assumption that CG_Π is weakly connected. Finally, we prove that $\mathcal{F}$ forms a strict-star factoring. Assume for contradiction that there exists an arc (v_i, v_j) in CG_Π, where $v_i \in L_i$ and $v_j \in L_j$, connecting L_i and L_j. This is in contradiction to type (ii) edges in the definition of $\mathbf{PSLG}_\Pi(\mathcal{S})$, since I forms an independent set in the graph. □

With Lemma 6, and the observation that every leaf that subsumes an effect schema is mobile, we can construct *mobile* strict-star factorings from the independent sets of $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$, and vice versa.

Theorem 12. *Let* Π *be a planning task and let* $\mathbf{ES}^*_\Pi$ *be the set of its potential leaves. Then from an independent set of size* k *of* $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$ *we can construct a mobile strict-star factoring* $\mathcal{F}$ *with* k *leaves, and vice versa.*

Proof. From left to right, let $I = \{L_1, \dots, L_k\}$ be an independent set of size k in $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$. As $\mathbf{ES}^*_\Pi \subseteq \mathcal{P}(\mathcal{V}) \setminus \{\emptyset, \mathcal{V}\}$ we know from Lemma 6 that $\mathcal{F} = \{L_1, \dots, L_k, C\}$, with $C = \mathcal{V} \setminus L_1 \cup \ldots \cup L_k$, forms a strict-star factoring with k leaves. By construction, every leaf L_i is mobile.

From right to left, let $\mathcal{F}$ be a mobile strict-star factoring with center C and k leaves $\mathcal{L} = \{L_1, \dots, L_k\}$. By definition, each mobile leaf is affected by at least one leaf-only action. We refer to the leaf-only action that affects L_i as a_i. We next show that $\mathcal{F}' = \mathcal{L}' \cup \{C'\}$ is a strict-star factoring with leaves $\mathcal{L}' = \{\mathsf{vars}(\mathsf{eff}(a_1)), \dots, \mathsf{vars}(\mathsf{eff}(a_k))\}$ and center $C' = \mathcal{V} \setminus \bigcup_{L \in \mathcal{L}'} L$. Assume for contradiction that there exist $L'_i, L'_j \in \mathcal{L}'$ and $v_i \in L'_i$ and $v_j \in L'_j$, such that (v_i, v_j) is an arc in CG_Π. Since $L'_i \subseteq L_i$ and $L'_j \subseteq L_j$, this would imply a causal-graph connection between L_i and L_j in contradiction to the assumption that $\mathcal{F}$ forms a strict-star factoring. Thus, $\mathcal{F}'$ is a strict-star factoring. Since $\mathbf{ES}^*_\Pi$ contains all effect schemas, so in particular all $\mathsf{vars}(\mathsf{eff}(a_i))$, it follows from Lemma 6 that $\{\mathsf{vars}(\mathsf{eff}(a_1)), \dots, \mathsf{vars}(\mathsf{eff}(a_k))\}$ is an independent set in $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$. □

With this result, we have an algorithm that computes a strict-star factoring $\mathcal{F}$ with the maximum number of mobile leaves from a maximum independent set of $\mathbf{PSLG}_\Pi(\mathbf{ES}^*_\Pi)$.

5.4.2 General-Star Factorings

All factoring strategies that we introduced so far produce only strict-star factorings, but none of the methods is able to identify general-star factorings. In this section, we introduce the *potential general-leaf hypergraph*, based on which we can compute general-star factorings.

Definition 26 (Potential General-Leaf Hypergraph)**.** *Let* Π *be a planning task and* CG_Π *its causal graph. The* potential general-leaf hypergraph $\mathbf{PGLG}_\Pi(\mathcal{S})$ *is a hypergraph with vertices* $\mathcal{S} \subseteq \mathcal{P}(\mathcal{V}) \setminus \{\emptyset, \mathcal{V}\}$*, and edges (i)* $\{S, S'\}$ *if* $S \cap S' \neq \emptyset$*, (ii)* $\{S, S'\}$ *if there exist* $v \in S$*, and* $a \in \mathcal{A}$*, such that* $v \in \mathsf{vars}(\mathsf{pre}(a))$*, and* $\mathsf{vars}(\mathsf{eff}(a)) \subseteq S'$*, and*

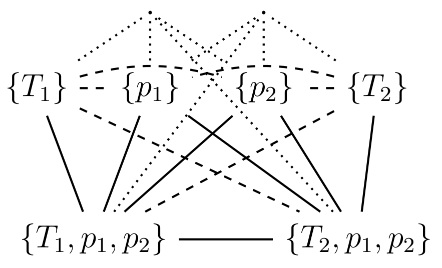


Figure 5.5: The potential general-leaf hypergraph of the example.

(iii) $\{S_1, \dots, S_n\}$ *if* $n > 1$ *and there exists an action* $a \in \mathcal{A}$ *such that* $\mathsf{vars}(\mathsf{eff}(a)) \subseteq S_1 \cup \dots \cup S_n$ *and* $\forall S_i : \mathsf{vars}(\mathsf{eff}(a)) \cap S_i \neq \emptyset \wedge \mathsf{vars}(\mathsf{eff}(a)) \not\subseteq S_i$.

The potential general-leaf hypergraph $\mathbf{PGLG}_\Pi(\mathcal{S})$ has an edge between (i) every pair of overlapping variable sets as in the strict-star case. Type (ii) edges prevent precondition-effect dependencies between two leaves caused by leaf-only actions $a \in \mathcal{A}^\mathcal{L} \setminus \mathcal{A}^C$. Condition (iii) ensures that no action $a \in \mathcal{A}$ in the obtained factoring affects more than one leaf without affecting the center at the same time. At least one of $S_1, \dots, S_n$ cannot become a leaf, since otherwise a affects only leaf factors. Note that the last part of the condition ($\mathsf{vars}(\mathsf{eff}(a)) \not\subseteq S_i$) only prevents hyperedges between sets that are connected by type (i) edges, anyway.

Example 9. *Figure 5.5 shows the potential general-leaf hypergraph* $\mathbf{PGLG}_\Pi(\mathbf{ES}^*_\Pi)$ *of our example, where, like in Figure 5.4, type-(ii)-only edges are dashed, and all solid connections result from both type (i) and type (ii) edges in the graph. The difference here is that there is no type (ii) edge between the packages, because, although there are causal-graph edges between them (caused by the loadAllDrive*$()$ *actions), condition (ii) of the definition is not satisfied. Additionally, there are* 10 *type (iii) hyperedges that result from the loadAllDrive*$()$ *actions. Figure 5.5 only shows two of them (dotted), namely* $E^0_x = \{\{T_x\}, \{p_1\}, \{p_2\}, \{T_y, p_1, p_2\}\}$, *where* $x \neq y$. *From each* E^0_x, *there result* 4 *additional hyperedges, namely all subsets of* E^0_x *that cover all variables of the effect schema of the loadAllDrive*$()$ *actions of truck* t_x. *These are:* $E^1_x = \{\{T_x\}, \{T_y, p_1, p_2\}\}$, $E^2_x = \{\{T_x\}, \{p_1\}, \{p_2\}\}$, $E^3_x = \{\{T_x\}, \{p_2\}, \{T_y, p_1, p_2\}\}$, *and* $E^4_x = \{\{T_x\}, \{p_1\}, \{T_y, p_1, p_2\}\}$. *We omitted these in the example for readability.*

There are two maximum independent sets of the hypergraph, again of size 2, *namely* $\{\{T_1\}, \{T_2\}\}$ *and* $\{\{p_1\}, \{p_2\}\}$. *The latter forms a general-star, though not a strict-star, factoring because the loadAllDrive*$()$ *actions affect one of the trucks, thus are center actions for which there is no restriction in general-star factorings.*

As for strict-star factorings, we can prove that from an independent set of size k of $\mathbf{PGLG}_\Pi(\mathbf{ES}^*_\Pi)$ we can construct a mobile general-star factoring $\mathcal{F}$ with k leaves. The proof idea is similar to that of Theorem 12.

Lemma 7 (General-Star Factoring from Independent Set)**.** *Let* Π *be a planning task and let* $\mathcal{S}=\mathcal{P}(\mathcal{V}) \setminus \{\emptyset, \mathcal{V}\}$*. Then* $\mathcal{F}=\{L_1, \dots, L_k, C\}$ *is a general-star factoring for* Π*, iff* $I = \{L_1, \dots, L_k\}$ *forms an independent set of* $\mathbf{PGLG}_\Pi(\mathcal{S})$*, where* $C = \mathcal{V} \setminus \bigcup_{i=1}^{k} L_i$*.*

Proof. From left to right, let $\mathcal{F} = \{L_1, \dots, L_k, C\}$, with $L_i \in \mathcal{S}$, be a general-star factoring. We prove that $I = \{L_1, \dots, L_k\}$ forms an independent set in $\mathbf{PGLG}_\Pi(\mathcal{S})$. Assume for contradiction that there exists $\{L_{i_1}, \dots, L_{i_n}\} \subseteq I$ such that $\{L_{i_1}, \dots, L_{i_n}\}$ forms a hyperedge in $\mathbf{PGLG}_\Pi(\mathcal{S})$. According to the definition of $\mathbf{PGLG}_\Pi(\mathcal{S})$, such a hyperedge exists for one of three reasons denoted by (i), (ii), and (iii). Reason (i) contradicts the assumption that $\{L_1, \dots, L_k, C\}$ forms a valid factoring, i. e., a partitioning of $\mathcal{V}$. Reason (ii) is in contradiction to $\mathcal{F}$ forming a general-star factoring, as it would imply that there exists an action $a \in \mathcal{A}$ with $\mathsf{vars}(\mathsf{eff}(a)) \cap C = \emptyset$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq F$ but $\mathsf{vars}(\mathsf{pre}(a)) \not\subseteq F \cup C$. Reason (iii) also contradicts the fact that $\mathcal{F}$ forms a general star factoring, more precisely, that for each action $a \in \mathcal{A}$ with $\mathsf{vars}(\mathsf{eff}(a)) \cap C = \emptyset$ it holds that there exists $F \in \mathcal{F}$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq F$.

From right to left, let $I = \{L_1, \dots, L_k\}$ be an independent set in $\mathbf{PGLG}_\Pi(\mathcal{S})$. Note that $\mathcal{F} = \{L_1, \dots, L_k, C\}$ with $C = \mathcal{V} \setminus L_1 \cup \ldots \cup L_k$ is a valid factoring, i. e., a partitioning of $\mathcal{V}$, for the same reason as in the proof of Lemma 6.

We next prove that $\mathcal{F}$ forms a general-star factoring. Let $a \in \mathcal{A}$ be an action such that $\mathsf{vars}(\mathsf{eff}(a)) \cap C = \emptyset$. We must show that there exists an $F \in \mathcal{F}$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq F$ and $\mathsf{vars}(\mathsf{pre}(a)) \subseteq F \cup C$. Assume for contradiction that there exists no $F \in \mathcal{F}$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq F$. From $\mathsf{vars}(\mathsf{eff}(a)) \cap C = \emptyset$ it follows that there exists $\{L'_1, \dots, L'_n\}$ with $n > 1$, $L'_i \neq C$, and $L'_i \cap \mathsf{vars}(\mathsf{eff}(a)) \neq \emptyset$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L'_1 \cup \dots \cup L'_n$. This is in contradiction to (iii) in the definition of $\mathbf{PGLG}_\Pi(\mathcal{S})$, which introduces a hyperedge between such L'_i. Consequently, there exists an $F \in \mathcal{F}$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq F$. Further, we need to show that $\mathsf{vars}(\mathsf{pre}(a)) \subseteq F \cup C$. Assume for contradiction that there exists a $v \in \mathsf{vars}(\mathsf{pre}(a))$ such that $v \in F'$ where $F' \neq F$ and $F' \neq C$. This contradicts part (ii) in the definition of $\mathbf{PGLG}_\Pi(\mathcal{S})$, which introduces an edge in $\mathbf{PGLG}_\Pi(\mathcal{S})$ between such F and F'. □

Similar to before, we are now ready to proof that we can construct a general-star factoring with the maximum number of mobile leaves from a maximum independent set of the potential general-leaf hypergraph:

Theorem 13. *Let* Π *be a planning task and let* $\mathbf{ES}^*_\Pi$ *be the set of its potential leaves. Then from an independent set of size* k *of* $\mathbf{PGLG}_\Pi(\mathbf{ES}^*_\Pi)$ *we can construct a mobile general-star factoring* $\mathcal{F}$ *with* k *leaves, and vice versa.*

Proof. From left to right, let $I = \{L_1, \dots, L_k\}$ be an independent set of size k in $\mathbf{PGLG}_\Pi(\mathbf{ES}^*_\Pi)$. As $\mathbf{ES}^*_\Pi \subseteq \mathcal{P}(\mathcal{V}) \setminus \{\emptyset, \mathcal{V}\}$ it follows directly from Lemma 7 that $\mathcal{F} = \{L_1, \dots, L_k, C\}$, with $C = \mathcal{V} \setminus L_1 \cup \dots \cup L_k$, forms a general-star factoring with k leaves. By construction, every leaf L_i is mobile.

From right to left, let $\mathcal{F}$ be a mobile general-star factoring with k leaves $\mathcal{L} = \{L_1, \dots, L_k\}$ and center factor C. By definition, each mobile leaf is affected by at least one leaf-only action. We refer to the leaf-only action that affects L_i as a_i. We next show that $\mathcal{F}' = \mathcal{L}' \cup \{C'\}$ is a general-star factoring with leaves $\mathcal{L}' = \{\mathsf{vars}(\mathsf{eff}(a_1)), \dots, \mathsf{vars}(\mathsf{eff}(a_k))\}$ and center $C' = \mathcal{V} \setminus \bigcup_{L \in \mathcal{L}'} L$. Let $a \in \mathcal{A}$ be an action such that $C' \cap \mathsf{vars}(\mathsf{eff}(a)) = \emptyset$. We show that there exists $L'_i \in \mathcal{L}'$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L'_i$ and $\mathsf{vars}(\mathsf{pre}(a)) \subseteq L'_i \cup C'$. Assume for contradiction that no $L'_i \in \mathcal{L}'$ exists such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L'_i$. From $\mathsf{vars}(\mathsf{eff}(a)) \cap C' = \emptyset$ it follows that there exist at least two leaf factors $L'_n = \mathsf{vars}(\mathsf{eff}(a_n))$ and $L'_m = \mathsf{vars}(\mathsf{eff}(a_m))$ such that $L'_n \cap \mathsf{vars}(\mathsf{eff}(a)) \neq \emptyset$ and $L'_m \cap \mathsf{vars}(\mathsf{eff}(a)) \neq \emptyset$. From $L'_n \subseteq L_n$ and $L'_m \subseteq L_m$ it follows that $L_n \cap \mathsf{vars}(\mathsf{eff}(a)) \neq \emptyset$ and $L_m \cap \mathsf{vars}(\mathsf{eff}(a)) \neq \emptyset$. From $C \subseteq C'$, it follows that also $C \cap \mathsf{vars}(\mathsf{eff}(a)) = \emptyset$, in contradiction to $\mathcal{F}$ forming a general-star factoring. Thus, there exists a leaf factor $L'_i \in \mathcal{L}'$ such that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L'_i$. Further, we prove that $\mathsf{vars}(\mathsf{pre}(a)) \subseteq L'_i \cup C'$. Assume for contradiction that there exists a $v \in \mathsf{vars}(\mathsf{pre}(a))$ such that $v \notin L'_i \cup C'$, i. e., there exists $L'_j = \mathsf{vars}(\mathsf{eff}(a_j))$ with $L'_i \neq L'_j$ and $v \in L'_j$. From $L'_i \subseteq L_i$ and $L'_j \subseteq L_j$ it follows that $\mathsf{vars}(\mathsf{eff}(a)) \subseteq L_i$ and $v \in L_j$. As before, since $C \subseteq C'$ and therefore $C \cap \mathsf{vars}(\mathsf{eff}(a)) = \emptyset$ this is in contradiction to $\mathcal{F}$ forming a general-star factoring. Thus, $\mathcal{F}'$ forms a general-star factoring. As $\mathsf{vars}(\mathsf{eff}(a_i)) \in \mathbf{ES}^*_\Pi$, it follows from Lemma 7 that $\{\mathsf{vars}(\mathsf{eff}(a_1)), \dots, \mathsf{vars}(\mathsf{eff}(a_k))\}$ is an independent set in $\mathbf{PGLG}_\Pi(\mathbf{ES}^*_\Pi)$. □

This result provides us with the first approach to compute general-star factorings, the most-general types of factorings. As we will see in Chapter 7.3, there are many benchmark domains in which general-star factorings can be identified. This is, however, not always beneficial for the search, because the additional dependencies between the leaves reduce the potential reduction of decoupled search.

5.4.3 Objective Function

Concluding this chapter, we briefly describe the ILP encoding and the objective function. We construct ILPs that compute independent sets on the aforementioned graphs. For each potential leaf $L \in \mathbf{ES}^*_\Pi$, we have a binary variable x_L that encodes if L is a leaf ($x_L = 1$) or not. We add a constraint for every (hyper-)edge in the potential-leaf graphs $\mathcal{G}$:

$$x_{L_1} + \dots + x_{L_k} \leq k - 1 \quad \text{for } \{L_1, \dots, L_k\} \in \mathcal{G}$$

It is easy to see that the solutions to this ILP directly correspond to star factorings $\mathcal{F}$ where each potential leaf L where $x_L = 1$ becomes a leaf in $\mathcal{F}$. The center consists of the remaining, non-leaf, variables.

For every potential leaf, we specify an *objective value* o_L, which is 1 if we aim at maximizing the number of mobile leaves, or the mobility $|\mathcal{A}^L_{\not{C}}|$ of the leaf if that is the property we want to optimize. Then $\sum_{L \in \mathbf{ES}^*_\Pi} o_L \cdot x_L$ is the objective function of the ILP, maximizing the sum of the respective measures in the factoring.

We remark that, when maximizing the number of mobile leaves, an optimal ILP solution indeed results in the factoring that globally maximizes that number. In contrast, when maximizing mobility, we do not obtain the factoring with maximum mobility, because we do only consider leaves that result from a single effect schema. To obtain the global maximum, we need to consider also all unions of effect schemas. We did some preliminary experiments with this, but experienced that even considering only all pairs of schemas leads to a prohibitive runtime overhead.

Chapter 6

Related Work – Exponential Separations

Decoupled search relates to several existing techniques that attempt to tackle the state explosion problem by pruning the state space or representing it in a compact way. In this chapter, we will compare decoupled search to the most popular such methods, namely partial-order reduction via stubborn sets, Petri-net unfolding, symmetry breaking, and symbolic representations using binary decision diagrams (BDDs). We also show the relation to existing works in factored planning, and discuss the connection to other more remotely related techniques.

We put an emphasis on the relation to Petri-net unfolding and stubborn sets pruning, which exploit the independence of components, and actions affecting these, similar to decoupled search. For both, we present exponential separations showing that the reduction achieved by these methods is in general incomparable to the reduction of decoupled search. For unfolding, we additionally prove that under certain conditions—single-variable factors, absence of prevail conditions, and compatible search orders—the state-space representations are polynomially related. We also look more closely into the relation to symbolic representations via BDDs, which, like decoupled search, use a different state representation and search over compactly represented sets of states.

Previous works have introduced several forms of factored planning in the past, where the search considers partial states with support for different types of dependencies across factors. We show how decoupled search relates to these techniques.

Symmetry breaking is not that closely related to decoupled search, being based on symmetric components, not component independence. Consequently, the reduction achieved is naturally orthogonal to decoupled search.

This chapter is mostly based on Gnad and Hoffmann [2018]. The formal comparison to unfolding was introduced in Gnad and Hoffmann [2019], the exponential separations to stubborn set pruning were introduced in Gnad et al. [2019a]. The idea behind the proof of Theorem 22 is due to Álvaro Torralba.

6.1 Petri-Net Unfolding

Petri net unfolding is a well-known partial-order reduction method (e. g. McMillan, 1992; Esparza et al., 2002; Baldan et al., 2012). Instead of building the forward state space and trying to prune permutative parts as in other methods (e. g. Valmari, 1989; Godefroid and Wolper, 1991; Wehrle and Helmert, 2012), the state variables are not multiplied with each other in the first place. The unfolding process incrementally adds transitions to an acyclic graph, when the transition's input "places" (precondition facts) can be reached jointly. A new output place is then added for each effect. The outcome structure is an acyclic Petri net, a *complete prefix*, that preserves reachability exactly relative to the input Petri net.

Planning tasks can be translated into Petri nets [Hickmott et al., 2007; Bonet et al., 2008], where each place in an unfolding then corresponds to a state-variable value and net transitions correspond to actions. The complete prefix is an acyclic fact-action dependency structure that captures reachability.

Unfolding and decoupled search are related. Consider a *singleton-component* factoring, where each factor contains a single state variable. The component states then are exactly the places in the unfolding, and, like unfolding, decoupled search expands these separately. So how do the techniques relate exactly?

A significant known separation is the complexity of deciding whether a conjunctive condition is reachable. Given an unfolding prefix, this test is **NP**-complete as the transition histories supporting two places may be in conflict [McMillan, 1992]. In contrast, given a decoupled state space, the test can be done in time linear in the number of reached decoupled states: thanks to the star-topology organization, there are no conflicts. But what about the sizes of the fully expanded prefixes, i. e., the respective complete representation of reachability?

We will show that the reachable decoupled state space can be exponentially smaller in the presence of prevail conditions (non-deleted preconditions, whose treatment in Petri nets leads to blow-ups); and that the complete unfolding prefix can be exponentially smaller in the presence of non-singleton factors. We also show corresponding dominance results: without prevail conditions, unfolding size dominates the size of the reachable decoupled state space; with only singleton components, the size of the reachable decoupled state space dominates unfolding size. Both hold subject to *compatible* search orders, that prefer expansions on leaves over ones on the center whenever possible, and that are identical on the ordering of center-action sequences. For incompatible search orders, exponential advantages are possible in either direction. Overall, we obtain a complete classification along the three dimensions of prevail conditions (yes/no), non-singleton components (yes/no), and incompatible search orderings (yes/no).

In this chapter, we only state our results in the form of separation and domination theorems. The full proofs as well as additionally required backgrounds covering unfoldings are provided in Appendix A.3.

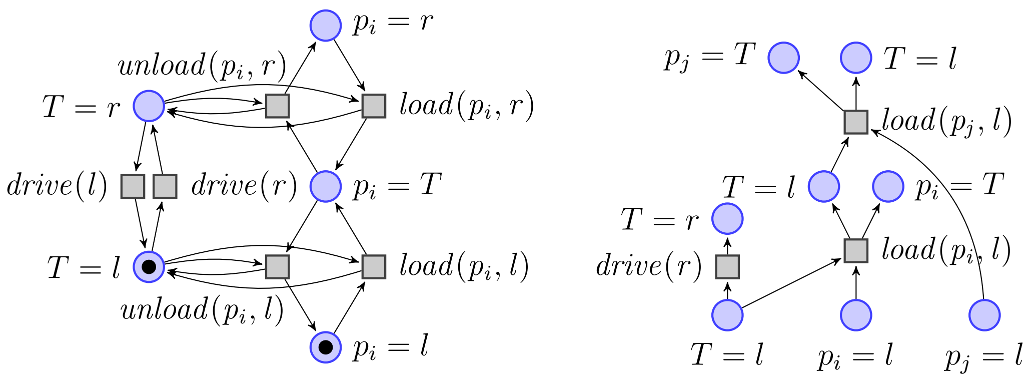


Figure 6.1: Part of the petri-net encoding of our running example (left), and an incomplete prefix of its unfolding (right).

6.1.1 Background – Petri-Net Unfolding

Planning tasks can be encoded in *Petri nets* $\Sigma = \langle P, T, F, M_0 \rangle$, which are digraphs whose nodes are the *places* P and *transitions* T of Σ. When encoding a planning task Π, the places P correspond to the facts of Π, and the transitions T correspond to the actions $\mathcal{A}$ [Hickmott et al., 2007; Bonet et al., 2008]. The *flow relation* F connects precondition places as input to transitions, and effects as their outcome, e. g., $(p, t) \in F$ means that t has precondition p. A state s of Π (a set of facts) becomes a *marking* M in Σ (a set of places). $M_0 = \mathcal{I}$ is the initial marking. A transition t can *fire* (i. e., is applicable) in a marking M if $\mathsf{pre}(t) \subseteq M$. The resulting marking is $M' = (M \setminus \mathsf{pre}(t)) \cup \mathsf{eff}(t)$.[1]

We will use a scaling variant of our logistics example with a single truck T, n packages p_i, and two locations l, r. The truck can drive left, $drive(l)$, and right, $drive(r)$, and packages can be (un)loaded, $(un)load(p_i, x)$ for $x \in \{l, r\}$, as usual. Initially, truck and packages are at l. Figure 6.1 (left) illustrates part of the Petri net encoding of the task, showing only one package. Places are blue circles, transitions gray boxes. A marking M places a token in every place $p \in M$.

Petri nets do not natively support prevail conditions—variables preconditioned, but not affected by an action. The input places of transitions are “consumed” when a transition fires. Prevail conditions can be encoded by re-adding a token to the respective place. In our example, the encoding of (un)load actions consumes the current truck position, and adds it back in the effect. This incurs a blow-up in the unfolding, as a distinction is introduced between the truck position before vs. after such actions.

The outcome of the unfolding process for a Petri net Σ is a triple $Unf_\Sigma = \langle B, E, G \rangle$ that captures all markings reachable from M_0 in Σ. The *conditions* $b \in B$ and *events* $e \in E$ of Unf_Σ are labeled with the places $p \in P$ and transitions $t \in T$ in Σ. G extends

[1]We assume that there are no actions with effect-only variables v, i. e., $v \in \mathsf{vars}(\mathsf{eff}(a))$ but $v \notin \mathsf{vars}(\mathsf{pre}(a))$. Such actions can cause an exponential blow-up in the net itself, as mentioned in Chapter 7.2.

the flow relation F according to these labels. The unfolding process ensures that G is acyclic. A *configuration* is a set $C \subseteq E$ of events, partially ordered by G, that includes all its predecessor events and that can be sequenced to an executable transition sequence. Working on singleton conditions (facts), the unfolding does not enumerate permutations of concurrent events, i. e., events whose effects are defined on disjoint variable sets.

The unfolding is generated as follows. Initially, for every $p \in M_0$, there is a *condition* b in the unfolding. Then, the unfolding incrementally extends Unf_Σ by appending possible events e, adding a new condition to Unf_Σ for every $b \in \mathsf{eff}(e)$. An event e can fire at a configuration C if the outcome $C \cup \{e\}$ is again a configuration (i. e., can still yield an executable transition sequence), and e is not a *cut-off* event. An event e is *cut-off* if its marking M, obtained when executing the configuration supporting e, equals the marking M' of an already generated configuration. The unfolding terminates if no more events can fire; it then represents all markings reachable in Σ. We define the size of an unfolding $|Unf_\Sigma| := |B|$ as the number of its conditions. We assume a *search order* $\ll$, an order over configurations, constraining the firing order: at each point in the unfolding process, the $\ll$-minimal possible event e is added.

Figure 6.1 (right) illustrates some unfolding steps. Observe the blow-up due to missing support for prevail conditions: any load event can choose to either use the initial occurrence of $(T = l)$, or any other occurrence generated by previous load events, so that the number of possibilities multiplies over the number of load events and thus packages.

We denote the Petri net encoding a planning task Π by $\Sigma(\Pi)$, and the corresponding unfolding by Unf_Π.

6.1.2 Results Overview

Our results are exponential separations and domination properties, between decoupled search and unfolding. Here we give an overview and state the theorems and proof intuitions afterwards. We show that, using compatible search orders, unfolding dominates decoupled search on planning tasks without prevail conditions, and decoupled search dominates unfolding with singleton-component factorings. We consider the following three dimensions:

(i) Presence or absence of prevail conditions.

(ii) Presence or absence of multi-variable components.

(iii) Compatibility, or lack thereof, of the search orders $\ll$.

Dimension (i) concerns the planning tasks Π. We denote the class of task Π *without prevail conditions* by “-P”, and the class of all (arbitrary) task Π, not making that restriction, by “+P”. For dimension (ii), we denote by -M the restriction where the factoring $\mathcal{F}$

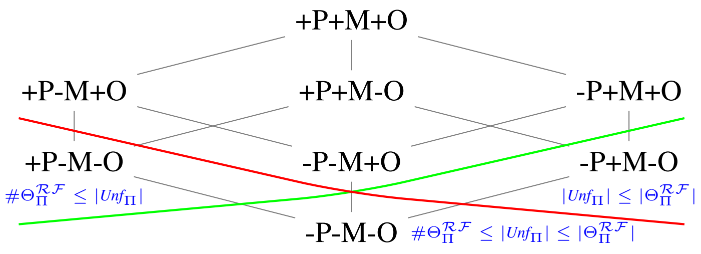

+P+M+O
+P-M+O
+P+M-O
-P+M+O
+P-M-O
-P-M+O
-P+M-O
$\#\Theta^{\mathcal{RF}}_{\Pi} \leq |\mathit{Unf}_{\Pi}|$
$|\mathit{Unf}_{\Pi}| \leq |\Theta^{\mathcal{RF}}_{\Pi}|$
-P-M-O
$\#\Theta^{\mathcal{RF}}_{\Pi} \leq |\mathit{Unf}_{\Pi}| \leq |\Theta^{\mathcal{RF}}_{\Pi}|$

Figure 6.2: Subsumption hierarchy and results overview. Above the green line, decoupled search can yield exponentially smaller representation size, below that line it cannot. Above the red line, unfolding can yield exponentially smaller representation size, below it cannot.

may not contain multi-variable components, and by +M the class of all $\mathcal{F}$ not imposing this limitation.

Regarding dimension (iii), note that we are not interested in heuristic search here; the target is to build a complete representation of reachability (the reachable decoupled state space, a complete prefix in unfolding). Still, the order of expansions can significantly impact representation size, potentially incurring or avoiding exponential blow-ups as we shall see. We capture this in terms of the search orders $\ll$. We say that a pair of search orders $(\ll_U, \ll_D)$ is *compatible* if *(O1)* $\ll_U$ always orders new leaf events before new center events; and *(O2)* $\ll_D$ and $\ll_U$ agree on center paths: a $\ll$-minimal event in the unfolding corresponds to $\ll_D$-minimal center paths adding the corresponding center action to the decoupled state space. In other words, the only degree of freedom in $\ll_U$, over $\ll_D$, is the relative ordering of leaf events. We denote that restriction by -O, and the unrestricted case by +O. Note that (O1) mimics the factor-state generation order in decoupled search, where after adding a center action, all reachable leaf states are added prior to considering the next center action. Formal definitions of (O1) and (O2) are given in Appendix A.3.1.

For comparing the size of the state-space representations, we define the size of the reachable decoupled state space $\Theta^{\mathcal{RF}}_{\Pi}$ as the number of facts reached in the decoupled states $|\Theta^{\mathcal{RF}}_{\Pi}| := \sum_{s^{\mathcal{F}} \in \Theta^{\mathcal{RF}}_{\Pi}} (|C| + \sum_{s^L \in S^{\mathcal{L}} \wedge \mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty} |s^L|)$. We denote the number of decoupled states in $\Theta^{\mathcal{RF}}_{\Pi}$ by $\#\Theta^{\mathcal{RF}}_{\Pi}$.

Figure 6.2 gives an overview of the hierarchy of sub-classes induced by dimensions (i) – (iii), and the associated reachability representation size results. In this hierarchy, exponential separations are inherited upwards, to more permissive classes, as separating example families get preserved; while domination properties are inherited downwards, to more restricted classes, as the required prerequisites are preserved.

The next section shows our separation theorems. We show that for the class +P-M-O there exist planning task families where decoupled search results in exponentially smaller reachability representations. We show that, within -P+M-O, unfolding size can be exponentially smaller. For incompatible orders -P-M+O, we show separations in both directions.

Afterwards, we show our domination theorems. Within +P-M-O, the number of decoupled states is always at most as large as the unfolding, $\#\Theta_{\Pi}^{\mathcal{RF}} \leq |\mathit{Unf}_{\Pi}|$. On the other hand, within -P+M-O, the unfolding is at most as large as the decoupled state space, $|\mathit{Unf}_{\Pi}| \leq |\Theta_{\Pi}^{\mathcal{RF}}|$. As $\#\Theta_{\Pi}^{\mathcal{RF}}$ and $|\Theta_{\Pi}^{\mathcal{RF}}|$ are polynomially related given -M, with downward inheritance in particular we get that, in the most restricted class -P-M-O, decoupled state-space size and unfolding size are polynomially related.

6.1.3 Separation Theorems

We show exponential separations between decoupled search and unfolding. The planning task families Π^n in the following theorems have size linear in n.

Theorem 14. *There exists a family of tasks Π^n in +P, with factorings in -M and search orders in -O, where $|\Theta_{\Pi^n}^{\mathcal{RF}}|$ is polynomial in n while $|\mathit{Unf}_{\Pi^n}|$ is exponential in n.*

Our example is such a family Π^n. There are only 3 decoupled states, independently of n; the number of factor states is linear in n. The number of conditions in the unfolding is exponential in n, because all possible combinations of, e. g., $\mathit{load}(p_i, l)$ actions are enumerated in the initial state.

The so-called *place-replication* method in Petri nets encodes prevail conditions differently, with copies of the prevail places [Baldan et al., 2012]. In our example, though, this incurs the same blow-up. Contextual Petri nets [Baldan et al., 2012] have built-in support for prevail conditions ("read arcs"), yet contextual unfoldings must keep track of "event histories" which again incur the same blow-up.

Theorem 15. *There exists a family of tasks Π^n in -P, with factorings in +M and search orders in -O, where $\#\Theta_{\Pi^n}^{\mathcal{RF}}$ is exponential in n while $|\mathit{Unf}_{\Pi^n}|$ is polynomial in n.*

Such example families can be constructed through permutability (*concurrency*, in Petri net parlance) within factors. For example, scaling the number of trucks in logistics, if all truck variables are in the center, then decoupled search enumerates all possible interleavings of truck drives. Unfolding expands the trucks separately, avoiding that blow-up.

Theorem 16. *There exists a family of tasks Π^n in -P, with factorings in -M and search orders in +O, where $|\Theta_{\Pi^n}^{\mathcal{RF}}|$ is polynomial in n while $|\mathit{Unf}_{\Pi^n}|$ is exponential in n.*

Theorem 17. *There exists a family of tasks Π^n in -P, with factorings in -M and search orders in +O, where $\#\Theta_{\Pi^n}^{\mathcal{RF}}$ is exponential in n while $|\mathit{Unf}_{\Pi^n}|$ is polynomial in n.*

For both theorems, we construct example families and search orders where one technique enters a part of the search space that is exponential in n, while the other technique takes a "short-cut", entering a part of the search space that allows to capture complete reachability with polynomial representation size. For Theorem 16, the task family is constructed so that the unfolding search has a detrimental priority to expand center actions—even though leaf actions could be expanded, violating (O1)—missing the "short-cut" offered by leaf actions after a single center action has been applied. For Theorem 17, vice versa, complying with (O1) may lead to an exponential disadvantage, due to generating the leaf preconditions of center actions causing a blow-up.

6.1.4 Domination Theorems

We now show domination results between the size of the decoupled state space and that of the unfolding. With a singleton center factor C, there is no concurrency within C, so unfolding can only exploit the concurrency inherent in the factoring and within leaves. Formally:

Theorem 18. *For* Π *in +P, factorings with singleton center factor (-M), and search orders in -O, with hypercube pruning* $\#\Theta_{\Pi}^{\mathcal{RF}} \leq |\mathit{Unf}_{\Pi}|$.

The proof consists of Lemmas 22 and 23 in Appendix A.3.2. Lemma 22 shows that, with a singleton center factor C, no events that affect C can be concurrent. In Lemma 23, we show that decoupled states $s^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{RF}}$ that are not pruned by hypercube pruning can be injectively mapped to non-cut-off events in Unf_{Π}. Say action occurrence a in $s^{\mathcal{F}}$ generates a state not contained in any previous hypercube, and say a is mapped to e. Then e is not a cut-off: with compatibility of $\ll$, the only additional conditions generated in Unf_{Π} are duplicates of prevail conditions, and the only additional events are duplicates of actions consuming these conditions. With Lemma 22, conditions in Unf_{Π} cannot be combined across decoupled states, because their center sub-configurations are not concurrent.

If additionally all leaf factors are singleton, too, there is no blow-up relative to unfolding incurred there, either, and the number of factor state occurrences in $\Theta_{\Pi}^{\mathcal{RF}}$ is at most the number of conditions in Unf_{Π}.

Our next result is perhaps more surprising. The exponential advantage of decoupled search disappears without prevail conditions:

Theorem 19. *For* Π *in -P, factorings in +M, and search orders in -O,* $|\mathit{Unf}_{\Pi}| \leq |\Theta_{\Pi}^{\mathcal{RF}}|$.

The proof consists of two parts. The first part considers the non-pruned versions of the decoupled state space and the unfolding prefix, and shows that the factor-state occurrences in $\Theta_{\Pi}^{\mathcal{RF}}$ can be surjectively mapped to corresponding *factor co-sets* in Unf_{Π}: jointly reachable conditions over the variables of a factor. During the construction of

Unf_Π and $\Theta_\Pi^{\mathcal{RF}}$, for every new event e in Unf_Π, every new factor co-set supported by e is matched by corresponding new factor states in $\Theta_\Pi^{\mathcal{RF}}$. The crucial part of the argument is that, in the absence of prevail conditions, all new conditions b generated by e correspond to a factor-state change in the planning task, matched by the generation of a new factor state in $\Theta_\Pi^{\mathcal{RF}}$.

The second part of the proof observes that, for any event e, corresponding new factor state occurrences in $\Theta_\Pi^{\mathcal{RF}}$ map to factor co-sets including the new conditions added by e. The factor co-sets mapped to are different for every event. Further, at any point in the construction, with compatibility of $\lll$, the current $\Theta_\Pi^{\mathcal{RF}}$ prefix cannot represent states not represented in the Unf_Π prefix (Lemma 24). Thus, if e is a non-cut-off event, at least one decoupled state $s^{\mathcal{F}}$ containing the new factor-state occurrences is not pruned by hypercube pruning.

Corollary 1. *For* Π *in -P, factorings in -M, and search orders in -O, with hypercube pruning* $\#\Theta_\Pi^{\mathcal{RF}} \leq |\mathit{Unf}_\Pi| \leq |\Theta_\Pi^{\mathcal{RF}}|$.

Our results completely characterize the possibility of exponential size differences, or lack thereof, between decoupled search and unfolding as a function of three major dimensions. The search organization in decoupled search, exploiting the star topology, yields advantages over unfolding not only in terms of the complexity of deciding reachability, but also in terms of state-space size when there are prevail conditions. The latter is the only source of size advantages, given compatible search orders. The only source of size advantages for unfolding are non-singleton components.

6.2 Partial-Order Reduction – Stubborn Sets

Partial-order reduction (POR) methods prune applicable transitions, preserving completeness (and optimality) while avoiding the exploration of unnecessary action permutations. POR originates in formal verification (e. g. Valmari, 1989; Godefroid and Wolper, 1991; Valmari, 1992; Peled, 1993; Godefroid, 1996; Edelkamp et al., 2004a), and has also been successfully applied in planning, specifically in the form of *stubborn sets* [Wehrle et al., 2013; Wehrle and Helmert, 2014; Winterer et al., 2017].

Some POR variants, e. g., *sleep sets* [Godefroid and Wolper, 1991] and their derivatives, reduce only the number of transitions in the reachable state space. Decoupled search is trivially exponentially separated from transition-pruning methods, since these do not affect the set of states reached during search.

Stubborn sets [Valmari, 1989], on the other hand, reduce the number of reachable states. Herein, we will formally compare decoupled search to three of its variants introduced in planning, namely *strong stubborn sets*, *weak stubborn sets*, and *compliant stubborn sets* [Sievers and Wehrle, 2021]. We distinguish these three variants since there is no dominance of the pruning power between them.

Applying stubborn sets pruning in search, a set of actions T—the stubborn set—is identified for each state s, such that pruning all actions applicable in s that are not in T preserves completeness and optimality of the underlying search algorithm. The computation of stubborn sets relies crucially on the notions of *interference* of actions and *necessary enabling sets*. The latter capture that progress needs to be made towards the goal, interference ensures that actions not in T are not missed that might be needed later on, but that might not be applicable behind s any more.

Interference, or lack thereof, of actions hints at a type of conditional independence, similar to what decoupled search exploits. If there are two actions applicable in the same state that are not interfering, stubborn sets will capture this by only including one of them, thereby reducing the search space.

Decoupled search and stubborn sets are complementary in that each is exponentially separated from the other. So although both make use of a similar structure of a planning task, this is exploited in different ways. We next show results of the exponential separation, giving brief intuitions here only. The full proofs are given in Appendix A.4.

Theorem 20. *Decoupled search is exponentially separated from all three variants of stubborn sets.*

The scaling example detailed in the proof is similar to our standard logistics task, but has an additional action that interferes with all load actions. This causes all variants of stubborn sets to include all load actions, so the search visits, e. g., all 2^n states where any subset of the packages are loaded into the truck, vs. still in the initial location. The decoupled state space, though, has only three reachable states and the additional action only introduces transitions that go back from each of these states to $\mathcal{I}^{\mathcal{F}}$.

Theorem 21. *All three variants of stubborn sets are exponentially separated from decoupled search.*

The opposite claim follows from an example with n variables that can be changed independently after applying an action a that synchronizes them in the initial state. Here, the state space under stubborn sets pruning is linear because after applying a to $\mathcal{I}$ only a single applicable action will be in any stubborn set. Decoupled search, however, cannot exploit the independence because of leaf-leaf interactions that forbid a star factoring, but never trigger during search.

6.3 Symbolic State Representation

Decoupled search relates to search with compact state-set representations, in particular with binary decision diagrams (BDD) (e. g. Bryant, 1986; McMillan, 1993; Edelkamp, 2003b), to the extent that a decoupled state $s^{\mathcal{F}}$ is a compact representation of a state set, namely its hypercube $[s^{\mathcal{F}}]$. This connection is weak, of course, as hypercubes are very particular state sets, pertaining to particular points in the search, whereas BDDs are used to represent large parts of the search space. Hypercubes always have a small representation by definition; whereas BDDs can represent arbitrary state sets and, consequently, may become large. The source of reduction is different in nature, where decoupled search exploits component independence, BDDs rely on the fact that large state sets can be characterized with a boolean function that results in a compact representation in the decision diagram.

We next show that decoupled search is exponentially separated from symbolic breadth-first search with BDDs. The full proof is given in Appendix A.5.

Theorem 22. *Decoupled search is exponentially separated from symbolic breadth-first search with BDDs.*

The task family that leads to an exponential blow-up in the symbolic representation essentially encodes the function that represents the adjacency matrix of an $n \times n$ grid, which has been proven to necessarily cause blow-ups in BDDs [Edelkamp and Kissmann, 2011].

For the opposite direction, examples where symbolic search has a polynomial representation size and decoupled search shows an exponential blow-up can be constructed easily. The Gripper domain, which is a standard benchmark in classical planning, for example, is known to have a compact BDD representation [Edelkamp and Kissmann, 2008], but cannot be decoupled due to pairwise variable dependencies.

6.4 Factored Planning

Decoupled search has been inspired by factored planning, which also partitions the state variables into factors [Sacerdoti, 1974; Knoblock, 1994; Lansky and Getoor, 1995; Amir and Engelhardt, 2003; Brafman and Domshlak, 2006; Kelareva et al., 2007; Brafman and Domshlak, 2008; Fabre et al., 2010; Nissim et al., 2010; Crosby et al., 2013; Brafman and Domshlak, 2013; Nissim and Brafman, 2014; Wang and Williams, 2015].

In *localized* factored planning, the planning process is formulated as local planning on individual components, followed by global cross-component constraint resolution. The latter is done in terms of constraint satisfaction or message-passing methods over exhaustive sets of local plans. Several works have analyzed the worst-case complexity of such planning processes in depth, with major sources of exponential complexity being the length of local plans, and the treewidth of the global constraint graph (the interaction graph, in our terminology). Decoupled search eliminates both of these complexity sources by restricting the form of global interactions allowed.

The most recent, and empirically the most successful, localized factored planning method is *partition-based pruning* [Nissim and Brafman, 2014], which originates from multi-agent planning. The local planning here takes place as part of a state-space search, where a local component state space—an agent's "private" part of the search—is pursued exclusively, until "public" state changes (relevant to other components) are encountered. This can be viewed as a form of partial-order reduction, exploiting permutability of local plans across components.

In *hierarchical* factored planning, the factors are used within a hierarchy of increasingly more detailed levels, accumulating the factors processed so far as search proceeds down the hierarchy. As a high-level plan may not be realizable at lower levels, one needs to allow backtracking across levels. In decoupled search, the maintenance of compliant paths (low-level plans in a 2-level hierarchy with bidirectional dependencies) eliminates that need.

Almost all factored planning approaches cannot guarantee global plan optimality; sometimes, it is a hypothetical but impractical possibility involving exhaustive search over all local plan lengths. The single exception is the work by Fabre et al. [2010], which uses finite automata to represent the local-plan languages (the sets of local plans) at any point in the search.

Algorithmically, decoupled search is quite different from both, localized and hierarchical factored planning. At the core of the differences is the assumption of a particular structural profile, a star topology, in our method; vs. the handling of arbitrary cross-factor interactions in previous methods. Consequently, whereas previous works concentrate on the resolution of complex interactions—via constraint satisfaction, tree decomposition, message passing, backtracking—our work concentrates on algorithms specialized to star topologies, which facilitate a much different direct handling of the simple interactions between center and leaves. In particular, thanks to the latter, our algorithms end up being like state-space search, only on a more complex state structure; this is not the case for any previous factored planning method (save partition-based pruning which is a partial-order reduction technique). The only strong conceptual commonality between decoupled search and factored planning thus remains the use of a state variable partition.

Turning our attention to exponential separations, consider first localized factored planning. It is easy to see that our scaling example is an exponential separation from partition-based pruning, as the only private actions are those of the truck, which do not affect the exponential behavior in the number n of packages. For most other localized factored planning approaches, exponential separations result from their exponential scaling behaviour in local plan length [Amir and Engelhardt, 2003; Brafman and Domshlak, 2006, 2008, 2013]. For example, one can modify our scaling example so that packages have to traverse a more complex internal state space in addition to being loaded/unloaded (leaf state spaces can be made arbitrarily complex, without increasing the number of decoupled states, so long as the additional transitions do not have preconditions on the center). Similarly, the framework of Fabre et al. [2010] scales exponentially on leaf factors whose sets of local plans have no compact finite-automata representation.

Regarding hierarchical factored planning, in our scaling example such approaches will start with high levels containing only the packages (planning “from the leaves to the root”). So the example becomes an exponential separation when each object may choose between several goal paths. For example, this happens when introducing several trucks which serve different parts of the map.

6.5 Other Methods

Symmetry reduction (e. g. Starke, 1991; Emerson and Sistla, 1996; Fox and Long, 1999; Rintanen, 2003; Domshlak et al., 2012) reduces the search space by not distinguishing between states that are symmetric. Given a symmetry relation, for each new state s in the search it is checked whether a state s' symmetric to s has been visited before. If that is the case, then s is pruned. That way, symmetry breaking can exponentially reduce the search effort. The achieved reduction, however, is in general incomparable to that of decoupled search. It is complementary to decoupled search as leaf factors do not need to be symmetric at all. What is more, even if leaf factors *are* symmetric, decoupled search may have a stronger effect than symmetry reduction. Our scaling example is an exponential separation because even perfect symmetry breaking still keeps track of the number of objects at every location. Vice versa, consider again the gripper domain mentioned in Chapter 6.3. The domain is about moving balls between two room with a robot. Here, all balls are symmetric and the search space using symmetry breaking is polynomial, while it is exponential for decoupled search, since it cannot be decomposed reasonably.

Fork/inverted-fork factorings may be somewhat reminiscent of *safe abstraction* for automatic planning-task specification. This method is rooted in early works on hierarchical factored planning with the *downward refinement* property [Bacchus and Yang, 1994; Knoblock, 1994], where all abstract plans are realizable at lower levels. Helmert [2006a] introduced this as an optimization in (an early version of) the Fast Downward system [Helmert, 2006b], removing inverted-leaf variables with strongly connected state spaces. Haslum [2007] generalized this method to consider only the relevant variable values, Tozicka et al. [2016] introduced a related reduction technique merging mutually reachable variable values. All these methods work at the level of single state variables, not factors; none of them can guarantee global optimality; downward refinement is only possible in case of single-directional dependencies, unable to handle more general star topologies. Furthermore, all these techniques rely on mutual reachability between variable values, so that exponential separations are trivial to construct by removing “backwards-actions” not needed in the plan (e. g., in our scaling example, truck moves towards l_1 and unloading actions at locations other than l_m).

Analyzing task structure has a long tradition in planning. The *causal graph*, which we use for factoring, has been extensively explored for *complexity analysis*, in particular the identification of tractable fragments [Jonsson and Bäckström, 1995; Domshlak and Dinitz, 2001; Brafman and Domshlak, 2003; Helmert, 2004; Katz and Domshlak, 2008; Giménez and Jonsson, 2008, 2009; Giménez and Jonsson, 2012; Katz and Keyder, 2012; Aghighi et al., 2015]. Recent research has extended this to the consideration of fixed-parameter tractability, with "backdoors" to planning encapsulating the exponential part of the search [Kronegger et al., 2014, 2015]. While many structural aspects considered in such research relate to our factoring types, decoupled search is directed not at tractable parts of the task, but at worst-case exponential parts that interact in a limited manner. That said, work on the identification of backdoors might inspire new factoring methods in the future.

Compositional verification (e. g. Pnueli, 1985; Clarke et al., 1989; Cobleigh et al., 2003) checks properties of composed systems by making assumptions on, and proving guarantees for, the behavior of their components. This is very different from the interleaved exploration of center paths vs. compliant leaf paths in decoupled search.

Chapter 7

Experimental Evaluation

In this chapter we will provide an experimental evaluation of decoupled search. We will compare its performance in the three algorithmic planning problems—optimal planning, satisficing planning, and proving unsolvability—to state-of-the-art planners. Extending the theoretical comparison to related techniques presented in Chapter 6, we will compare decoupled search empirically to planners based on orthogonal state-space-pruning techniques, namely partial-order reduction, symmetry breaking, symbolic representations, Petri-net unfolding, and partition-based pruning.

We start by describing our implementation and experimental setup. Our first evaluation compares the different factoring strategies presented in Chapter 5. Before showing our evaluation for each of the three algorithmic planning problems, we take a closer look at the state-space representation size of decoupled search and related methods.

7.1 Implementation

Our implementation of decoupled search is based on the Fast Downward (FD) framework [Helmert, 2006b][1], extending its C++ search component. Our decoupled-search fork of FD is publicly available.[2] FD is the most commonly used framework in classical planning, and it contains state-of-the-art heuristic search algorithms. Our implementation is modular in that it affects only FD's state representation, and the computation of state transitions. All of its search algorithms can be run via Proposition 6; all of its heuristic functions can be run (in principle) via Theorem 6. In our experiments, we instantiate these connections with canonical baseline algorithms. These algorithms are orthogonal to ours, and their details are not needed to understand our results. Hence we only give a brief overview. The search algorithms we use are:

[1]`http://www.fast-downward.org`

[2]`https://gitlab.com/dgnad/decoupled-fast-downward/`

- A^*, which is used in most state-of-the-art heuristic search optimal-planning systems [Hart et al., 1968; Pearl, 1984].
- *Greedy best-first search (*GBFS*)* [Doran and Michie, 1966], specifically FD's GBFS implementation with *lazy state evaluation* and optionally a dual-queue open list for *preferred operators* [Richter and Helmert, 2009]. This algorithm is canonical for satisficing heuristic search planning, in that, if used with a strong heuristic, it has state-of-the-art performance.

  Preferred operators are applicable actions used in the abstract plan internally generated by a heuristic. One of the two open lists uses only these actions; the other open list retains the states generated by other actions, preserving completeness. For decoupled search, we restrict the set of preferred operators to center actions.

We extended the following heuristics to support our decoupled state representation:

- h^{max} [Bonet and Geffner, 2001], a basic admissible heuristic function mainly useful for detecting unsolvable states during search. The heuristic estimates remaining cost by assuming that, from each subgoal conjunction, it suffices to achieve the single most costly variable value.
- $h^{\mathrm{LM\text{-}cut}}$ [Helmert and Domshlak, 2009], a canonical admissible heuristic function, close to the state-of-the-art in optimal planning and widely used as a baseline in that context. The heuristic estimates remaining cost through identifying *landmarks* [Hoffmann et al., 2004; Karpas and Domshlak, 2009], action sets that must be hit by every plan.
- h^{FF} [Hoffmann and Nebel, 2001], a canonical inadmissible heuristic function, widely used in satisficing-planning systems. The heuristic estimates remaining cost through the cost of an abstract plan pretending that state variables accumulate, rather than switch between, their values (this is commonly known as the *delete relaxation*, McDermott [1999]; Bonet and Geffner [2001]).
- A *blind* heuristic, which returns 0 on goal states and the cost of the cheapest action otherwise. Using this in A^* is the most common design of a cost-optimal blind search in FD.

For the decomposition of the input planning task for decoupled search, we implemented the following factoring strategies to compute star factorings:

- Fork and inverted-fork (IFork) factorings as described in Chapter 5.3, maximizing the number of leaf factors.

  We do not employ more complex strategies based on these strategies that extend the causal graph leaf SCCs, since sticking with the leaf SCCs gives consistently good results. Combining fork and inverted-fork factorings in an X-shape strategy almost always simplifies to a portfolio that returns either of the two, since X topologies are not common in our benchmarks, so we focus on the simpler topologies.

- *Incident-arcs* strategy (IA): we use the algorithm shown in Figure 5.2.
- Factorings based on causal graph *maximum independent sets* (MIS): we use the algorithm shown in Figure 5.1.
- Strategies based on *linear programming*: we distinguish between strict-star (LPS) and general-star (LPG) encodings, as described in Chapter 5.4. We solve the maximum independent set optimization problem with the IBM CPLEX solver[3], maximizing leaf mobility.

As the benefit of decoupled search lies in avoiding the multiplication of states across leaves, we *abstain* from solving a task if the factoring strategy returns a single-leaf factoring. In such cases, it is more reasonable to use explicit-state search instead, since the overhead of handling decoupled states outweighs the reduction achieved, which is linear in the number of leaf-only actions. We implemented the checks described in Theorems 8 and 9 to efficiently decide if a factoring with at least two leaves is possible. Thus, there is little overhead before taking the decision to abstain and handing over to standard search.

As decoupled states explicitly enumerate leaf-factor state spaces, these should not be excessively large. We apply a simple sanity criterion, where we check that all leaves L satisfy the size bound $\prod_{v \in L} |\mathcal{D}(v)| < 2^{32}$. While this bound may seem extremely high, note that the *reachable* leaf state spaces are typically significantly smaller. Without running the search, however, we do not know how many leaf states will be reached, so we opt for this simple approximation.

A crucial question when implementing decoupled search is how to represent and process decoupled states during execution. To efficiently store leaf states across decoupled states and maintain pricing functions, we precompute the component state space of each leaf factor $L \in \mathcal{L}$ just once before search begins by projecting the task onto L, pretending that all conditions outside L are achieved.[4] This gives an over-approximation of the set of leaf states reachable during search. We can hence give IDs to the leaf states and store the reachability functions and pricing functions efficiently as bit-vectors, respectively vectors of numbers, assigning $\top$, respectively the price, to the entry with the corresponding leaf state ID. We reconstruct compliant-path graphs merely once the

[3]`https://www.ibm.com/analytics/cplex-optimizer`

[4]We limit the size of the leaf state spaces that are precomputed for non-optimal planning to at most 10,000 states, since the memory and runtime overhead can be prohibitive otherwise. This only pertains to leaf-state transitions, though, we still compute IDs for leaf states and store them only once.

search terminates successfully on a decoupled goal state. This construction is linear in the length of the decoupled plan and polynomial in the size of the leaves, so efficient.

Another important part of the implementation is dominance pruning. As indicated in Gnad [2021b], dominance pruning can consume a significant percentage of the overall search time, in particular in optimal planning where up to more than 90% of the time is spent on dominance pruning. The underlying issue is that dominance cannot be checked efficiently, in contrast to exact duplicate checking in explicit-state search that is usually done via hashing. Since dominance of decoupled states is just a generalization of equivalence, as pointed out in Chapter 3.4.1, we also experiment with exact duplicate checking for decoupled states, trading an efficient duplicate check for the pruning power of dominance. Like for explicit states, we can simply use hashing to decide if two decoupled states differ.

For the implementation of hypercube pruning, we use an algorithm called *cube elimination* proposed by Hoffmann and Kupferschmid [2005]. We also experimented with an encoding into SAT, and a SAT solver API (Z3, Moura and Bjørner [2008]), but found our specialized cube elimination implementation more efficient. Still, hypercube pruning consumes prohibitive amounts of runtime and memory. We enable hypercube pruning in the experiments in Chapter 7.4, to show its effect on the reachable state space size when exhausting the decoupled state space. It remains an open question whether hypercube pruning can be made effective in practice.

7.2 Experimental Setup

We consider five different types of evaluations. First, we investigate the effectiveness of the factoring strategies, i. e., in how many instances are they successful, how many leaves are produced, what is the average leaf mobility. We then take a closer look at the representation size of the reachable state space under decoupled search, vs. explicit-state search, and related reduction techniques, namely partial-order reduction, symbolic state representation, symmetry breaking, and Petri-net unfolding. We do so by exhausting the entire reachable state space using these methods, pretending that no goal state is reachable. We then address the three main algorithmic planning problems, satisficing planning (finding any plan), optimal planning (finding an optimal plan), and proving unsolvability (proving that no goal state is reachable).

In our evaluation, we consider the planning systems listed below, most of which are implemented in Fast Downward (FD). Some of the planners have competed in one of the International Planning Competitions (IPC) of the past years, which were held from 1998 to 2018.[5]

- Explicit-state search (Base), the default in FD.

[5]`http://www.icaps-conference.org/index.php/Main/Competitions`

- Partition-based pruning (PP) [Nissim and Brafman, 2014], a pruning method which is a variant of factored planning originating from multi-agent planning, implemented in FD.
- Partial-order reduction via strong stubborn sets (POR). We use the variant *SSS-EC* that is implemented in FD [Wehrle et al., 2013].
- Symmetry breaking (Sym), in particular the orbit search algorithm from [Domshlak et al., 2015b], which is also implemented in FD.
- A combination of the former two methods, stubborn sets pruning and symmetry breaking in explicit-state search (PSy).
- A version of FD extended with symbolic state representation [Torralba et al., 2017] via binary decision diagrams (SBD). We use a blind bidirectional symbolic search, which according to Torralba et al. [2017] performs very similar to the Symba planner, the winner of the optimal track in IPC'14 [Torralba et al., 2014].
- Two Petri-net unfolding tools, Cunf [Rodríguez and Schwoon, 2013], and Mole [Bonet et al., 2014]. Cunf performs *contextual unfolding* [Baldan et al., 2012], which has native support for prevail conditions (compare Chapter 6.1). Cunf constructs a finite prefix, so can only be used to exhaust the state space. Mole does directed unfolding with a heuristic and can perform optimal and satisficing planning as well [Bonet et al., 2008]. It supports blind search, $h^{\max}$, and h^{FF}.
- LAMA is a strong planner commonly used as baseline for comparisons in satisficing planning [Richter and Westphal, 2010; Richter et al., 2011]. It is based on landmarks [Richter et al., 2008] in combination with the delete-relaxation heuristic h^{FF} [Hoffmann and Nebel, 2001]. LAMA is implemented in FD.
- Best-First Width Search (BFWS) is a search algorithm based on the notion of *state novelty* [Lipovetzky and Geffner, 2012, 2017; Francès et al., 2018]. We use the variant called DUAL-BFWS. BFWS is not based on FD and uses a different preprocessing that leads to alternative task encodings. BFWS is the runner-up of IPC'18 in the satisficing track. We do not include the winner here, since it is a portfolio approach that runs many planner configurations with short time limits sequentially. Therefore, its results are not very consistent and not well suited for our evaluations.
- The Complementary planner (C2) competed successfully in IPC'18, being runner-up in the optimal planning track [Franco et al., 2018]. We include it here, and not the winner, since the winner is based on a learned portfolio that was trained on

a subset of the domains that we use in our evaluation. It is based on symbolic pattern databases [Franco et al., 2017]. $\mathrm{C2}$ is implemented in FD.

- The Sympa planner [Torralba, 2016], the runner-up of the Unsolvability IPC 2016, which is build on top of the Symba planner [Torralba et al., 2014]. The UIPC'16 winner is also a portfolio, so we do not include it in our comparison. Sympa is implemented in FD.

Many of the planners that participated in an IPC make use of an enhanced preprocessing that can decrease the size of the FDR planning task significantly by pruning irrelevant parts [Alcázar and Torralba, 2015]. We disable this so-called h^2-preprocessor in all planners to get a fair comparison.

For both unfolding tools, we construct a Petri net based on the FDR encoding produced by the translator component of FD [Helmert, 2009], where an FDR fact directly corresponds to a net place. Care must be taken with planning actions with (1) prevail conditions, i. e., variables defined in the preconditions, but not in the effect of an action, and (2) effect-only variables which are not defined in the precondition. Prevail conditions (1) are only an issue for Mole, where we need to add a redundant effect to the net transition. In Cunf, we can simply encode prevail conditions with read arcs. Effect-only variables (2) can lead to a blow-up in the net, since we need to add preconditions for all combinations of facts of the corresponding variables to obtain a *safe* net [Hickmott et al., 2007]. We also experimented with an encoding into *transition normal form*, which avoids the blow-up, but makes plans longer [Pommerening and Helmert, 2015]. For Cunf, the exponential encoding always worked better. For Mole, the results were mixed, different encodings benefit different settings. In the following sections we will indicate which encoding was used.

For decoupled search, we distinguish the following configurations:

- DS: basic configuration of decoupled search, using a pricing function, augmented-cost dominance $\preceq_{\mathrm{aug}}$, g-value Adaptation, and a star heuristic in optimal planning, and a reachability function, basic dominance pruning $\preceq$, and a center heuristic in non-optimal planning.

- $\mathrm{opt/sat}$: denote the optimal/non-optimal planning variant of DS in Chapter 7.4.

- dup: like DS, but with exact duplicate checking instead of $\preceq_{\mathrm{aug}}$.

- hc: sat with hypercube pruning.

- aT: an implementation of the Anytime D-A^* algorithm described in Chapter 4.2.1.

- sd: like DS for optimal planning, but using the *simple dominance* pruning $\preceq$ introduced in Chapter 3.4.1, and disabling the g-value Adaptation from Chapter 4.3.

Our comparison is done on the established planning benchmarks introduced in the International Planning Competitions. The benchmarks are grouped into *domains* that contain *instances* described using the same action rules. Domains and instances are modeled in the Planning Domain Definition Language (PDDL) [McDermott et al., 1998], which planners translate into an FDR task before attempting to find a plan. We use all domains and instances from the deterministic optimal, satisficing, and unsolvability tracks of all IPCs. To extend the set of unsolvable benchmarks, we show results for the domains introduced in Hoffmann et al. [2014] that have not been reused in the Unsolvability IPC 2016. Since there is a large overlap of benchmark instances used in different competitions, we filter out duplicate instances, i. e., instances where the PDDL files are identical. This results in 2727 instances distributed across 62 domains. We aggregate domains that appeared in several iterations into a single entry.

In all settings except the optimal planning results from Chapter 7.6, we apply a cost transformation to all instances, setting the cost of all actions to 1. These *unit costs* can often be exploited by heuristics and search algorithms. For decoupled search, we use reachability functions in these cases, anyway, so distinguish only if a leaf state is reached or not. Thus, the unit costs only affect the center actions.

In our evaluation, we mainly report two types of results: (1) *coverage*, i. e., the number of instances *solved* by a planner (respectively proved unsolvable), and (2) *scatter plots* showing a per-instance comparison of the runtime, memory consumption, or the size of the generated search space of two planners. For (1), we use tables that, for each pair of a planner and a domain show the coverage. Each such table only contains decoupled search configurations that use the same factoring strategy. This allows us to filter out instances in which that strategy abstains, focusing on the subset of instances in which decoupled search is actually effective. Thus, the instance basis for tables with different factoring strategies is not the same.

We include a comparison across strategies at the end of each section, taking all instances into account. In this comparison, if a strategy abstains, we run explicit-state search instead.[6] In the factoring comparison, we also include results for a *best combined factoring portfolio* ($\langle \mathcal{F}_\mathrm{i} \rangle$), which simulates a sequential portfolio of several factoring strategies. This is a fixed sequence of strategies, where, on each instance, the factoring obtained by the first successful strategy is employed. If none of the methods returns a factoring, we run Base. If, for example, the best portfolio for a setting is $\langle \mathrm{Fork}, \mathrm{LPG} \rangle$, then we run decoupled search with factorings obtained by Fork; where Fork fails, we employ factorings of LPG; if both fail, we run Base. Then, an instance is considered solved if the sum of (the preprocessing time and) the factorings times plus the search time are below the time limit. We compute the best sequence of strategies by simulat-

[6]We did not actually run the experiments like this, but rather combined the results of explicit-state search and decoupled search by considering an instance solved if either decoupled search does not abstain and solves the task, or decoupled search abstains, but explicit-state search solves the task such that (preprocessing time plus) factoring time plus search time of explicit-state search are below the time limit.

ing all possible sequences, and return the sequence that maximizes total coverage. We remark that, although the results for $\langle \mathcal{F}_{\mathrm{i}} \rangle$ in our evaluation are only simulated by combining the results of several runs, this portfolio could easily be implemented in practice without additional overhead.

The factoring evaluation takes the form of a table that shows a pairwise comparison of all factorings, plus Base, $\langle \mathcal{F}_{\mathrm{i}} \rangle$, and some strong competitors. An entry in row X and column Y indicates in how many domains method X solved strictly more instances than method Y. We also show the total coverage across all domains of all techniques.

The scatter plots (2) compare two planners directly, with one planner on the x-axis and the other on the y-axis. Each point in a plot corresponds to one instance that was solved by at least one of the planners. For runtime data, for example, a point below the diagonal line (which is shown in each plot), corresponds to an instance where the method on the y axis terminated faster than the method on the x-axis. All plots are in log-log scale. When comparing the size of the search space, we consider the number of expanded states for the state-space exhaustion and proving unsolvability; the number of expanded states until the last f-layer of A^* for optimal planning, and the number of state evaluations for satisficing planning. For unfolding, we take the total number of histories of non-cut-off events for Cunf, Mole does not output useful statistics. BFWS and SBD do not have any measure for state-space size that could be reasonably compared to the statistics of FD.

The evaluation was performed on a cluster of Intel E5-2660 machines running at 2.20 GHz, running a Fedora 33 Linux. We used the Downward Lab Python package to conduct the experiments [Seipp et al., 2017].

We use the common memory and runtime limits of 30min and 4GiB of memory. For all factoring strategies we impose a time limit of 30 seconds (except in the next section). As we shall see in Chapter 7.3, the factoring time is typically small and all strategies are able to identify a factoring within the time limit in almost all instances.

The source code of our decoupled search implementation as well as all evaluation data are publicly available [Gnad, 2021a].

7.3 Factoring Statistics

In this section, we provide an analysis of the factoring strategies. We did *not* impose a separate time limit for the factoring process in this evaluation, but wait until the process terminates, or the 30 min overall time limit is reached. First, we want to briefly look into the runtime and memory requirements of the factoring process:

The checks described in Theorems 8 and 9 are usually very efficient, with a median (mean) runtime across all instances rounded to 0.0s (0.37s). The maximum time to perform this check is 169.13s, which is in an instance of the Organic domain that has more than 200.000 actions.

Strategy	Median	Mean	Max	#> 30s	#Timeout
Fork	0.003s	0.12s	103.077s	1	0
IFork	0.003s	0.11s	89.5538s	1	0
IA	0.006s	0.63s	193.213s	14	0
MIS	0.009s	12.31s	> 1800s	88	191
LPS	0.014s	28.17s	> 1800s	190	51
LPG	0.017s	31.72s	> 1800s	260	110

Figure 7.1: Runtime statistics of the factoring strategies when not imposing a time limit on the left. On the right, we show in how many instances the factoring time is > 30s, respectively > 1800s. The total number of instances considered is 2679.

The actual strategies are typically fast, too, but some fail to terminate on huge instances. We show detailed statistics in Figure 7.1. The Fork and IFork strategies finish almost instantaneously for all instances; the only instance where the runtime is >30s is in Organic. The IA strategy shows a similar runtime behaviour, yet, as we will see soon, is a lot more effective in computing factorings. MIS, LPS, and LPG all run into the timeout of 30 min in many instances. Recall that all three at some point solve an **NP**-hard subproblem, so it is not surprising that on huge instances this can be prohibitive.

An important question when looking at the cases with high factoring times is whether there is any benefit in giving the strategy arbitrary time to terminate. It turns out that this is not the case. Fork, IFork, and IA are successful in the same instances if the factoring is cut off after 30s and the best factoring obtained until that point is returned, if one has been found. For MIS we implemented a mechanism that terminates the computation of the causal-graph `MIS` after the timeout, still doing the post-processing described in the algorithm in Figure 5.1, which is always fast, on the largest independent set found until then. This mechanism actually *increases* the number of instances where MIS is successful by 78. For LPS and LPG, though, imposing the time limit we lose 48, respectively 124, instances. However, we observed that even if a factoring is detected, then decoupled search does usually obtain only a small state-space reduction in these instances. We conjecture that in particular the ILP-based strategies typically perform bad if the planning task is tighly coupled, having to satisfy many constraints induced by the edges in the potential-leaf graphs. Hence, even if a factoring is returned, there will probably be few leaves, or a lot of cross-factor interaction, leading to a poor reduction of decoupled search. Therefore, in the coming sections, we will limit the factoring time to 30s, without losing many instances where decoupled search would perform well.

The MIS, LPS, and LPG strategies occasionally run out of memory, too. This happens in 12 instances for MIS (in 3-SAT, Maintenance, Organic-Split, PegSolR5), 1 instances for LPS (Organic-Split), and 11 instances for LPG (BagBarman, Organic-Split, Tidybot). When imposing a time limit of 30s, the memory limit is never reached.

Figure 7.2 provides detailed results on the number of instances successfully decomposed, the obtained type of factoring, the number of leaf factors, and the average leaf mobility. We group the results by domain, where the column # shows the total number of instances of a domain. The column #$\mathcal{F}$ shows the number of instances in which the checks in Theorems 8 and 9 imply that a mobile factoring with at least two leaves exists. The subsequent columns show the number of fork (F), inverted-fork (IF), strict-star (SS), and general-star (GS) factorings obtained by the respective factoring strategies. For all instances where a strategy returned a factoring, #L denotes the average number of leaf factors, M denotes the average of the per-instance leaf mobility (the total number of leaf-only actions).

First and foremost, it is remarkable that almost all domains can be decomposed for decoupled search. Only in Blocksworld, PegSol, Snake, Sokoban, and VisitAll, there provably exists no instance where a mobile 2-leaf factoring is possible at all. Overall, more than 86% of the instances can be tackled with decoupled search. The average number of leaf factors and average leaf mobility indicate furthermore that in many domains we can expect a strong state-space reduction.

Comparing the different strategies, we observe that Fork and IFork are often orthogonal, returning factorings in different domains. Still, there are seven domains where both types of factorings are possible, indicating a loosely coupled causal graph. Since these strategies are complete, i. e., detect a (inverted-)fork factoring if one exists, the results show that in 563 instances of our benchmark set there exists a fork factoring, and in 589 instances there exists an inverted-fork factoring.

IA, which is very greedy and comes without any guarantee, proves very successful. It finds factorings in all domains where there exist fork or inverted-fork factorings, except BagTransport. It does not succeed in all instances in these domains, though, and sometimes returns strict-star factorings instead. The MIS strategy shows not as good results as IA. We conjecture that the limited post-processing performed on the causal-graphs `MIS`s is not enough to "repair" bad independent sets into mobile factorings.

The ILP-based strategies yield by far the best results, with more than 1900 computed factorings each getting significantly closer to the possible maximum of 2313. Comparing the two strategies in detail reveals that they often agree, i. e., there are no general-star factorings with higher leaf mobility than that of strict-star factorings. There are many domains, however, where LPG achieves a higher mobility, so prefers a general-star factoring. Note that LPG also has a certain overhead compared to LPS, which is due to the more complex potential-leaf graphs. This results in a few instances where LPS returns a factoring, but LPG does not (e. g., in Airport, BagBarman, PegSolR5).

			Fork			IFork			IA					MIS					LPS					LPG					
Domain	#	#$\mathcal{F}$	F	#L	M	IF	#L	M	F	IF	SS	#L	M	F	IF	SS	#L	M	F	IF	SS	#L	M	F	IF	SS	GS	#L	M
Agricola	40	40	0	-	-	0	-	-	0	0	40	7.7	64.5	0	0	0	-	-	0	0	40	8.7	13.8k	0	0	0	40	16.4	35.7k
Airport	50	44	0	-	-	0	-	-	0	0	0	-	-	0	0	10	3.4	38.2	0	0	44	5.5	257	0	0	3	40	5.3	246
Barman	74	74	0	-	-	0	-	-	0	0	2	17.0	170	0	0	0	-	-	0	0	74	10.6	84.6	0	0	0	74	12.6	88.6
Blocksworld	35	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Childsnack	30	30	0	-	-	30	3.1	36.8	0	30	0	3.1	36.8	0	0	0	-	-	0	6	24	16.4	59.9	0	6	24	0	16.4	59.9
DataNetwork	40	40	0	-	-	0	-	-	0	0	40	96.9	96.9	0	0	40	96.9	96.9	0	0	40	75.4	260	0	0	0	40	97.0	515
Depots	22	22	0	-	-	22	2.8	120	0	22	0	2.8	120	0	11	0	3.3	193	0	22	0	2.8	120	0	22	0	0	2.8	120
Driverlog	20	20	20	8.3	1045	0	-	-	6	0	14	9.1	1150	12	0	8	9.8	1143	0	0	20	11.2	1319	0	0	20	0	11.2	1319
Elevators	70	70	0	-	-	70	4.6	209	0	40	0	3.9	107	0	0	5	4.4	119	0	70	0	4.6	209	0	10	0	60	4.6	227
Floortile	70	70	0	-	-	70	2.3	4.6	0	0	70	6.2	55.5	0	0	70	14.2	116	0	0	70	13.9	116	0	0	0	70	28.1	232
Freecell	80	80	0	-	-	0	-	-	0	0	42	2.2	6.9	0	0	0	-	-	0	0	79	3.6	12.9	0	0	0	80	4.9	306
GED	40	40	0	-	-	0	-	-	0	0	40	13.4	26.9	0	0	40	13.4	26.9	0	0	40	13.4	37.4	0	0	0	40	26.9	253
Grid	5	5	0	-	-	0	-	-	0	0	5	11.8	207	0	0	5	11.8	207	0	0	5	11.8	207	0	0	5	0	11.8	207
Gripper	20	20	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Hiking	30	30	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	25	2.3	136	0	0	25	0	2.3	136
Logistics	63	63	63	11.2	5953	63	23.3	2113	29	34	0	25.3	2226	29	34	0	25.3	2226	60	3	0	13.5	6047	60	3	0	0	13.5	6047
Maintenance	25	25	1	2.0	27.0	1	2.0	27.0	0	0	14	5.0	7.6	0	0	5	7.6	11.6	0	0	25	10.6	12.3	0	0	0	15	45.5	49.2
Miconic	150	145	145	16.0	32.0	0	-	-	145	0	0	16.0	32.0	145	0	0	16.0	32.0	145	0	0	16.0	32.0	145	0	0	0	16.0	32.0
Mprime	35	35	0	-	-	0	-	-	0	0	6	2.0	632	0	0	0	-	-	0	0	35	2.0	133	0	0	31	4	2.0	134
Mystery	30	25	0	-	-	4	2.5	135	0	0	1	2.0	109	0	0	0	-	-	0	0	8	2.2	25.6	0	0	1	24	2.3	91.2
NoMystery	40	40	40	9.0	190	0	-	-	40	0	0	9.0	190	40	0	0	9.0	190	40	0	0	9.0	190	40	0	0	0	9.0	190
Openstacks	135	128	0	-	-	0	-	-	0	0	105	43.5	48.3	0	0	88	34.3	34.3	0	0	128	72.7	72.7	0	0	23	105	72.6	283
Organic	40	6	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	1	2.0	4.0	0	0	0	5	2.0	1107
Organic-Split	40	39	0	-	-	5	48.8	0.0	0	0	35	11.1	7573	0	0	0	-	-	0	0	5	8.4	656	0	0	0	9	21.0	1815
ParcPrinter	30	28	0	-	-	0	-	-	0	0	20	2.2	175	0	0	0	-	-	0	0	26	4.2	143	0	0	11	16	6.3	222
Parking	80	80	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Pathways	30	30	29	21.1	42.2	0	-	-	0	0	30	40.6	73.6	0	0	30	24.7	38.5	0	0	30	52.0	172	0	0	0	30	57.2	183
PegSol	36	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
PetriNet	20	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Pipesworld	100	100	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	61	2.4	6.4	0	0	0	70	9.4	13.4
PSR	50	50	3	2.0	4.0	0	-	-	0	0	48	6.8	8.9	0	0	48	7.2	7.8	0	0	48	2.9	255	0	0	13	37	3.7	255
Rovers	40	40	40	20.9	1122	38	7.7	731	0	0	40	21.4	2382	12	0	28	18.4	979	0	0	40	33.9	3530	0	0	40	0	33.9	3530
Satellite	36	36	36	54.8	525	34	7.4	108k	7	0	29	33.9	4043	26	1	9	54.2	1071	0	0	36	14.1	102k	0	0	36	0	14.1	102k
Scanalyzer	30	30	0	-	-	0	-	-	0	0	9	8.6	182	0	0	9	8.7	226	0	0	9	4.9	3151	0	0	9	0	4.9	3151
Snake	40	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Sokoban	40	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Spider	40	40	0	-	-	0	-	-	0	0	0	-	-	0	0	3	3.3	7.0	0	0	6	9.2	15.2	0	0	0	16	18.9	60.8
Storage	30	25	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	17	2.0	10.2
Termes	40	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Tetris	37	37	0	-	-	0	-	-	0	0	30	12.3	41.2	0	0	0	-	-	0	0	6	13.8	27.7	0	0	0	9	80.7	161
Thoughtful	20	20	0	-	-	0	-	-	0	0	13	2.4	3.5	0	0	5	2.6	5.2	0	0	20	6.0	733	0	0	0	20	19.4	778
Tidybot	40	40	0	-	-	0	-	-	0	0	40	4.3	5.9	0	0	0	-	-	0	0	1	2.0	419	0	0	0	4	6.0	584
TPP	30	30	27	8.3	3856	26	4.8	86.3	0	0	29	10.8	3991	0	0	3	3.0	3.0	0	0	29	27.1	1262	0	0	29	0	27.1	1262
Transport	117	117	0	-	-	117	3.1	713	0	39	0	2.5	171	0	0	6	2.0	17.3	0	117	0	3.1	713	0	0	0	117	3.1	918
Trucks	30	30	0	-	-	0	-	-	0	0	27	11.4	11.5k	0	0	13	51.3	15.1k	0	0	30	32.5	13.7k	0	0	30	0	32.5	13.7k
VisitAll	66	0	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Woodworking	70	70	47	4.1	571	62	4.2	467	0	1	62	7.2	985	0	0	1	2.0	4.0	0	0	70	14.6	772	0	0	2	68	15.6	774
Zenotravel	20	20	20	9.9	1191	18	3.5	6466	0	0	20	13.2	1787	0	0	20	13.2	1787	0	18	2	3.5	5825	0	18	2	0	3.5	5825
$\sum$	2256	1914	471			560			227	166	811			264	46	446			245	236	1077			245	59	304	1010		
Unsolvability IPC 2016																													
BagBarman	20	20	0	-	-	0	-	-	0	0	16	3.0	59.6	0	0	0	-	-	0	0	20	3.9	32.3	0	0	0	12	6.0	52.0
BagGripper	25	2	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
BagTransport	29	29	0	-	-	29	2.9	799	0	0	0	-	-	0	0	0	-	-	0	29	0	2.9	799	0	18	0	4	2.5	205
Bottleneck	25	25	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	0	0	-	-
Cavediving	25	25	0	-	-	0	-	-	0	0	21	14.3	438	0	0	13	7.4	69.2	0	0	25	11.8	376	0	0	0	25	11.8	381
Chessboard	23	23	0	-	-	0	-	-	0	0	0	-	-	0	0	0	-	-	0	0	22	37.2	37.2	0	0	0	23	74.9	74.9
DocTransfer	20	20	0	-	-	0	-	-	0	0	20	35.6	2470	0	0	20	35.6	2470	0	0	20	35.6	2470	0	0	20	0	35.6	2470
NoMystery	23	23	23	14.3	460	0	-	-	23	0	0	14.3	460	23	0	0	14.3	460	23	0	0	14.3	460	23	0	0	0	14.3	460
Rovers	19	19	19	13.2	90.6	0	-	-	0	0	19	18.9	773	0	0	19	20.6	1023	0	0	19	21.3	1025	0	0	1	18	2.7	4070
TPP	30	30	0	-	-	0	-	-	0	0	30	6.3	21.1	0	0	14	3.3	6.6	0	0	30	4.6	112	0	0	30	0	4.6	112
PegSolR5	15	14	0	-	-	0	-	-	0	0	12	2.9	14.3	0	0	0	-	-	0	0	13	16.6	33.2	0	0	0	10	17.1	34.2
PegSol	24	24	0	-	-	0	-	-	0	0	24	4.0	8.0	0	0	0	-	-	0	0	24	4.0	8.0	0	0	0	24	8.0	16.0
Tiles	20	20	0	-	-	0	-	-	0	0	10	2.0	88.0	0	0	0	-	-	0	0	20	2.5	49.0	0	0	20	0	2.5	49.0
Tetris	20	20	0	-	-	0	-	-	0	0	20	9.5	33.5	0	0	0	-	-	0	0	10	16.0	32.0	0	0	0	10	50.0	100
$\sum$	318	294	42			29			23	0	172			23	0	66			23	29	203			23	18	71	126		
Unsolvable Benchmarks from Hoffmann et al. [2014]																													
3-SAT	30	30	0	-	-	0	-	-	0	0	1	2.0	2.0	0	0	0	-	-	0	0	20	3.0	3.0	0	0	0	30	9.0	9.0
NoMystery	25	25	25	15.0	360	0	-	-	25	0	0	15.0	360	25	0	0	15.0	360	25	0	0	15.0	360	25	0	0	0	15.0	360
Rovers	25	25	25	26.6	172	0	-	-	20	0	5	26.8	213	0	0	25	30.0	716	0	0	25	29.8	720	0	0	1	24	2.8	3736
TPP	25	25	0	-	-	0	-	-	0	0	25	8.0	42.2	0	0	0	-	-	0	0	25	7.8	196	0	0	25	0	7.8	196
$\sum$	105	105	50			0			45	0	31			25	0	25			25	0	70			25	0	26	54		
Total	2679	2313	563			589			295	166	1014			312	46	537			293	265	1350			293	77	401	1190		
										1475					895					1908						1961			

Figure 7.2: Statistics for all factoring methods. The number of resulting fork/inverted-fork/strict-star/general-star factorings is denoted by F/IF/SS/GS. The average number of leaf factors is denoted by #L, the average mobility of the factorings by M. #$\mathcal{F}$ is the number of instances per domain where a mobile 2-leaf factoring exists.

7.4 State-Space Size

We herein compare decoupled search to related state-space reduction methods when completely exhausting the reachable state space. This evaluation compares the potential for reduction, avoiding the impact that a heuristic, or stopping at a goal state, can have. In this section, we show results on all IPC benchmarks, i. e., we include domains and instances from all competition tracks described in Chapter 7.2.

In Figure 7.3 we show results on instances where Fork is successful, i. e., returns a factoring with at least two mobile leaves. We compare three variants of decoupled search, opt, sat, and hc to explicit-state search without and with several pruning methods, unfolding (Unf.) with Cunf, and symbolic (Sym.) bidirectional search with SBD. All variants of decoupled search strongly outperform the explicit-state methods and Cunf. There are only three domains (Rovers, Satellite, Woodworking) where pruning with symmetries or stubborn sets can construct more state spaces. Cunf also shows good results in these domains. The strongest competitor is symbolic search, which constructs more state spaces than the optimality-preserving decoupled search variant opt, and decoupled search with hypercube pruning (hc). Interestingly, decoupled and symbolic search excel in different domains, showing their orthogonal reduction effects.

Comparing the decoupled search variants to each other, we clearly see the advantage of the reachability functions, sat can construct 112 more state spaces than opt. This effect is most pronounced in Miconic, where the difference is 99 instances. Hypercube pruning overall clearly deteriorates the performance, which is to be expected given its computational overhead. But this highly varies across domains. In some domains it is just as good as the other variants, often with a higher coverage than opt.

In Figure 7.4, we see the same evaluation for inverted-fork factorings. The difference to the results on fork factorings is obvious. While sat and hc still outperform all explicit-state variants and unfolding, this is no longer true for opt. The configurations that include symmetry breaking (Sym, PSy) can construct more state spaces than opt consistently across most domains. In general, the advantage of decoupled search is significantly reduced, and symbolic search has a higher coverage in many domains. The most significant advantage is in Floortile (49 instances), where it is known that symbolic search performs well. Still, we see that decoupled and symbolic search are orthogonal; no method dominates the other in all domains.

We conjecture that the performance difference between decoupled search with fork and inverted-fork factorings is due to two reasons: (1) the different kind of *monotonicity* briefly discussed in Chapter 3.4.1 and Chapter 5.1, and (2) the different state-space topology. Point (1) highly influences the pruning potential of dominance pruning, as discussed in Example 4, where, in particular in optimal planning, leaf preconditions of center actions can cause diverging pricing functions. The same effect occurs with reachability functions, too, if the leaf state spaces are not invertible, so a commitment to a leaf precondition restricts the set of reachable leaf states in all descendant states.

			Explicit Search					Unf.	Sym.	Decoupled		
Domain	#	#$\mathcal{F}$	Base	PP	POR	Sym	PSy	Cunf	SBD	opt	sat	hc
Driverlog	20	20	5	5	5	6	6	3	8	8	**11**	10
Logistics	63	63	12	12	12	13	13	11	18	25	**27**	18
Miconic	150	145	45	45	40	51	43	30	107	46	**145**	79
NoMystery	40	40	11	11	11	12	12	7	16	28	**30**	13
Pathways	30	29	3	3	3	3	3	3	**4**	3	3	3
Rovers	40	40	5	5	6	5	6	6	**13**	5	5	5
Satellite	36	36	4	5	5	5	5	6	**7**	4	4	4
TPP	30	27	5	4	5	6	6	4	7	11	**14**	8
Woodworking	70	47	7	8	10	7	12	13	**22**	9	10	10
Zenotravel	20	20	7	7	7	7	7	4	7	7	**8**	7
Other	1727	4	4	4	4	4	4	4	4	4	4	4
$\sum$	2226	471	108	109	108	119	117	91	213	150	**261**	161
Unsolvability IPC 2016												
NoMystery	23	23	2	2	2	2	2	1	5	11	**12**	3
Rovers	19	19	6	6	5	6	5	2	**12**	7	7	7
Other	276	0	0	0	0	0	0	0	0	0	0	0
$\sum$	318	42	8	8	7	8	7	3	17	18	**19**	10
Unsolvable Benchmarks from Hoffmann et al. [2014]												
NoMystery	25	25	0	0	0	0	0	0	4	**24**	**24**	2
Rovers	25	25	0	0	0	0	0	0	**5**	0	0	0
Other	85	0	0	0	0	0	0	0	0	0	0	0
$\sum$	135	50	0	0	0	0	0	0	9	**24**	**24**	2
Total	2679	563	116	117	115	127	124	94	239	192	**304**	173

Figure 7.3: Coverage data, i. e., the number of state spaces exhausted completely on instances where Fork does not abstain. #$\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage in **bold face**.

Point (2) is also related to the inter-factor dependencies. Consider the Logistics domain, which is very similar to our logistics running example, where all instances are tackled by both factoring strategies. Here, twice as many state spaces can be constructed using Fork. With a fork factoring, i. e., all trucks in the center, decoupled search only enumerates different sequences of drive actions. Thus, the branching factor compared to explicit-state search is reduced (the same drive actions applicable in a decoupled state are applicable in all its member states). Considering an inverted-fork factoring in the same domain, where each truck forms a leaf and all packages are in the center, the branching factor actually *increases* dramatically. This is because the set of load/unload actions applicable in a decoupled state is the union of all such actions applicable in any member state. As each truck can always reach all locations, unloading is possible

			Explicit Search					Unf.	Sym.	Decoupled		
Domain	#	$\#\mathcal{F}$	Base	PP	POR	Sym	PSy	Cunf	SBD	opt	sat	hc
Childsnack	30	30	0	0	0	**6**	**6**	0	4	0	0	0
Depots	22	22	4	4	3	**5**	4	2	3	4	**5**	**5**
Elevators	70	70	11	13	6	17	13	2	18	17	**23**	**23**
Floortile	70	70	2	2	2	2	2	0	**51**	2	2	2
Logistics	63	63	12	12	12	13	13	11	**18**	13	15	15
Organic-Split	40	5	**1**	**1**	**1**	**1**	**1**	0	**1**	0	0	0
Rovers	40	38	3	3	4	3	4	4	**11**	3	4	4
Satellite	36	34	2	3	3	3	3	4	**5**	3	**5**	**5**
TPP	30	26	2	1	2	3	3	1	**4**	2	2	2
Transport	117	117	19	23	18	23	18	10	24	23	**28**	**28**
Woodworking	70	62	8	10	13	9	15	16	**34**	12	12	12
Zenotravel	20	18	5	5	5	5	5	2	5	5	**8**	**8**
Other	1618	1	1	1	1	1	1	1	1	1	1	1
$\sum$	2226	556	70	78	70	91	88	53	**179**	85	105	105
Unsolvability IPC 2016												
BagTransport	29	29	7	7	5	7	5	3	8	8	**12**	**12**
Other	289	0	0	0	0	0	0	0	0	0	0	0
$\sum$	318	29	7	7	5	7	5	3	8	8	**12**	**12**
Unsolvable Benchmarks from Hoffmann et al. [2014]												
Mystery	30	4	0	0	1	0	**2**	0	0	0	0	0
Other	105	0	0	0	0	0	0	0	0	0	0	0
$\sum$	135	4	0	0	1	0	**2**	0	0	0	0	0
Total	2679	589	77	85	76	98	95	56	**187**	93	117	117

Figure 7.4: Same setup as in Figure 7.3, on instances where IFork does not abstain.

everywhere for all packages loaded in a truck in the center state.

Another interesting observation is that hypercube pruning does not seem to incur an overhead for tasks under IFork. This is probably because the leaf state spaces in the tackled instances are in fact invertible, so even for sat dominance pruning only preserves a single decoupled state per center state.

Figures 7.5 and 7.7 show the same evaluation for factorings obtained by IA and LPG, as representatives that produce strict-star, respectively general-star, factorings. The overall picture is similar in both cases. Overall, all decoupled search variants beat explicit-state search and unfolding, except with LPG, where Sym has higher total coverage than opt. Symbolic search consistently shows the highest coverage.

A more interesting per-domain comparison reveals that all presented methods are quite complementary. While for example POR works really well in ParcPrinter and 3-SAT, symmetries are apparently good in Barman, Childsnack, Organic-Split, and Cave-diving. SBD is extremely good in Floortile, Openstacks, and Woodworking. Partition-

			Explicit Search					Unf.	Sym.	Decoupled		
Domain	#	#$\mathcal{F}$	Base	PP	POR	Sym	PSy	Cunf	SBD	opt	sat	hc
Agricola	40	40	0	0	0	2	0	0	**4**	0	0	0
Childsnack	30	30	0	0	0	**6**	**6**	0	4	0	0	0
DataNetwork	40	40	1	1	0	1	0	0	**6**	1	1	0
Depots	22	22	4	4	3	**5**	4	2	3	4	**5**	**5**
Driverlog	20	20	5	5	5	6	6	3	8	8	**11**	10
Elevators	70	40	11	13	6	17	13	2	18	17	**23**	**23**
Floortile	70	70	2	2	2	2	2	0	**51**	2	2	2
Freecell	80	42	**3**	2	0	**3**	0	0	2	0	2	2
GED	40	40	10	10	10	**15**	10	7	**15**	**15**	**15**	**15**
Grid	5	5	0	0	0	0	0	0	**1**	0	0	0
Logistics	63	63	12	12	12	13	13	11	18	25	**26**	19
Miconic	150	145	45	45	40	51	43	30	107	46	**145**	79
Mprime	35	6	2	1	1	2	1	1	**4**	1	2	2
NoMystery	40	40	11	11	11	12	12	7	16	28	**30**	13
Openstacks	135	105	12	12	12	14	13	12	**58**	13	13	13
Organic-Split	40	35	8	7	6	**10**	7	1	8	5	8	8
ParcPrinter	30	20	1	1	**20**	1	**20**	1	10	3	3	3
Pathways	30	30	4	4	4	4	4	4	**5**	4	4	4
PSR	50	48	47	47	45	**48**	**48**	38	**48**	**48**	**48**	**48**
Rovers	40	40	5	5	6	5	6	6	**13**	6	6	6
Satellite	36	36	4	5	5	5	5	6	**7**	5	5	5
Scanalyzer	30	9	**6**	**6**	3	**6**	3	3	**6**	3	**6**	**6**
Tetris	37	30	4	4	2	**5**	3	1	2	4	4	4
TPP	30	29	5	4	5	6	6	4	**7**	4	4	4
Transport	117	39	19	20	18	20	18	10	20	20	**22**	**22**
Trucks	30	27	5	5	4	7	4	2	**9**	4	4	4
Woodworking	70	63	10	12	15	11	17	18	**37**	11	13	13
Zenotravel	20	20	7	7	7	7	7	4	7	7	**13**	9
Other	826	69	6	6	6	6	6	6	6	6	6	6
$\sum$	2226	1203	249	251	248	290	277	179	**500**	290	421	325
Unsolvability IPC 2016												
BagBarman	20	16	12	12	4	12	7	0	**14**	8	12	12
Cavediving	25	21	3	3	3	**7**	**7**	1	**7**	4	4	4
DocTransfer	20	20	5	5	5	5	**10**	4	5	5	5	5
NoMystery	23	23	2	2	2	2	2	1	5	11	**12**	3
PegSol	24	24	**24**	**24**	**24**	**24**	**24**	20	**24**	**24**	**24**	**24**
Rovers	19	19	6	6	5	6	5	2	**12**	9	9	9
Tetris	20	20	5	5	5	**10**	5	5	5	5	5	5
TPP	30	30	17	17	14	17	14	9	**20**	12	12	12
Other	137	22	2	2	2	2	2	2	2	2	2	2
$\sum$	318	195	76	76	64	85	76	44	**94**	80	85	76
Unsolvable Benchmarks from Hoffmann et al. [2014]												
3-SAT	30	1	**1**	**1**	**1**	**1**	**1**	0	0	**1**	**1**	**1**
NoMystery	25	25	0	0	0	0	0	0	4	**24**	**24**	2
Rovers	25	25	0	0	0	0	0	0	**5**	0	0	0
TPP	25	25	6	5	0	6	1	0	**7**	0	0	0
Other	30	1	1	1	1	1	1	1	1	1	1	1
$\sum$	135	77	8	7	2	8	3	1	17	**26**	**26**	4
Total	2679	1475	333	334	314	383	356	224	**611**	396	532	405

Figure 7.5: Same setup as in Figure 7.3, on instances where IA does not abstain.

based pruning (PP) does not beat all other planners in any domain.

In general, with these more complex factorings, the picture is somewhat mixed for decoupled search. There are still domains where it is significantly better than all other methods, e. g., NoMystery, Logistics, Miconic, but in some domains it even performs

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle\mathcal{F}_\mathrm{i}\rangle$	Sym	SBD	Coverage
Base	-	0	1	9	2	12	13	2	0	8	623
Fork	**9**	-	**9**	10	4	**13**	**15**	2	8	13	699
IFork	**6**	5	-	12	7	**14**	**14**	5	2	8	639
IA	**17**	10	**13**	-	7	11	**13**	5	8	12	686
MIS	**11**	**5**	**12**	**10**	-	**11**	**13**	0	8	12	693
LPS	**16**	11	12	11	8	-	**7**	5	11	11	684
LPG	13	8	11	11	6	5	-	5	11	11	668
$\langle\mathcal{F}_\mathrm{i}\rangle$	**16**	**9**	**13**	**13**	**5**	**14**	**15**	-	11	13	712
Sym	**33**	**28**	**28**	**29**	**30**	**34**	**33**	**27**	-	16	738
SBD	**42**	**36**	**42**	**37**	**37**	**39**	**41**	**35**	**33**	-	**966**

Figure 7.6: Pairwise comparison of factoring strategies using opt, including a simulated sequential portfolio of strategies $\langle\mathcal{F}_\mathrm{i}\rangle$, plus Sym and SBD.

worse than the explicit-state search baseline, e. g., Hiking, Pipesworld, or the unsolvable variant of TPP. That said, Base also outperforms all of the other methods in several domains, so all reduction techniques share that if the obtained reduction is not high enough, there is a certain overhead that outweighs the use of the technique. In decoupled search, we believe that being more selective in the factorings for which decoupled search is actually invoked could prevent many cases where it would be better to run explicit-state search. There are several domains where, although there are at least two mobile leaves in each factoring, (some of) these leaves do not contribute enough to the reduction. We get back to this discussion at the end of this chapter.

Figures 7.6 and 7.8 compare the factoring strategies to each other, where whenever a strategy abstains, we run Base as backup.[7] We compare all strategies to each other, a simulated portfolio combining several of them ($\langle\mathcal{F}_\mathrm{i}\rangle$), Sym, and SBD, the two strongest competitors. In Figure 7.6 we use opt for decoupled search, in Figure 7.8 we use sat.

For opt, observe that the best single factoring is Fork, the by-far weakest results are achieved by IFork. This means that, on tasks tackled by decoupled search, it performs significantly better with fork factorings compared to inverted-fork factorings, even though more instances are tackled with IFork.

[7]See Chapter 7.2 for details

			Explicit Search					Unf.	Sym.	Decoupled		
Domain	#	#$\mathcal{F}$	Base	PP	POR	Sym	PSy	Cunf	SBD	opt	sat	hc
Agricola	40	40	0	0	0	2	0	0	**4**	0	0	0
Airport	50	18	11	11	**13**	12	**13**	**13**	11	11	11	11
Barman	74	74	4	4	4	**11**	4	0	**11**	4	4	4
Childsnack	30	20	0	0	0	**6**	**6**	0	4	0	0	0
DataNetwork	40	40	1	1	0	1	0	0	**6**	1	1	0
Depots	22	20	4	4	3	**5**	4	2	3	4	**5**	**5**
Driverlog	20	20	5	5	5	6	6	3	8	9	**11**	10
Elevators	70	70	11	13	6	17	13	2	18	11	**20**	**20**
Floortile	70	70	2	2	2	2	2	0	**51**	0	2	0
Freecell	80	80	**19**	15	8	**19**	8	7	14	13	14	14
GED	40	40	10	10	10	**15**	10	7	**15**	10	**15**	**15**
Grid	5	5	0	0	0	0	0	0	**1**	0	0	0
Hiking	30	25	6	5	1	**11**	4	0	9	4	5	5
Logistics	63	63	12	12	12	13	13	11	18	25	**27**	18
Miconic	150	145	45	45	40	51	43	30	107	46	**145**	81
Mprime	35	35	2	1	1	4	1	1	**5**	1	2	2
NoMystery	40	40	11	11	11	12	12	7	16	28	**30**	13
Openstacks	135	125	12	12	12	14	13	12	**71**	17	17	17
Organic	40	5	**5**	4	4	**5**	4	4	**5**	**5**	**5**	**5**
ParcPrinter	30	27	3	2	**27**	3	**27**	2	16	4	4	4
Pathways	30	30	4	4	4	4	4	4	**5**	4	4	4
Pipesworld	100	46	13	13	12	**27**	17	4	12	10	12	12
PSR	50	50	49	49	47	**50**	**50**	40	**50**	48	49	49
Rovers	40	40	5	5	6	5	6	6	**13**	7	7	7
Satellite	36	36	4	5	5	5	5	6	**7**	5	**7**	**7**
Scanalyzer	30	9	**6**	**6**	3	**6**	3	3	**6**	3	4	4
Spider	40	6	**6**	**6**	**6**	**6**	**6**	2	**6**	**6**	**6**	**6**
Storage	30	11	0	0	0	**3**	2	0	0	0	0	0
TPP	30	29	5	4	5	6	6	4	**7**	5	5	5
Transport	117	117	19	23	18	23	18	10	**24**	13	19	23
Trucks	30	30	6	6	5	8	5	3	**10**	5	5	5
Woodworking	70	70	11	13	18	12	20	21	**42**	15	17	16
Zenotravel	20	20	7	7	7	7	7	4	7	7	**10**	**10**
Other	539	32	7	7	7	7	7	7	7	7	7	7
$\sum$	2226	1488	305	305	302	378	339	215	**589**	328	470	379
Unsolvability IPC 2016												
BagBarman	20	8	**8**	**8**	4	**8**	7	0	**8**	**8**	**8**	**8**
BagTransport	29	19	7	7	5	7	5	3	8	8	**11**	**11**
Cavediving	25	19	7	7	7	**11**	**11**	5	**11**	7	7	7
Chessboard	23	23	5	5	6	5	6	5	**11**	5	5	5
DocTransfer	20	20	5	5	5	5	**10**	4	5	5	5	5
NoMystery	23	23	2	2	2	2	2	1	5	11	**12**	3
PegSolR5	15	14	**4**	**4**	**4**	**4**	**4**	3	**4**	3	3	3
PegSol	24	24	**24**	**24**	**24**	**24**	**24**	20	**24**	20	20	20
Rovers	19	19	6	6	5	6	5	2	**12**	6	6	6
TPP	30	30	17	17	14	17	14	9	**20**	14	15	15
Other	90	20	10	10	10	10	10	10	10	10	10	10
$\sum$	318	219	95	95	86	99	98	62	**118**	97	102	93
Unsolvable Benchmarks from Hoffmann et al. [2014]												
3-SAT	30	30	15	15	**20**	15	**20**	8	10	15	15	15
Mystery	30	25	3	3	4	3	**5**	2	3	3	3	3
NoMystery	25	25	0	0	0	0	0	0	4	**24**	**24**	2
Rovers	25	25	0	0	0	0	0	0	**5**	1	1	1
TPP	25	25	6	5	0	6	1	0	**7**	1	1	1
$\sum$	135	130	24	23	24	24	26	10	29	**44**	**44**	22
Total	2679	1837	424	423	412	501	463	287	**736**	469	616	494

Figure 7.7: Same setup as in Figure 7.3, on instances where LPG does not abstain.

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle \mathcal{F}_\mathrm{i} \rangle$	Sym	SBD	Coverage
Base	-	0	1	5	1	9	9	1	0	8	623
Fork	**10**	-	**9**	6	4	10	11	2	10	16	811
IFork	**9**	8	-	9	6	11	12	2	8	12	663
IA	**19**	**12**	**14**	-	5	9	**13**	2	12	16	822
MIS	**16**	**10**	**14**	**8**	-	**9**	**12**	0	11	15	818
LPS	**19**	**13**	**14**	**10**	7	-	**6**	4	16	16	826
LPG	**17**	11	**13**	10	7	4	-	5	15	15	815
$\langle \mathcal{F}_\mathrm{i} \rangle$	**20**	**13**	**15**	**11**	**5**	**12**	**16**	-	16	19	856
Sym	**33**	**27**	**27**	**26**	**26**	**27**	**28**	**22**	-	16	738
SBD	**42**	**35**	**39**	**33**	**34**	**33**	**35**	**30**	**33**	-	**966**

Figure 7.8: Same setup as in Figure 7.6, but with sat.

For $\langle \mathcal{F}_\mathrm{i} \rangle$, the best portfolio for both opt and sat uses the following strategies: $\langle$MIS, Fork, IFork$\rangle$, where, as explained above, first MIS is executed, and decoupled search is invoked if it does not abstain, then the same is done with Fork, and finally with IFork. If none of the strategies returns a factoring, the portfolio runs Base with the remaining time.[8]

In Figure 7.9 and 7.10 we show scatter plots comparing the search space size (number of expanded states), memory consumption, and runtime of decoupled search to the other methods. We do not include PP in the comparison, since it is by far the weakest form of pruning, as we have seen in the coverage results.

For the methods that preserve optimality, i. e., Base, POR, and Sym, we compare to opt, for Cunf we compare to sat, and for SBD we compare to both opt and sat. We also compared the optimality-preserving methods to sat, where similar to the comparison to SBD in Figure 7.10, the advantage of decoupled search is more pronounced when moving to sat.

Figure 7.9 proves that decoupled search consistently expands significantly fewer states than all other methods, up to many orders of magnitude. The decoupled state representation comes at a cost though, which is visible in the memory and runtime plots. Here, we see the complementarity of the methods again. There is a certain trend that decoupled search performs quite well in terms of memory usage, though. For all explicit-state methods, all techniques have strengths in a subset of the instances, although decoupled search seems to compare well against the stubborn-sets pruning of POR, overall. Cunf is not competitive in most instances.

The comparison to SBD in Figure 7.10 shows some weird behaviour of the symbolic-search planner. We conjecture that this is due to the underlying BDD library allocating a lot of memory initially, respectively the precomputation of the symbolic transition re-

[8]We remark that one could easily improve the results by running Sym or SBD as base planner if all strategies abstain.

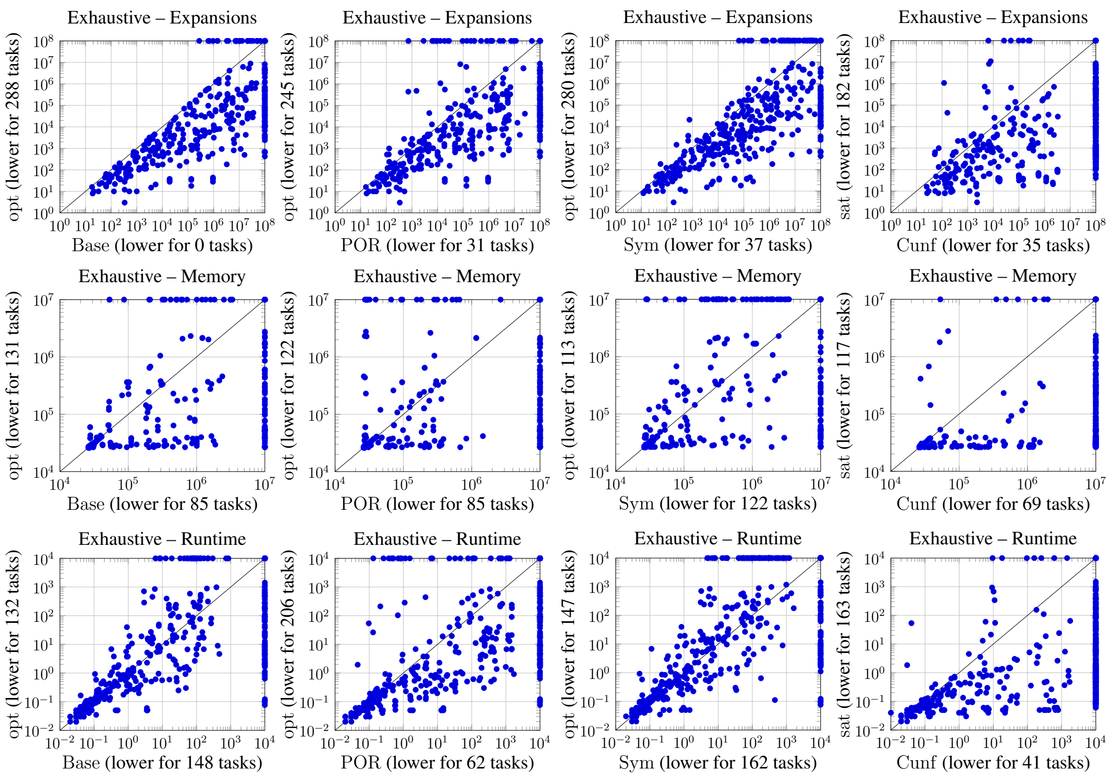


Figure 7.9: Scatter plots comparing the number of expanded states (top), memory (middle), and runtime (bottom) of opt to (from left to right) explicit-state search without pruning, with stubborn-sets pruning, and symmetry breaking. In the rightmost column, we compare sat to unfolding with Cunf. In all plots we use factorings obtained by IA.

lation. Besides this, we see that both methods can gain huge advantages over the other in many instances, where decoupled search can increase its margin when using sat.

The comparison of sat to hc (Figure 7.10, right) reveals that across all instances, hypercube pruning very rarely leads to smaller reachable decoupled state spaces. This does not come as a surprise, since we have seen in the top left of Figure 7.9 that even opt always expands fewer decoupled states than Base expands standard states. Looking at the runtime, the picture is mixed. On many instances, there is little to no overhead visible. On the other hand, the steep increase of runtimes shows that there are also many instances where hypercube pruning has to spend a lot of time to check if new decoupled states are covered by previously seen states.

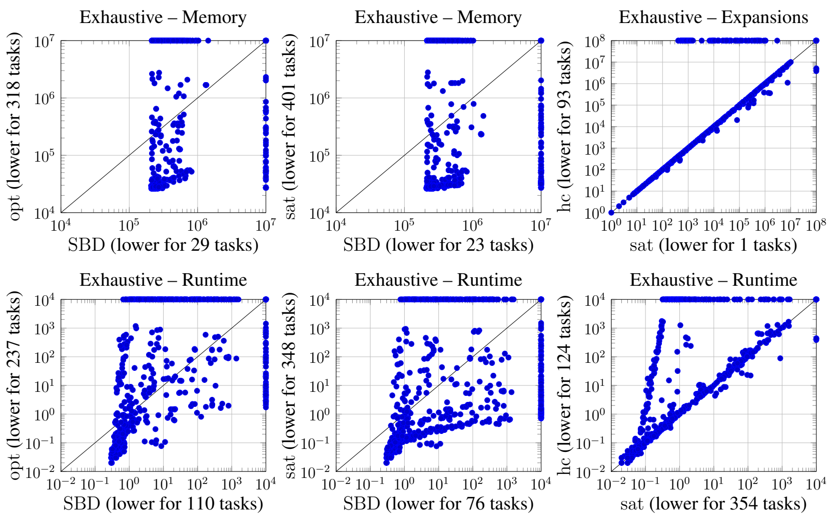


Figure 7.10: Scatter plots comparing the memory usage (top), and runtime (bottom) of opt (left) and sat (middle) to SBD. In the rightmost column, we compare sat to hc. In all plots we use factorings obtained by IA.

7.5 Satisficing Planning

In this section, we investigate the performance of decoupled search (DS) in satisficing planning. We compare decoupled search using GBFS with the h^{FF} heuristic to directed unfolding using Mole, explicit-state search without pruning (Base, abbreviated B), with PP, and with Sym. We split the evaluation into two, with and without the use of preferred-operator (PO) pruning (which is not supported for Mole). With PO, we also include the state-of-the-art planners LAMA (abbreviated LA) and BFWS (abbreviated BF). For Mole, we use the Petri-net encoding that can lead to a blow-up, which nevertheless showed better performance. We run the comparison on all satisficing-track IPC benchmarks.

Figures 7.11 and 7.12 show extremely good results for decoupled search. With the Fork and IFork factoring strategies, decoupled search beats all baseline competitors by a large margin and even shows higher coverage than the state-of-the-art planners LAMA and BFWS. For Fork without preferred operators, there are only four domains where another method has higher coverage; with PO, it is only in Pathways. In some domains, decoupled search has state-of-the-art performance even without PO. With the pruning, DS solves almost all instances tackled—427/437.

Using IFork, decoupled search without PO is only outperformed in Childsnack (by Sym), Organic-Split (by Mole), and Woodworking (by PP). With PO, there is only a single domain, where Sym and BFWS have an advantage of one instance.

With the more general factoring strategies IA and LPG shown in Figures 7.13

			GBFS with h^{FF}					+ PO Pruning					
			Unf.	Explicit			Dec.	Explicit			Dec.		
Domain	#	#$\mathcal{F}$	Mole	B	PP	Sym	DS	B	PP	Sym	DS	LAMA	BFWS
Driverlog	20	20	15	18	18	**19**	**19**	**20**	**20**	**20**	**20**	**20**	**20**
Logistics	63	63	44	51	53	57	**63**	61	61	60	**63**	**63**	62
Miconic	150	145	134	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**
NoMystery	20	20	13	8	9	9	**19**	9	9	9	**19**	11	17
Pathways	30	29	4	10	10	10	**12**	20	28	20	20	22	**29**
Rovers	40	40	16	22	**23**	**23**	22	**40**	**40**	**40**	**40**	**40**	**40**
Satellite	36	36	12	25	24	**32**	26	**36**	27	**36**	**36**	**36**	31
TPP	30	27	4	22	21	**24**	23	**27**	**27**	**27**	**27**	**27**	**27**
Woodworking	40	34	6	28	**33**	28	30	**34**	**34**	**34**	**34**	**34**	18
Zenotravel	20	20	16	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Other	1237	3	3	3	3	3	3	3	3	3	3	3	3
Total	1686	437	267	352	359	370	**382**	415	414	414	**427**	421	412

Figure 7.11: Coverage data, i. e., the number of solved tasks on instances where Fork does not abstain. #$\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage (separately for search with vs. without PO) in **bold face**.

			GBFS with h^{FF}					+ PO Pruning					
			Unf.	Explicit			Dec.	Explicit			Dec.		
Domain	#	#$\mathcal{F}$	Mole	B	PP	Sym	DS	B	PP	Sym	DS	LAMA	BFWS
Childsnack	20	20	0	0	0	**7**	2	6	3	17	**20**	6	8
Depots	22	22	12	15	14	18	**19**	19	18	**22**	21	20	**22**
Elevators	40	40	12	39	37	39	**40**	39	38	39	**40**	**40**	**40**
Floortile	40	40	4	8	8	8	**9**	8	**9**	**9**	**9**	8	5
Logistics	63	63	44	51	53	57	**63**	61	61	60	**63**	**63**	62
Mystery	30	4	**1**	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Organic-Split	20	2	**1**	0	0	0	0	0	0	0	0	**1**	**1**
Rovers	40	38	14	20	21	21	**38**	**38**	**38**	**38**	**38**	**38**	**38**
Satellite	36	34	10	23	22	30	**34**	**34**	25	**34**	**34**	**34**	29
TPP	30	26	2	19	18	21	**26**	**26**	**26**	**26**	**26**	**26**	25
Transport	58	58	10	17	16	20	**58**	41	35	40	**58**	52	**58**
Woodworking	40	38	7	32	**37**	32	35	**38**	**38**	**38**	**38**	**38**	22
Zenotravel	20	18	14	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
Other	1227	0	0	0	0	0	0	0	0	0	0	0	0
Total	1686	403	131	242	244	271	**343**	329	310	342	**366**	345	329

Figure 7.12: Same setup as in Figure 7.11, on instances where IFork does not abstain.

			GBFS with h^{FF}						+ PO Pruning						
			Unf.	Explicit			Dec.		Explicit			Dec.			
Domain	#	#$\mathcal{F}$	Mole	B	PP	Sym	DS	dup	B	PP	Sym	DS	dup	LA	BF
Agricola	20	20	0	**10**	0	8	**10**	**10**	**12**	1	10	**12**	**12**	**12**	10
Barman	40	2	0	0	0	**1**	0	0	0	0	**2**	0	0	**2**	**2**
Childsnack	20	20	0	0	0	**7**	2	2	6	3	17	**20**	**20**	6	8
DataNetwork	20	20	3	**7**	4	**7**	6	6	10	10	10	7	5	**13**	9
Depots	22	22	12	15	14	18	**19**	**19**	19	18	**22**	21	21	20	**22**
Driverlog	20	20	15	18	18	**19**	**19**	**19**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Elevators	40	10	9	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
Floortile	40	40	4	**8**	**8**	**8**	5	3	8	**9**	**9**	6	3	8	5
Freecell	80	42	30	**42**	41	**42**	41	41	**42**	41	**42**	**42**	**42**	**42**	**42**
GED	20	20	0	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Grid	5	5	3	**4**	**4**	**4**	3	3	4	**5**	4	**5**	**5**	**5**	**5**
Logistics	63	63	44	51	53	57	**63**	**63**	61	61	60	**63**	**63**	**63**	62
Maintenance	20	9	**1**	**1**	**1**	**1**	**1**	**1**	2	1	2	2	2	3	**8**
Miconic	150	145	134	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**
NoMystery	20	20	13	8	9	9	**19**	**19**	9	9	9	**19**	**19**	11	17
Openstacks	90	60	9	**60**	34	**60**	**60**	**60**	**60**	34	**60**	**60**	**60**	**60**	**60**
Organic-Split	20	19	7	**12**	6	**12**	11	10	11	6	**12**	11	11	**12**	**12**
ParcPrinter	30	20	6	**20**	19	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	19
Pathways	30	30	5	11	11	11	**14**	**14**	21	29	21	21	21	23	**30**
PSR	50	48	46	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**
Rovers	40	40	16	22	**23**	**23**	21	20	**40**	**40**	**40**	**40**	**40**	**40**	**40**
Satellite	36	36	12	25	24	**32**	26	26	**36**	27	**36**	33	33	**36**	31
Scanalyzer	30	9	7	**9**	6	**9**	**9**	**9**	**9**	6	**9**	**9**	**9**	**9**	**9**
Tetris	20	17	1	4	1	**5**	4	4	9	9	10	11	10	**14**	13
Thoughtful	20	13	**6**	5	5	5	5	5	7	**12**	7	6	6	11	**12**
Tidybot	20	20	3	**16**	12	**16**	15	15	16	13	16	15	15	17	**18**
TPP	30	29	4	22	21	24	**26**	**26**	**29**	**29**	**29**	25	24	**29**	28
Transport	58	9	8	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
Trucks	30	27	9	13	**16**	15	13	13	16	**18**	17	15	15	16	14
Woodworking	40	37	8	31	**36**	31	33	33	**37**	**37**	**37**	**37**	**37**	**37**	21
Zenotravel	20	20	16	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Other	542	7	7	7	7	7	7	7	7	7	7	7	7	7	7
Total	1686	899	438	673	625	703	**704**	700	763	717	780	779	772	**788**	776

Figure 7.13: Same setup as in Figure 7.11, on instances where IA does not abstain.

and 7.14, on top of DS we also include dup in the comparison, which does exact duplicate checking instead of dominance pruning. Duplicate checking shows the same coverage across most domains, but only improves in a single domain, namely Woodworking with LPG. On the other hand, coverage is significantly reduced in Floortile,

			GBFS with h^{FF}						+ PO Pruning						
			Unf.	Explicit			Dec.		Explicit			Dec.			
Domain	#	#$\mathcal{F}$	Mole	B	PP	Sym	DS	dup	B	PP	Sym	DS	dup	LA	BF
Agricola	20	20	0	**10**	0	8	**10**	**10**	**12**	1	10	**12**	**12**	**12**	10
Airport	50	18	**18**	17	17	**18**	**18**	17	**18**	**18**	**18**	**18**	**18**	**18**	**18**
Barman	40	40	0	17	30	**38**	3	2	24	33	**40**	19	16	**40**	**40**
Childsnack	20	10	0	0	0	**7**	0	0	6	3	**10**	0	0	5	7
DataNetwork	20	20	3	**7**	4	**7**	1	1	10	10	10	7	5	**13**	9
Depots	22	20	11	14	13	17	**18**	**18**	17	17	**20**	19	19	19	**20**
Driverlog	20	20	15	18	18	**19**	**19**	**19**	**20**	**20**	**20**	19	19	**20**	**20**
Elevators	40	40	12	39	37	39	**40**	**40**	39	38	39	**40**	**40**	**40**	**40**
Floortile	40	40	4	8	8	8	**31**	22	8	9	9	**35**	26	8	5
Freecell	80	80	66	**80**	79	**80**	**80**	**80**	**80**	79	**80**	78	78	79	**80**
GED	20	20	0	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Grid	5	5	3	**4**	**4**	**4**	3	2	4	**5**	4	4	3	**5**	**5**
Hiking	20	18	0	17	14	**18**	**18**	**18**	**18**	17	**18**	**18**	**18**	**18**	8
Logistics	63	63	44	51	53	57	**63**	**63**	61	61	60	**63**	**63**	**63**	62
Maintenance	20	20	1	6	4	6	**13**	**13**	9	10	9	13	13	11	**17**
Miconic	150	145	134	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**
Mprime	35	35	25	31	30	**34**	28	28	**35**	34	**35**	**35**	**35**	**35**	**35**
Mystery	30	25	**17**	16	16	16	15	15	14	14	15	15	15	**17**	**17**
NoMystery	20	20	13	8	9	9	**19**	18	9	9	9	**19**	**19**	11	17
Openstacks	90	80	9	**80**	54	**80**	40	40	**80**	54	**80**	7	7	**80**	**80**
Organic	20	1	0	**1**	0	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**
ParcPrinter	30	27	9	**27**	23	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	26
Pathways	30	30	5	11	11	11	**16**	**16**	21	29	21	18	18	23	**30**
Pipesworld	100	46	35	36	35	**44**	37	36	44	44	45	45	45	**46**	**46**
PSR	50	50	48	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
Rovers	40	40	16	22	**23**	**23**	22	22	**40**	**40**	**40**	**40**	**40**	**40**	**40**
Satellite	36	36	12	25	24	32	**36**	**36**	**36**	27	**36**	**36**	**36**	**36**	31
Scanalyzer	30	9	7	**9**	6	**9**	8	8	**9**	6	**9**	**9**	**9**	**9**	**9**
Storage	30	11	2	6	5	**8**	7	7	7	7	8	8	8	6	**11**
TPP	30	29	4	22	21	**24**	21	20	**29**	**29**	**29**	**29**	28	**29**	28
Transport	58	58	10	17	16	20	**58**	**58**	41	35	40	**58**	**58**	52	**58**
Trucks	30	30	10	14	**17**	16	14	14	17	**20**	18	16	15	17	15
Woodworking	40	40	8	34	39	34	38	**40**	**40**	**40**	**40**	38	**40**	**40**	24
Zenotravel	20	20	16	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Other	337	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Total	1686	1171	562	887	850	**954**	944	931	1016	976	1040	986	971	**1060**	1049

Figure 7.14: Same setup as in Figure 7.11, on instances where LPG does not abstain.

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle\mathcal{F}_\mathrm{i}\rangle$	Sym	Coverage
Base	-	0	0	7	3	7	8	0	1	1187
Fork	**7**	-	3	**7**	**6**	8	8	0	5	1217
IFork	**11**	**10**	-	**13**	**13**	**11**	10	0	11	1288
IA	**9**	4	3	-	**7**	7	8	1	7	1218
MIS	**5**	1	2	5	-	6	7	0	4	1211
LPS	**12**	**9**	6	**11**	**11**	-	**7**	4	10	1247
LPG	**15**	**12**	10	**15**	**13**	6	-	8	11	1244
$\langle\mathcal{F}_\mathrm{i}\rangle$	**14**	**10**	**3**	**13**	**14**	**11**	**10**	-	**13**	**1302**
Sym	**20**	**17**	**14**	**21**	**20**	**17**	**16**	12	-	1285

Figure 7.15: Pairwise comparison of factoring strategies without PO pruning. We include a simulated sequential portfolio of strategies $\langle\mathcal{F}_\mathrm{i}\rangle$, and Sym.

and somewhat in DataNetwork with PO. We conclude that in satisficing planning the advantage of the more efficient duplicate check via hashing does not lead to a big reduction in overall search time, which is confirmed by the plots on the right of Figure 7.18.

Similar to the previous section, we see that the advantage of decoupled search to the other techniques is less pronounced with more general factorings, since we now include instances in the benchmark set where the cross-factor dependencies are more complex. The results are particularly bad in Barman (-35 without PO), and Openstacks (-73 with PO) when using LPG. These domains aside, we see the known picture where every method has its advantages in some domains, and decoupled search is competitive with all other planners.

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle\mathcal{F}_\mathrm{i}\rangle$	Sym	LAMA	BFWS	Coverage
Base	-	0	0	**7**	2	**10**	9	0	3	4	12	1364
Fork	**2**	-	1	**7**	2	**10**	**9**	0	4	5	14	1376
IFork	**6**	**5**	-	**9**	**7**	**11**	**9**	0	5	8	14	1401
IA	6	4	3	-	4	9	9	2	7	7	13	1380
MIS	**5**	**3**	4	**8**	-	**11**	**11**	3	7	5	13	1376
LPS	6	4	2	9	5	-	5	1	6	7	12	1356
LPG	**10**	8	6	**11**	9	**8**	-	5	7	8	12	1334
$\langle\mathcal{F}_\mathrm{i}\rangle$	**7**	**5**	**1**	**9**	**7**	**11**	**9**	-	6	9	15	1411
Sym	**13**	**13**	**11**	**16**	**15**	**19**	**16**	**11**	-	8	12	1406
LAMA	**20**	**18**	**16**	**20**	**17**	**21**	**20**	**15**	**17**	-	**15**	1463
BFWS	**23**	**21**	**19**	**21**	**20**	**22**	**23**	**18**	**18**	13	-	**1477**

Figure 7.16: Same setup as in Figure 7.15, but with PO, and with LAMA and BFWS.

In Figure 7.15 and 7.16 we compare the factorings strategies again, without the use of preferred operators, respectively with PO. In both cases, IFork clearly outperforms

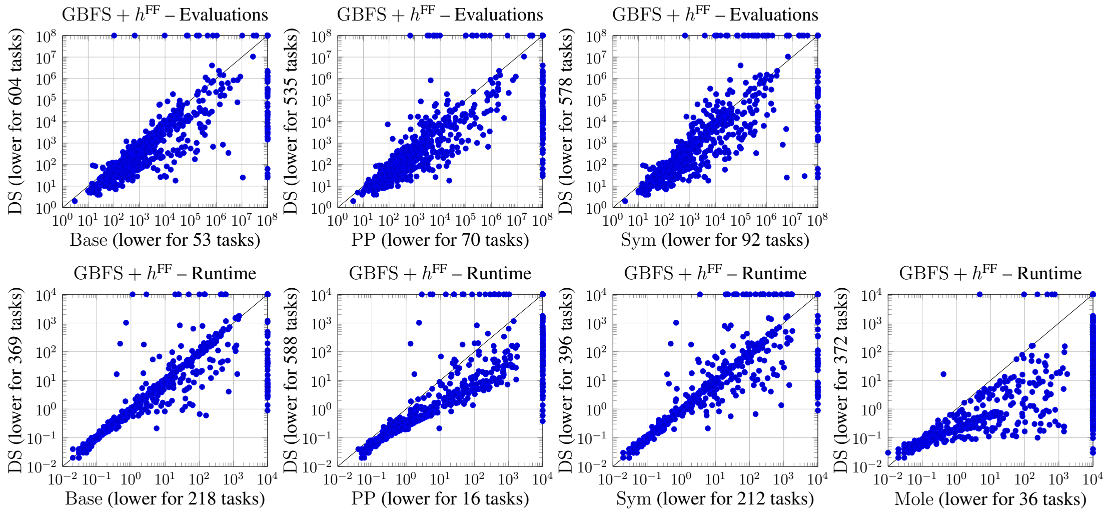


Figure 7.17: Scatter plots comparing the number of evaluated states (top), and runtime (bottom) of DS to (from left to right) explicit-state search without pruning, with partition-based pruning, and symmetry breaking. In the rightmost column, we compare the runtime of DS to unfolding with Mole. In all plots we use factorings obtained by IA and no preferred-operator pruning.

all other factoring strategies. The factoring portfolio ($\langle\mathcal{F}_\mathrm{i}\rangle$) accordingly first uses IFork, then Fork, and falls back to Base if none of the two produce a factoring. This indicates that in satisficing planning, simple factorings are by far better than structurally more complex strict-star or general-star factorings.

We conclude this section by looking at the search space size (number of state evaluations) and runtime, see Figure 7.17 and 7.18. In most of the instances, DS evaluates fewer states than Base, PP, and Sym. Regarding runtime, decoupled search also clearly outperforms Base, PP, and Mole. Compared to Sym, we see many instances where decoupled search is slower. As mentioned before, this kind of complementarity is expected given the different sources of state-space reduction.

The comparison to LAMA shows that decoupled search evaluates fewer states, when search spaces are small for both methods, but that there are also instances where LAMA requires significantly fewer evaluations. This is due to the landmark heuristic it uses, which could probably improve the search guidance for decoupled search, too. Regarding runtime, although it seems that there are many instances where DS has a small advantage, on instances where the runtimes differ significantly, more often LAMA is in front. For BFWS, we clearly see that the planner is based on a different, faster, translation from PDDL causing lower minimal runtimes. Since its approach is quite different from GBFS, we also see that it is very fast in solving many instances, resulting in a clear advantage over decoupled search. Nevertheless, there are also instances where

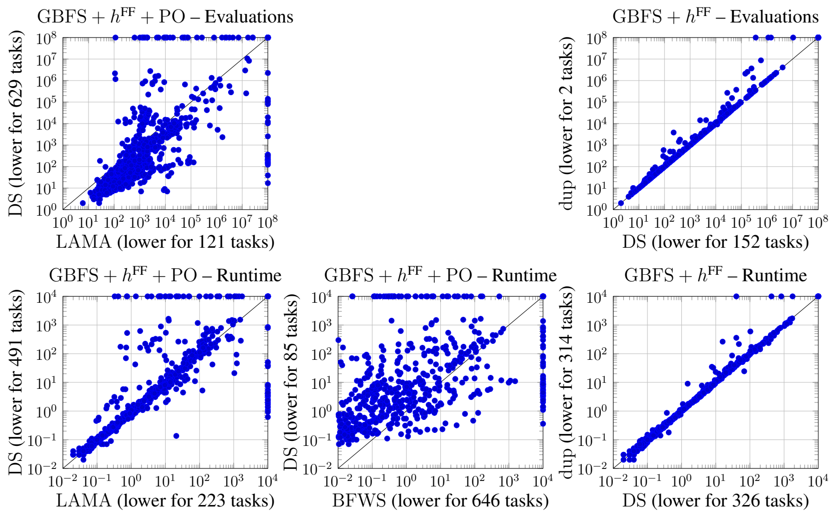


Figure 7.18: Scatter plots comparing the number of evaluated states (top), and runtime (bottom) of DS with PO to LAMA (left), and BFWS (middle; only runtime). In the rightmost column, we compare DS to dup. In all plots we use factorings from IA.

decoupled search is faster by up to two orders of magnitude.

Finally, in the right of Figure 7.18 we compare decoupled search with dominance pruning to exact duplicate checking. As already indicated by the coverage results, the more efficient checking rarely pays off in terms of overall runtime. On the positive side, there are also only few instances where the search space grows, i. e., where the weaker pruning has an effect.

			Blind Search								$h^{\max}$	A* with $h^{\text{LM-cut}}$								
			Unf.	Explicit Search					Dec.		Unf.	Explicit Search					Dec.			
Domain	#	#$\mathcal{F}$	Mole	B	PP	POR	Sym	PSy	aT	DS	Mole	B	PP	POR	Sym	PSy	aT	DS	SBD	C2
Driverlog	20	20	4	7	7	7	7	7	**11**	**11**	7	13	14	13	13	13	13	13	12	**15**
Logistics	63	63	11	12	14	12	14	14	**26**	**26**	13	26	26	27	26	28	33	**34**	24	28
Miconic	150	145	25	45	45	40	**51**	43	46	46	20	136	136	136	**137**	136	135	135	107	98
NoMystery	20	20	4	8	8	8	8	8	**20**	**20**	7	14	14	14	15	14	**20**	**20**	14	**20**
Pathways	30	29	2	**3**	**3**	**3**	**3**	**3**	**3**	**3**	3	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Rovers	40	40	2	5	6	**7**	5	**7**	6	6	5	7	10	9	7	9	9	9	**14**	13
Satellite	36	36	**7**	5	**7**	6	6	6	6	6	5	7	12	11	13	**14**	8	7	8	9
TPP	30	27	3	5	5	5	6	6	**23**	**23**	3	5	6	5	7	6	**23**	18	7	14
Woodworking	30	13	**7**	4	4	6	4	**7**	5	5	6	6	6	**11**	7	**11**	**11**	10	9	9
Zenotravel	20	20	7	8	8	7	8	7	**12**	**12**	7	**13**	**13**	**13**	**13**	**13**	12	**13**	10	**13**
Other	1191	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Total	1630	417	76	106	111	105	116	112	**162**	**162**	80	235	245	247	246	252	**272**	267	213	227

Figure 7.19: Coverage data, i. e., the number of solved tasks on instances where Fork does not abstain. #$\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage (separately for blind search and A* with $h^{\text{LM-cut}}$) in **bold face**.

7.6 Optimal Planning

For optimal planning, an effective pruning of the search space is most crucial, since the typical approach of using A* with an admissible heuristic often has to explore a large number of states before finding the optimal solution. In this section, we compare decoupled search to the baseline techniques with blind search and A* search with the $h^{\text{LM-cut}}$ heuristic. When using $h^{\text{LM-cut}}$, we also compare to the state-of-the-art planners SBD and C2. For Mole, only blind search and $h^{\max}$ are supported. For blind search, Mole performed best with the exponential net encoding, with $h^{\max}$ we use the alternative encoding based on TNF. We include all domains and instances from the optimal tracks of the IPCs.

For decoupled search, we show results for the standard configuration DS, and for some evaluations we show the Anytime D-A*algorithm aT, exact duplicate checking dup, and a simplified dominance pruning without g-value adaptation sd.

We start again by discussing the results obtained with Fork and IFork. With fork factorings, decoupled search clearly outperforms all other planners. There are only few domains, e. g., Miconic for blind search, where other planners perform better. The specialized anytime algorithm performs just like DS for blind search, where there is little difference between the two approaches. With $h^{\text{LM-cut}}$, aT has an advantage in TPP, and loses an instance in each of Logistics, and Zenotravel. The results in TPP are remarkable. Here, decoupled search beats SBD and C2 even without heuristic.

With inverted-fork factorings, the performance decoupled search degrades dramatically. We only show results with $h^{\text{LM-cut}}$; with blind search, the results are even worse. DS still beats explicit-state search without pruning, but overall performs worse than most other methods—it is equal to PP overall, and better than Mole. Similar to the

			Unf.	Explicit Search					Decoupled			
Domain	#	#$\mathcal{F}$	Mole	Base	PP	POR	Sym	PSy	aT	DS	SBD	C2
Childsnack	20	20	0	0	0	0	**6**	**6**	0	0	4	0
Depots	22	22	2	7	7	7	**8**	**8**	7	7	5	7
Elevators	30	30	9	22	23	22	22	22	16	23	**25**	**25**
Floortile	40	40	2	13	13	13	16	16	11	12	**34**	28
Logistics	63	63	13	26	26	27	26	**28**	22	27	24	**28**
Mystery	30	4	0	0	0	0	0	**1**	0	0	0	**1**
Organic-Split	20	3	1	1	1	1	**2**	**2**	0	0	0	1
Rovers	40	38	3	5	8	7	5	7	5	6	**12**	11
Satellite	36	34	3	5	10	9	11	**12**	5	10	6	7
TPP	30	26	1	2	3	2	4	3	2	2	4	**11**
Transport	59	59	14	17	17	17	18	18	16	17	**24**	23
Woodworking	30	24	9	12	13	**22**	14	**22**	13	17	17	15
Zenotravel	20	18	5	**11**	**11**	**11**	**11**	**11**	8	**11**	8	**11**
Other	1190	1	1	1	1	1	1	1	1	1	1	1
Total	1630	382	63	122	133	139	144	157	106	133	164	**169**

Figure 7.20: Same setup as in Figure 7.19, on instances where IFork does not abstain. We use A^* with $h^{\text{LM-cut}}$ for explicit-state and decoupled search; Mole uses $h^{\max}$.

results in Chapter 7.4, it seems like inverted-fork factorings are not a good choice when large search spaces need to be explored exhaustively.

The anytime algorithm performs significantly worse on inverted-fork factorings. We conjecture that this is because it is crucial to take the leaf-prices into account, here, since these provide necessary preconditions for center actions.

The tables in Figure 7.21 and 7.22 show results for IA and LPG. With blind search and IA, decoupled search consistently beats all competitors except Sym. The results are particularly good in Driverlog, Logistics, NoMystery, and Tidybot. Sym performs well in Openstacks, where we have seen before that decoupled search does not give good performance. With $h^{\text{LM-cut}}$, the differences between planners become smaller. Here, PSy is clearly better than decoupled search overall, being competitive with C2, even.

Exact duplicate checking performs well with blind search in Miconic, and improves moderately over DS in three other domains. With $h^{\text{LM-cut}}$, it performs similar to DS. The anytime search variant again shows bad results for the same reason as discussed above. We also see the effect of the augmented-cost dominance pruning and the g-value adaptation when comparing sd with DS. While in most domains there is little difference, coverage drops by 12 instances in Elevators.

When using LPG, the performance of decoupled search drops in comparison to IA. With blind search, total coverage is similar to Base and PP. Interestingly, stubborn-sets pruning is also ineffective on this set of instances. With $h^{\text{LM-cut}}$, DS overall beats Base

			Blind Search								$h^{\max}$	A* with $h^{\text{LM-cut}}$										
			Unf.	Explicit Search					Dec.		Unf.	Explicit Search					Decoupled					
Domain	#	#$\mathcal{F}$	Mo	B	PP	POR	Sym	PSy	DS	dup	Mo	B	PP	POR	Sym	PSy	aT	DS	dup	sd	SBD	C2
Agricola	20	20	0	0	0	0	**7**	0	0	0	0	0	0	0	0	0	0	0	0	0	**5**	0
Childsnack	20	20	0	0	0	0	**6**	**6**	0	0	0	0	0	0	**6**	**6**	0	0	0	0	4	0
DataNet	20	20	5	7	6	5	7	5	**9**	5	9	12	12	12	12	12	**14**	**14**	12	**14**	13	13
Depots	22	22	2	4	4	4	**6**	5	4	4	2	7	7	7	**8**	**8**	7	7	7	7	5	7
Driverlog	20	20	4	7	7	7	7	7	**11**	**11**	7	13	14	13	13	13	12	13	13	13	12	**15**
Elevators	30	30	9	13	**16**	12	15	13	**16**	10	9	22	23	22	22	22	16	23	22	11	**25**	**25**
Floortile	40	40	0	**2**	**2**	**2**	**2**	**2**	**2**	0	2	13	13	13	16	16	9	9	6	9	**34**	28
Freecell	80	42	0	**3**	2	0	**3**	0	0	2	0	2	2	1	2	1	2	2	2	1	2	**3**
GED	20	20	10	**15**	**15**	**15**	**15**	**15**	**15**	**15**	10	15	15	15	15	15	15	15	15	15	19	**20**
Grid	5	5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	1	2	2	2	2	2	2	2	2	2	2	**3**
Logistics	63	63	11	12	14	12	14	14	**25**	24	13	26	26	27	26	28	33	**36**	35	34	24	28
Miconic	150	145	25	45	45	40	51	43	46	**62**	20	136	136	136	**137**	136	135	135	136	135	107	98
Mprime	35	6	3	**6**	**6**	**6**	**6**	**6**	4	4	**6**	**6**	**6**	**6**	**6**	**6**	4	4	4	4	**6**	**6**
NoMystery	20	20	4	8	8	8	8	8	**20**	17	7	14	14	14	15	14	**20**	**20**	19	**20**	14	**20**
Openstacks	80	50	8	25	23	22	**30**	26	20	22	8	24	23	22	29	26	20	20	22	20	**50**	40
Org-Split	20	16	2	6	5	4	**7**	5	3	3	6	11	8	9	**15**	11	8	9	9	9	5	5
ParcPrinter	20	13	1	1	1	**13**	1	**13**	3	3	3	4	5	**13**	4	**13**	7	7	7	7	2	4
Pathways	30	30	3	**4**	**4**	**4**	**4**	**4**	**4**	**4**	4	**5**	**5**	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**	**5**
PSR	50	48	42	47	47	47	**48**	**48**	**48**	47	41	47	47	47	**48**	**48**	**48**	**48**	47	**48**	**48**	**48**
Rovers	40	40	2	5	6	**7**	5	**7**	**7**	**7**	5	7	10	9	7	9	8	8	8	8	**14**	13
Satellite	36	36	**7**	5	**7**	6	6	6	5	5	5	7	12	11	13	**14**	7	9	8	7	8	9
Scanalyzer	30	9	3	**6**	**6**	3	**6**	3	3	3	3	5	5	3	5	3	3	5	5	5	**6**	**6**
Tetris	17	13	2	4	5	4	**7**	5	5	4	2	5	5	4	7	5	5	4	4	4	2	**10**
Tidybot	30	30	3	14	10	5	14	5	**16**	**16**	3	18	18	17	**19**	17	18	17	18	18	7	18
TPP	30	29	3	5	5	5	**6**	**6**	5	5	3	5	6	5	7	6	5	5	5	5	7	**14**
Transport	59	30	12	**15**	**15**	13	**15**	13	**15**	13	13	14	14	14	14	14	14	14	14	12	**16**	**16**
Trucks	30	27	1	5	5	4	**7**	4	4	6	2	9	9	9	11	9	10	10	10	10	10	**12**
Woodwork	30	26	10	6	7	10	7	**13**	7	7	10	14	15	**23**	16	**23**	14	17	17	16	19	17
Zenotravel	20	20	7	8	8	7	8	7	**9**	8	7	**13**	**13**	**13**	**13**	**13**	11	**13**	**13**	12	10	**13**
Other	563	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
Total	1630	896	186	285	286	272	**325**	296	313	314	207	462	471	478	499	501	457	477	471	457	487	**502**

Figure 7.21: Same setup as in Figure 7.19, on instances where IA does not abstain.

and PP, and performs similar to POR. Like in all evaluations, so far, all methods have their individual strong and weak domains.

The factoring comparisons in Figure 7.23 and 7.24 confirm that Fork generates the most-suited factorings for optimal planning. For both blind search and A* with $h^{\text{LM-cut}}$, fork factorings give the best results. Inverted-fork factorings are bad in both cases, and LPG shows bad performance with blind search. The $\langle \mathcal{F}_\text{i} \rangle$ portfolio selects $\langle \text{Fork}, \text{IFork} \rangle$ for blind search and $\langle \text{MIS}, \text{Fork}, \text{LPG} \rangle$ for A* with $h^{\text{LM-cut}}$. The comparatively "bad" strategies are probably chosen because they still improve over Base in the instances not tackled by Fork, respectively MIS and Fork. With $h^{\text{LM-cut}}$, LPG for example solves 7 more instances in ParcPrinter compared to Base.

The scatter plots in Figure 7.25 nicely illustrate that decoupled search generates fewer states compared to all other pruning techniques. Still, this reduction does not always pay off. While there is a clear advantage in runtime compared to Base, PP, and Sym, POR can also gain significant speed-ups over decoupled search. The runtime comparison in Figure 7.26 clearly favors decoupled search over unfolding with

			Blind Search								$h^{\max}$	A* with $h^{\text{LM-cut}}$									
			Unf.	Explicit Search					Dec.		Unf.	Explicit Search					Decoupled				
Domain	#	#$\mathcal{F}$	Mo	B	PP	POR	Sym	PSy	DS	dup	Mo	B	PP	POR	Sym	PSy	DS	dup	sd	SBD	C2
Agricola	20	20	0	0	0	0	**7**	0	0	0	0	0	0	0	0	0	0	0	0	**5**	0
Airport	50	18	**13**	12	**13**	**13**	**13**	**13**	11	11	16	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	11	15
Barman	34	34	0	4	4	4	**11**	4	4	0	0	4	4	4	**11**	**11**	4	4	1	**11**	9
Childsnack	20	20	0	0	0	0	**6**	**6**	0	0	0	0	0	0	**6**	**6**	0	0	0	4	0
DataNetwork	20	20	5	**7**	6	5	**7**	5	6	4	9	12	12	12	12	12	**13**	12	8	**13**	**13**
Depots	22	20	2	4	4	4	**6**	5	4	3	2	7	7	7	**8**	**8**	7	7	7	5	7
Driverlog	20	20	4	7	7	7	7	7	**11**	9	7	13	14	13	13	13	14	14	12	12	**15**
Elevators	30	30	9	13	**16**	12	15	13	14	9	9	22	23	22	22	22	23	21	11	**25**	**25**
Floortile	40	40	0	**2**	**2**	**2**	**2**	**2**	**2**	**2**	2	13	13	13	16	16	2	2	2	**34**	28
Freecell	80	80	7	**20**	16	8	**20**	8	14	14	7	15	15	14	15	14	15	15	15	14	**22**
GED	20	20	10	**15**	**15**	**15**	**15**	**15**	13	13	10	15	15	15	15	15	13	15	13	19	**20**
Grid	5	5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	1	2	2	2	2	2	2	2	2	2	**3**
Hiking	20	15	1	6	6	2	**12**	5	6	4	1	4	4	3	8	6	5	4	3	10	**14**
Logistics	63	63	11	12	14	12	14	14	**26**	**26**	13	26	26	27	26	28	**34**	**34**	**34**	24	28
Miconic	150	145	25	45	45	40	51	43	46	**59**	20	136	136	136	**137**	136	135	136	135	107	98
Mprime	35	35	8	19	19	14	**20**	16	14	14	18	22	22	22	23	23	22	22	22	24	**25**
Mystery	30	25	8	**13**	**13**	10	**13**	10	11	10	14	15	15	15	15	**16**	15	15	15	13	14
NoMystery	20	20	4	8	8	8	8	8	**20**	17	7	14	14	14	15	14	**20**	19	**20**	14	**20**
Openstacks	80	70	8	25	23	22	**30**	26	17	11	8	24	23	22	29	26	20	11	18	**63**	40
Organic	20	4	1	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
ParcPrinter	20	17	2	4	4	**17**	4	**17**	5	5	6	7	8	**17**	7	**17**	14	14	14	5	7
Pathways	30	30	3	**4**	**4**	**4**	**4**	**4**	**4**	**4**	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Pipesworld	100	46	11	24	23	19	**32**	25	19	18	14	25	25	24	31	30	24	23	22	23	**33**
PSR	50	50	44	49	49	49	**50**	**50**	49	48	43	49	49	49	**50**	**50**	49	49	49	**50**	**50**
Rovers	40	40	2	5	6	**7**	5	**7**	**7**	**7**	5	7	10	9	7	9	9	9	8	**14**	13
Satellite	36	36	**7**	5	**7**	6	6	6	**7**	6	5	7	12	11	13	**14**	12	12	6	8	9
Scanalyzer	30	9	3	**6**	**6**	3	**6**	3	3	3	3	5	5	3	5	3	5	5	5	**6**	**6**
Spider	20	6	4	**6**	**6**	**6**	**6**	**6**	**6**	**6**	5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
Storage	30	11	0	1	1	1	**3**	**3**	0	0	0	2	2	2	**4**	**4**	2	2	2	1	2
TPP	30	29	3	5	5	5	**6**	**6**	5	4	3	5	6	5	7	6	6	5	5	7	**14**
Transport	59	59	12	**18**	**18**	15	**18**	15	14	8	14	17	17	17	18	18	17	16	12	**24**	23
Trucks	30	30	2	6	6	5	**8**	5	5	7	3	10	10	10	12	10	11	11	11	11	**13**
Woodworking	30	30	13	7	8	13	8	**16**	10	9	13	17	18	**27**	19	**27**	22	21	18	22	20
Zenotravel	20	20	7	**8**	**8**	7	**8**	7	**8**	**8**	7	**13**	**13**	**13**	**13**	**13**	**13**	**13**	11	10	**13**
Other	326	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
Total	1630	1124	237	372	374	347	**433**	382	373	351	280	547	559	567	598	608	567	552	520	613	**621**

Figure 7.22: Same setup as in Figure 7.19, on instances where LPG does not abstain.

Mole. Compared to symbolic search (SBD), while we see again the initial overhead when using BDDs, the methods are obviously complementary. C2 shows some interesting behaviour. This planner is based on abstraction heuristics and spends a significant amount of time on precomputing the heuristic. This is visible at the vertical line around 800s, where this precomputation has its timeout.

In Figure 7.27, we finally compare different variants of decoupled search to each other. In the left, we see that exact duplicate checking can cause a moderate to high increase in the number of expanded states, which then translates to an increase in runtime. Still, when there is little effect on the search space size, the more efficient hashing for duplicate checking clearly pays off, leading to a speed-up of up to two orders of magnitude. The effect of augmented-cost dominance ($\preceq_{\text{aug}}$) and the g-value adaptation is even more pronounced (middle plots with A* and $h^{\text{LM-cut}}$). Here, while most instances

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle \mathcal{F}_\mathrm{i} \rangle$	Sym	Coverage
Base	-	0	1	6	3	7	**12**	1	0	572
Fork	**9**	-	**7**	**10**	**7**	**9**	**14**	1	6	628
IFork	**5**	2	-	8	4	7	**13**	0	3	579
IA	**13**	7	**11**	-	5	**11**	**16**	6	9	600
MIS	**10**	5	**9**	**6**	-	**9**	**13**	4	7	605
LPS	**8**	4	**8**	4	4	-	**8**	4	7	588
LPG	9	5	9	4	4	1	-	5	7	573
$\langle \mathcal{F}_\mathrm{i} \rangle$	**10**	**2**	**6**	**11**	**7**	**9**	**15**	-	8	631
Sym	**25**	**21**	**22**	**22**	**22**	**22**	**27**	**20**	-	**660**

Figure 7.23: Pairwise comparison of factoring strategies when running blind search. We include a simulated sequential portfolio of strategies $\langle \mathcal{F}_\mathrm{i} \rangle$, and Sym.

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle \mathcal{F}_\mathrm{i} \rangle$	SBD	PSy	C2	Coverage
Base	-	1	2	7	4	5	5	5	19	4	8	790
Fork	**5**	-	**6**	**9**	5	5	5	4	21	7	11	822
IFork	**5**	4	-	**9**	7	6	5	6	20	5	10	801
IA	**10**	7	6	-	6	6	6	4	**22**	8	11	805
MIS	**7**	5	7	6	-	6	7	1	**21**	8	10	805
LPS	**11**	**6**	**9**	**9**	**7**	-	3	4	**21**	10	12	811
LPG	**12**	**8**	**10**	**11**	**8**	**6**	-	3	**21**	10	12	810
$\langle \mathcal{F}_\mathrm{i} \rangle$	**12**	**9**	**11**	**10**	**5**	**10**	**8**	-	**23**	10	13	832
SBD	**23**	**22**	**21**	19	19	20	19	17	-	17	12	840
PSy	**19**	**17**	**20**	**20**	**18**	**19**	**18**	**17**	**21**	-	15	864
C2	**28**	**24**	**26**	**23**	**25**	**23**	**23**	**23**	**25**	**23**	-	**903**

Figure 7.24: Same setup as in Figure 7.15, but with A^* using $h^{\text{LM-cut}}$. We additionally include SBD and $\mathrm{C2}$, and show PSy instead of Sym.

are unaffected, there can be tremendous advantages in terms of search space size and runtime when enabling both methods. In the plots on the right, we compare DS to aT using A^* with $h^{\text{LM-cut}}$. In general, as we have seen in the coverage tables, DS performs a lot better. There are, however, many instances where we see the effect of the early termination mechanism, where the search can terminate if the achieved leaf goal prices are globally minimal. This can be seen in the line above the diagonal.

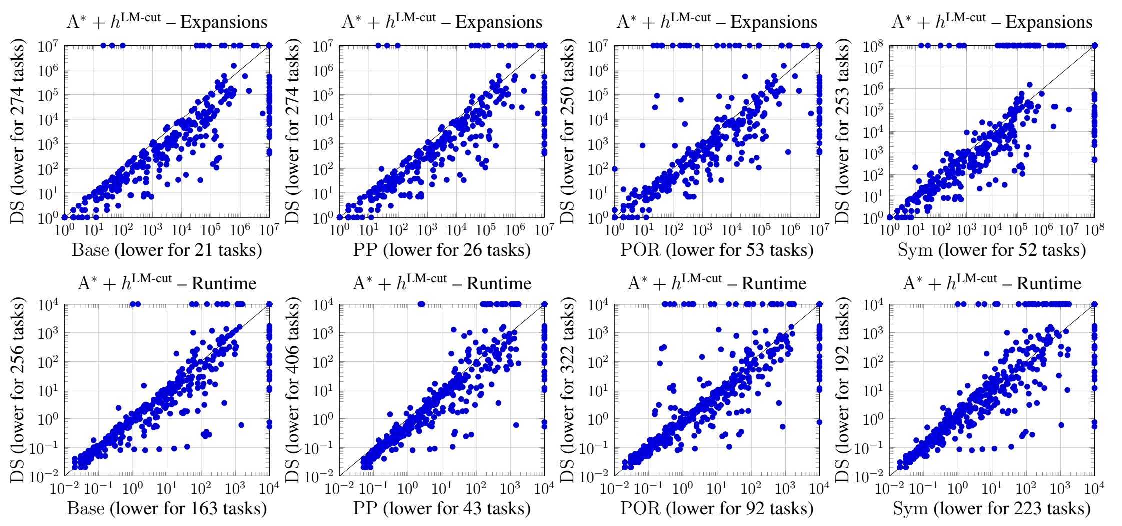


Figure 7.25: Scatter plots comparing the number of expanded states before the last f-layer in A^* (top), and runtime (bottom) of DS to (from left to right) explicit-state search without pruning, with partition-based pruning, stubborn-sets pruning, and symmetry breaking. In all plots we use factorings obtained by IA and run A^* with $h^{\text{LM-cut}}$.

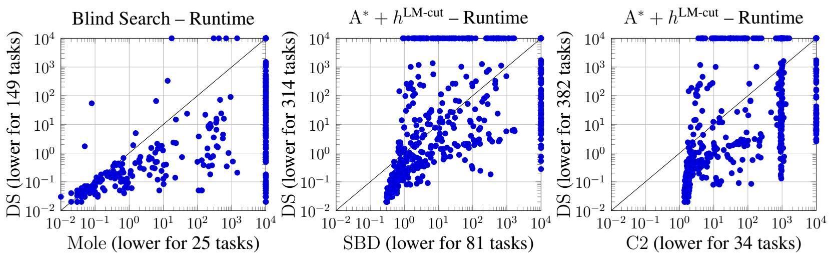


Figure 7.26: Scatter plots comparing the runtime of DS with blind search to (from left to right) unfolding with Mole, and with A^* and $h^{\text{LM-cut}}$ to symbolic search with SBD, and C2. In all plots we use factorings obtained by IA.

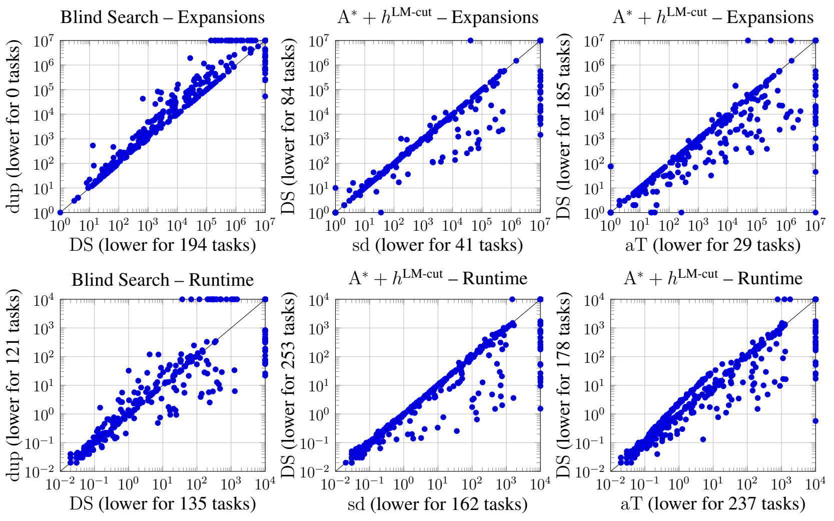


Figure 7.27: Scatter plots comparing the search space size and runtime of different decoupled search configurations. We investigate the effect of (from left to right) exact duplicate checking, different dominance relations and the g-value adaptation, and the anytime A^* variant. In all plots we use factorings obtained by IA.

			Explicit					Unf.	Sym.	Decoupled		
Domain	#	#$\mathcal{F}$	B	PP	POR	Sym	PSy	Mole	SBD	DS	dup	Sympa
BagBarman	20	16	8	7	4	8	4	0	**14**	8	8	11
Cavediving	25	21	3	3	3	**7**	**7**	1	6	4	5	3
DocTransfer	20	20	7	7	6	12	10	6	5	13	**14**	10
NoMystery	23	23	2	2	2	3	2	2	5	**12**	11	**12**
PegSol	24	24	**24**	**24**	**24**	**24**	**24**	20	**24**	**24**	**24**	**24**
Rovers	19	19	6	6	6	6	6	2	12	9	9	**16**
Tetris	20	20	5	5	5	**10**	5	5	5	5	5	5
TPP	30	30	16	16	14	16	14	10	21	15	16	**24**
Other	137	22	2	2	2	2	2	2	2	2	2	2
$\sum$	318	195	73	72	66	88	74	48	94	92	94	**107**
Unsolvable Benchmarks from Hoffmann et al. [2014]												
3-SAT	30	1	**1**	**1**	**1**	**1**	**1**	0	0	**1**	**1**	0
NoMystery	25	25	0	0	0	0	0	0	5	**25**	24	24
Rovers	25	25	1	1	1	2	1	0	5	2	2	**18**
TPP	25	25	6	5	2	6	2	0	7	1	5	**18**
Other	11	0	0	0	0	0	0	0	0	0	0	0
$\sum$	116	76	8	7	4	9	4	0	17	29	32	**60**
Total	434	271	81	79	70	97	78	48	111	121	126	**167**

Figure 7.28: Coverage data, i. e., the number of tasks proved unsolvable when using A* with $h^{\max}$, on instances where IA does not abstain. #$\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage in **bold face**.

7.7 Proving Unsolvability

We finally cover the last algorithmic planning problem, proving a task unsolvable. For all explicit-state methods, unfolding, and decoupled search, we run A* search with $h^{\max}$ for dead-end detection. For Mole, we use the Petri-net encoding based on TNF. We also include the Sympa planner in the comparison, which performed well in the unsolvability IPC 2016. We include all domains and instances from that competition, plus the instances from Hoffmann et al. [2014] that have not already been used in the competition.

In Figure 7.28, we show the results obtained on the instances where IA returns a factoring. We do not separately show tables for Fork and IFork, since these are subsumed by IA and LPG, i. e., all instances tackled by the former two are also tackled by the latter two with a fork, respectively inverted-fork factoring. Overall, decoupled search performs very well, clearly beating all other planners except Sympa, which is specialized to prove unsolvability. Like in the previous evaluations, the techniques yield good results in different domains. DS gives strong results in NoMystery and DocTransfer,

			Explicit					Unf.	Sym.	Decoupled		
Domain	#	#$\mathcal{F}$	B	PP	POR	Sym	PSy	Mole	SBD	DS	dup	Sympa
BagBarman	20	8	**8**	7	4	**8**	4	0	**8**	**8**	**8**	**8**
BagTransport	29	19	6	6	5	6	5	3	8	**11**	10	5
Cavediving	25	19	7	7	7	**11**	**11**	5	10	7	7	7
Chessboard	23	23	5	5	6	5	6	6	**11**	5	4	10
DocTransfer	20	20	7	7	6	12	10	6	5	13	**14**	10
NoMystery	23	23	2	2	2	3	2	2	5	**12**	11	**12**
PegSolR5	15	14	**4**	**4**	**4**	**4**	**4**	3	**4**	3	3	3
PegSol	24	24	**24**	**24**	**24**	**24**	**24**	20	**24**	22	**24**	**24**
Rovers	19	19	6	6	6	6	6	2	12	6	5	**16**
TPP	30	30	16	16	14	16	14	10	21	15	14	**24**
Other	90	20	10	10	10	10	10	10	10	10	10	10
$\sum$	318	219	95	94	88	105	96	67	118	112	110	**129**
Unsolvable Benchmarks from Hoffmann et al. [2014]												
3-SAT	30	30	15	15	**26**	15	**26**	10	10	15	15	8
Mystery	11	8	1	1	3	2	3	1	1	1	1	**4**
NoMystery	25	25	0	0	0	0	0	0	5	**25**	23	24
Rovers	25	25	1	1	1	2	1	0	5	1	1	**18**
TPP	25	25	6	5	2	6	2	0	7	2	1	**18**
$\sum$	116	113	23	22	32	25	32	11	28	44	41	**72**
Total	434	332	118	116	120	130	128	78	146	156	151	**201**

Figure 7.29: Same setup as in Figure 7.28, on instances where LPG does not abstain.

but is outperformed by SBD in BagBarman, and by Sym in Tetris. Sympa gives good results across all domains.

Moving to general-star factorings (Figure 7.29), the overall picture is quite similar, although decoupled search has a smaller advantage in coverage. This is partly due to more instances being tackled in 3-SAT, where stubborn-sets pruning works well in comparison. In BagTransport, however, which is tackled by LPG, DS has the highest coverage.

Comparing DS to dup, coverage increases by 4 instances with IA, obtaining better results overall. With LPG, dup rarely does a lot better than DS, but loses a few instances across many domains, so performs worse overall.

The factoring comparison in Figure 7.30 reveals that factorings obtained by MIS give the best results in combination with Base. Thus, the factoring portfolio chooses the following ordering of strategies: $\langle \mathrm{MIS}, \mathrm{IFork} \rangle$. The combination with IFork is due to its good performance in BagTransport. The pairwise comparison of factorings shows that, similar to the results in the previous sections, the strategies are quite complementary.

The plots in Figure 7.31 indicate a clear advantage of decoupled search over all explicit-state methods, both in terms of search space size (number expanded states), as

	Base	Fork	IFork	IA	MIS	LPS	LPG	$\langle\mathcal{F}_\mathrm{i}\rangle$	Sym	Sympa	Coverage
Base	-	0	0	2	0	2	4	0	0	4	151
Fork	**4**	-	**4**	2	0	2	**6**	0	3	5	189
IFork	**1**	1	-	3	1	2	**4**	0	1	4	156
IA	**6**	**3**	**6**	-	1	1	**5**	1	4	7	191
MIS	**6**	**4**	**6**	**3**	-	**2**	**6**	0	**6**	6	198
LPS	**6**	**4**	**5**	**3**	1	-	**4**	0	**6**	6	197
LPG	4	2	3	2	1	0	-	0	4	5	189
$\langle\mathcal{F}_\mathrm{i}\rangle$	**7**	**5**	**6**	**4**	**1**	**2**	**6**	-	**7**	6	203
Sym	**6**	**4**	**6**	**5**	3	5	**8**	3	-	7	168
Sympa	**11**	**9**	**11**	**8**	**8**	**8**	**9**	**8**	**10**	-	**233**

Figure 7.30: Pairwise comparison of factoring strategies when running A^* with h^{max}. We include a simulated sequential portfolio of strategies $\langle\mathcal{F}_\mathrm{i}\rangle$, and Sym and Sympa.

well as in runtime.

The runtime comparison in Figure 7.32 shows similar results. Decoupled search is clearly superior to unfolding with Mole. In comparison to SBD and Sympa, ignoring the initial overhead of the BDD library that both share, there is a clear trend favoring decoupled search on commonly solved instances.

Finally, for the relation of DS to dup (Figure 7.33), we get a mixed picture compared to satisficing and optimal planning. The advantage in runtime on instances where the pruning is not much affected is clearly stronger than for satisficing planning, but still weaker than for optimal planning. Overall, we see a moderate increase in the number of expanded states in many instances, but the efficiency of the duplicate check seems to outweigh this to some extent.

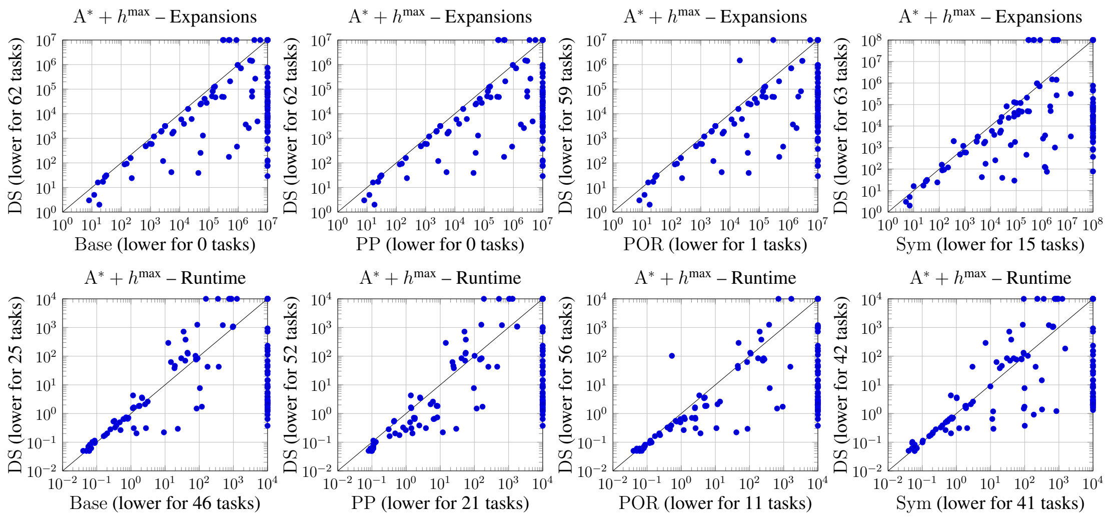


Figure 7.31: Scatter plots comparing the number of expanded states (top), and runtime (bottom) of DS to (from left to right) explicit-state search without pruning, with partition-based pruning, stubborn-sets pruning, and symmetry breaking. In all plots we use factorings obtained by IA and run A^* with $h^{\max}$.

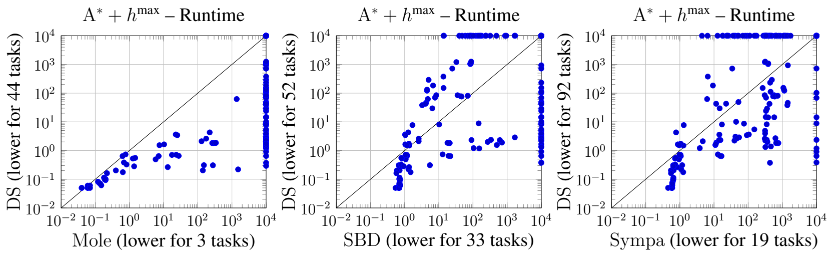


Figure 7.32: Scatter plots comparing the runtime of DS to (from left to right) unfolding (Mole), symbolic search (SBD), and Sympa. In all plots we use factorings obtained by IA and run decoupled search with A^* with $h^{\max}$.

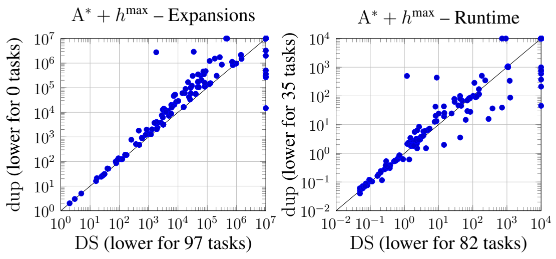


Figure 7.33: Scatter plots comparing the search space size (left) and runtime (right) of decoupled search with exact duplicate checking to dominance pruning. In all plots we use factorings obtained by IA.

7.8 Discussion

We want to conclude this chapter by briefly summarizing and discussing our findings. The most important observation is that decoupled search can yield very strong performance for all algorithmic planning problems, beating state-of-the-art planners when there are good problem decompositions. In general, decoupled search has shown a behaviour that is complementary to all existing state-space reduction techniques, i. e., the methods have their strengths in different domains. This confirms our theoretical findings from Chapter 6, where we showed exponential separations to these related methods.

We observed that, depending on the setting, different factoring types lead to very different results—while inverted-fork factorings beat state-of-the-art planners in satisficing planning, they perform poor in optimal planning. We tried to provide some intuitions and explain why such behaviour occurs, but we leave it to future work to investigate these properties more conclusively.

More generally, we found that it is not always beneficial to invoke decoupled search, if there are at least two mobile leaves. It seems like this condition is too weak to filter out instances where it would actually be better to perform, e. g., explicit-state search. We believe that it is an interesting topic for future research to more closely look into the conditions when decoupled search really excels, obtaining a significant state-space reduction. These conditions should be easy to check, for example requiring more leaf factors, or a minimum number of leaf-only actions per leaf. The monotonicity of the pricing function might play a role, too. Another option is to require a minimum ratio of leaf-only actions over the overall actions affecting a leaf. That way, it should be possible to filter-out instances where currently decoupled search does not perform well. As indicated before, since the factoring process is typically very fast, there is no drawback in checking if a good factoring exists, and any other method can be invoked if not.

Regarding the various different decoupled search configurations, we believe that it could also be interesting to investigate if there are easy-to-check conditions for when they perform better than the default. The anytime decoupled A^* variant, for example, works really well for fork factorings when using a good heuristic. The question is if similar conditions can be identified that imply, e. g., that duplicate checking will not lead to an increase in search space size.

In this work, we implemented a basic version of decoupled search in Fast Downward, which has proven to be highly competitive with related state-space reduction techniques, and state-of-the-art planners. This opens the stage for further fine-tuning of the implementation and planner configuration when running decoupled search.

Chapter 8

Summary

In this main part we introduced star-topology decoupled state-space search for classical AI planning. Decoupled search is based on decomposing a given planning task in terms of factoring its state variables into a center and multiple leaf components, so that the dependencies between these factors form a star topology. In this topology, the center factor can interact arbitrarily with the leaves, but any inter-leaf interaction must be via the center. The leaf factors in this decomposition are thus *conditionally independent*, given a sequence of actions that affect the center. Searching in the decoupled state space, we can hence branch only over center actions and enumerate all reachable leaf states for each of the leaf factors separately. An endpoint of a center path is called a decoupled state, a compact representation of the set of explicit states that results from multiplying out the reached leaf states across leaves. Thereby, a decoupled state represents *exponentially many* explicit states, leading to a tremendous reduction in state-space size.

We formally introduced the decoupled state space, which captures the reachability of leaf states in terms of the compliant-path graph, a structure that allows us to compactly keep track of the leaf paths *compliant* with the current center path. We proved the correctness of the decoupled state space and showed that it preserves completeness as well as optimality, when maintaining a pricing function for each decoupled state during search. The pricing function represents the costs of the compliant leaf paths, avoiding the need to store the compliant-path graph, and is sufficient to perform search. We looked into the size of the decoupled state space in comparison to the explicit state space, showing that in general (1) it can in fact be exponentially *larger*, and (2) even infinite. (2) can easily be tackled by using a simple dominance pruning that prunes decoupled states if the price of all leaf states is higher than in a previously seen decoupled state. We can also address (1), but at the cost of solving a co-**NP**-hard problem for each reached decoupled state. The corresponding algorithm, *hypercube pruning*, incurs a prohibitive overhead, as shown in our evaluation, but rarely leads to a significantly increased state pruning. We conclude that, although possible in theory, exponential blow-ups of the decoupled state space do not usually occur in practice, as far as this is captured by standard planning benchmarks.

Connecting the decoupled state representation to standard search algorithms and planning heuristics, we opened up the possibilities to combine decoupled search with all the state-of-the-art techniques that are widely employed in planning research. We showed that completeness and optimality of the base search algorithm are preserved by using a simple transformation. For heuristics, we introduced a compilation that (in principle) allows us to take any planning heuristic and compute estimates for decoupled states. Importantly, our compilation preserves all important properties of the heuristic, like admissibility and perfectness in the limit.

We presented several methods that are capable of identifying the required structure in a planning task and compute a star factoring. We proved that, in general, it is **NP**-hard to compute a factoring that maximizes the number of leaf components, which is desirable because the reduction of decoupled search is exponential in that number. Besides the number of leaves, we identified the leaf *mobility* to capture an important property of factorings, namely the "amount of work" each leaf can do on its own, which is measured in the number of actions that only affect the leaf, but not the center. We devised several *factoring strategies* that are effective in identifying good factorings.

Decoupled search is related to several other state-space reduction techniques, such as partial-order reduction (POR) and Petri-net unfolding, with which it shares the source of its reduction, the independence of components/actions of a planning task. For the relation to unfolding, we proved that under certain conditions—singleton factors, absence of prevail conditions, and compatible search orders—the state-space representation of the two methods is actually polynomially related. Compared to both POR and unfolding, we proved that decoupled search can lead to exponentially higher state-space reduction, and vice versa. Decoupled search is by nature a form of *factored planning*, which has previously seen attention in the planning community in various works. We argued that decoupled search is a new form of factored planning, since the structural requirements it imposes on the planning task lead to an efficient handling of the cross-factor dependencies that other forms of factored planning need to resolve in intricate ways. There is also a connection to techniques like symbolic state representation, which also compactly represent sets of states, and symmetry breaking, which can exponentially reduce search effort, too. Decoupled search is exponentially separated from both.

Finally, we empirically evaluated our implementation of decoupled search in the Fast Downward planning framework on three algorithmic planning problems—optimal planning, satisficing planning, and proving unsolvability. We compared decoupled search to state-of-the-art planners and the aforementioned related techniques. This revealed that, indeed, decoupled search can lead to dramatic reductions in search effort, beating state-of-the-art planning systems in various scenarios. We also observed that the reduction achieved by decoupled search is often orthogonal to that of other popular and effective techniques such as symmetry breaking.

Part III

Combination with Other State-Space Reduction Methods

Chapter 9

Introduction

In AI planning, as well as many related areas, the state explosion problem has been identified as a major challenge to make state-space search algorithms scale up to larger-sized problems. Various techniques have been developed to tackle the state explosion. We have discussed several methods in Part II and analyzed their relation to decoupled search in theory and practice. In this part, we focus on techniques that can yield state-space reductions *orthogonal* to the one achieved by decoupled search, and *combine* them with decoupled search. In particular, we devise combined algorithms with partial-order reduction via strong stubborn sets (Chapter 10), symmetry breaking (Chapter 11), symbolic state representations (Chapter 12), and state-dominance pruning (Chapter 13).

For the first three of these techniques, we have shown exponential separations to decoupled search indicating that the state-space reduction is in general orthogonal. Therefore, it is highly promising to combine the methods with decoupled search to, ideally, obtain a novel algorithm that inherits the strengths of both components. The connection to state-dominance pruning is rather weak. While decoupled search exploits component independence to compactly represent sets of states, state-dominance pruning is based on dominance relations that ensure that a state s is *as close to the goal* as another state s'. Then, if s has been visited first, s' can be pruned without sacrificing completeness and optimality. Given this completely different underlying principles, the question arises if state-dominance can prove effective for pruning the decoupled state space.

Stubborn sets pruning can obtain exponential advantages over decoupled search in the presence of concurrent center actions. We devise *decoupled strong stubborn sets*, which can exploit such independent actions by branching only over a subset of the actions applicable in a decoupled state. We prove that decoupled strong stubborn sets are optimality-preserving, and show exponential separations from both individual components. This advantage is confirmed in our experimental evaluation, where indeed there are domains in which the combination improves over both baselines.

The reduction of symmetry breaking is due to objects, or groups of objects, that have symmetric properties. During search, this can be exploited by keeping only a single of a set of symmetric states, the *canonical* representative of this so-called *orbit* of states. This is obviously completely different from decoupled search. Hence, we extend the symmetry relation over explicit states to decoupled states, which allows a similar pruning in the decoupled state space, pruning all but the canonical decoupled representative state from each orbit. Like with stubborn sets, we prove the correctness of the combined algorithm, show that it is exponentially separated from its components, and evaluate it empirically on the standard benchmark set.

For the combination with symbolic state representations, we use decision diagrams, namely binary and algebraic decision diagrams (BDDs/ADDs) to represent the pricing function of decoupled states. With factorings that have large leaf factors, representing the pricing function explicitly can be prohibitive both in terms of runtime and memory. We show how to represent the set of reached leaf states and the prices symbolically, how to connect to the main search which branches over center actions, and how to compute decoupled heuristic functions on the symbolic representation.

State-dominance pruning has mostly been applied in optimal planning, where such kind of pruning leads to significant reductions. We introduce novel notions of decoupled-state dominance specialized to fork factorings. These dominance relations can improve the dominance pruning of decoupled search by ignoring leaf states that cannot possibly contribute to cheaper goal prices, or restricting the dominance check to a relevant subset of leaf states. We provide a theoretical analysis comparing the new dominance relations and evaluate their pruning power empirically.

Chapter 10

Partial-Order Reduction

Partial-order reduction is an established method to reduce the search space without affecting the existence of (optimal) plans [Valmari, 1992; McMillan, 1992; Godefroid and Wolper, 1991; Esparza et al., 2002; Edelkamp et al., 2004a; Rodríguez and Schwoon, 2013]. It analyzes action dependencies to prune commutative parts of the search space. In this chapter, we show how to apply this idea to decoupled state-space search.

Just like explicit-state search, decoupled search is adversely affected by commutative parts of its search space. We herein adapt *strong stubborn set* pruning (SSS), which was first devised in formal verification [Valmari, 1989], and was later adapted to AI planning [Alkhazraji et al., 2012; Wehrle et al., 2013; Sievers and Wehrle, 2021].

A major challenge is the more complex structure of the decoupled state space. Here, for the leaf factors, the distinction between "past" (search path so far) and "future" (remaining search path) becomes complicated, because even the path the leaf takes up to a decoupled state $s^{\mathcal{F}}$ is committed to only in the future. Hence, even if a leaf state s^L is reached in $s^{\mathcal{F}}$, SSS pruning needs to reason about the enablers of s^L, i. e., about the actions that *might* be committed to in order to support s^L at plan extraction time.

We spell out how to address this challenge, designing optimality-preserving *decoupled strong stubborn set (DSSS)* pruning methods. We introduce a design for star topologies in full generality, as well as simpler design variants for the practically relevant fork and inverted-fork special cases. We show that there are cases where DSSS pruning is exponentially separated from both, decoupled search and SSS pruning, exhibiting true synergy where the whole is more than the sum of its parts. For the practical evaluation, we focus on optimal planning, where stubborn sets pruning has been most successfully applied in planning. DSSS pruning reliably inherits the best of decoupled search and SSS pruning. Sometimes—being more than the sum of its parts—it outperforms both. Where both techniques are effective, DSSS pruning is able to compete with state-of-the-art optimal planning systems. Some technical details are only summarized in the main text. The full details and proofs are available in Appendix B.1.

This chapter is based on Gnad et al. [2019a], which in turn extends a version of decoupled strong stubborn sets restricted to fork topologies [Gnad et al., 2016d]. Martin Wehrle contributed to the latter work and to early concepts of Gnad et al. [2019a]. All contributions included in this chapter are due to the author of this work.

10.1 Background

Intuitively, a strong stubborn set (SSS) for a state s is a subset A_s of the applicable actions guaranteed to contain the starting action of at least one optimal solution for s. In a nutshell, such a set A_s is derived by selecting an open part p of the goal condition, collecting all actions a that may recursively be used to *enable* p, and collecting all actions a' that these a *interfere* with. Actions not included by this process can be safely ignored in s, as they are not relevant to p, and as they are commutative with A_s, i. e., they can still be applied later on if needed for some other part of the goal.

In the remainder of this section, we give a summary presentation of strong stubborn sets pruning, sufficient to understand its workings and introducing the notations we will use in our analysis later on.

We need the notion of strong optimality to ensure that the search makes progress and does not get trapped in 0-cost cycles. A plan π for a state s is *strongly optimal* if it is optimal and contains the minimum number of 0-cost actions among all optimal plans for s. For decoupled search, we say that a decoupled plan $\pi^{\mathcal{F}}$ for a decoupled state $s^{\mathcal{F}}$ is *strongly optimal* if it is augmented-optimal and $\pi^{C}(\pi^{\mathcal{F}})$ contains the minimal number of 0-cost actions among all augmented-optimal decoupled plans for $s^{\mathcal{F}}$.

We will often talk about compatible vs. incompatible partial variable assignments (action preconditions or effects), so we introduce shorthands for this. Given partial assignments p and q, we say that p and q are *compatible*, written $p \parallel q$, if there exists no $v \in \mathcal{V}$ such that $v \in \mathsf{vars}(p) \cap \mathsf{vars}(q)$ and $p[v] \neq q[v]$; we say that p and q are *incompatible*, written $p \nparallel q$, if such v does exist.

We say that an action a is *reached* in a (decoupled) state, if its precondition $\mathsf{pre}(a)$ is reached in that state.

Throughout this chapter, we will use a running example to illustrate the concepts and definitions. The example is a variant of the logistics task used in other parts of this work. Next, we briefly describe the variants of the example we will use in this chapter.

Example 10. *We have two locations A and B, a single truck whose position is modeled by variable T with $\mathcal{D}(T) = \{A, B\}$, and n packages whose position is modeled by variables p_i with $\mathcal{D}(p_i) = \{A, B, T\}$. The truck and all packages are initially at A, and the goal is for the packages to be at B. The actions take the form drive(T, x, y), load(T, p_i, x), and unload(T, p_i, x) where $x, y \in \{A, B\}$. Actions costs are unit 1 for simplicity.*

In what we will call the Vanilla *variant of this example, the action preconditions and effects are the commonly used ones, namely: drive(T, x, y) has precondition $\{T = x\}$*

and effect $\{T = y\}$; load(T, p_i, x) has precondition $\{p_i = x, T = x\}$ and effect $\{p_i = T\}$; and unload(T, p_i, x) has precondition $\{p_i = T, T = x\}$ and effect $\{p_i = x\}$.

In what we will call the NoEmpty *variant of this example, the actions are the same except that driving the truck now takes the form drive(T, x, y, p_i), with precondition $\{T = x, p_i = T\}$ and effect $\{T = y\}$. That is, here the truck cannot drive without having a package inside.*

Unless stated otherwise, we will consider the default factoring *$\mathcal{F}$ setting $\{T\}$ as the center, and setting each $\{p_i\}$ as a leaf. For the Vanilla example, this is a fork factoring. For the NoEmpty example, where the dependencies between T and p_i go in both directions, this is (not a fork but) a star factoring.*

We will sometimes consider the inverted-fork factoring*, setting $\{p_1, \dots, p_n\}$ as the center and $\{T\}$ as the (single) leaf. Observe that, for the Vanilla example, this is an inverted-fork factoring.*

Strong stubborn set pruning is a form of partial-order reduction in state-space search, exploiting action commutativity to safely prune a subset of the applicable actions in a state. As previously outlined, the method analyses, given a state s, how actions enable each other to achieve some open part of the goal when starting from s, and how these actions may interfere with other actions. We briefly spell this out in what follows, in a form connecting directly to our later extensions for decoupled search.

We use a *syntactic* characterization of stubborn sets, based directly on the syntactic specifications of actions in the input task, as opposed to a *semantic* characterization relying on properties of action executions in particular states. The syntactic characterization is coarser, but is simpler and is what we use in practice.

Let s be a solvable non-goal state, $s \not\models \mathcal{G}$, and let $A_s \subseteq \mathcal{A}$ be a set of actions. We say that A_s is *safe* for s if there exists a strongly optimal plan $\pi = \langle a_1, \dots, a_n \rangle$ for s such that $a_1 \in A_s$. We need π to be *strongly* optimal, because otherwise the search can become incomplete in the presence of 0-cost actions.[1] Note that we need to take care only of solvable states here as, on unsolvable states, any pruning method (any subset of applicable actions) is optimality-preserving.

Strong stubborn sets are a means to derive safe sets A_s. Towards this, two basic notions are employed:

- Given a partial assignment p where $s \not\models p$, an action set $A \subseteq \mathcal{A}$ is a *necessary enabling set* for p in s if there exists $v \in \mathsf{vars}(p)$ such that $s[v] \neq p[v]$ and $A = \{a' \in \mathcal{A} \mid \mathsf{eff}(a')[v] = p[v]\}$.

- Given $a, a' \in \mathcal{A}$, we say that a and a' *interfere* if $\mathsf{eff}(a) \not\parallel \mathsf{pre}(a')$ or $\mathsf{eff}(a') \not\parallel \mathsf{pre}(a)$ or $\mathsf{eff}(a) \not\parallel \mathsf{eff}(a')$.

[1] This happens in the following example: Let s, s' be two solvable non-goal states such that $s[\![a_1]\!] = s'$, and $s'[\![a_2]\!] = s$, where $\mathsf{cost}(a_1) = \mathsf{cost}(a_2) = 0$. Then a_1 starts an optimal plan in s and a_2 in s', but the action sets $A_s = \{a_1\}$ and $A_{s'} = \{a_2\}$ prune all solutions for s and s'.

In other words, a necessary enabling set for p is the set of achievers of one part of p that is still open; a and a' interfere if one disables the other's precondition, or they have conflicting effects.

These two notions can be combined in a simple recursive way to derive a safe action set for any non-goal state s. Namely, an action set A_s is a *strong stubborn set (SSS)* for s if the following conditions hold:

(i) A_s contains a necessary enabling set for $\mathcal{G}$ in s.

(ii) For all actions $a \in A_s$ not applicable to s, A_s contains a necessary enabling set for $\mathsf{pre}(a)$ in s.

(iii) For all actions $a \in A_s$ applicable to s, A_s contains all actions a' interfering with a and where $\mathsf{pre}(a) \parallel \mathsf{pre}(a')$.

The proof that such A_s is safe was originally given in planning by Alkhazraji et al. [2012], and was later enhanced by Wehrle and Helmert [2014] allowing (amongst other things) reachability information, captured above in the requirement $\mathsf{pre}(a) \parallel \mathsf{pre}(a')$ in (iii). We include the proof here as our later proofs for decoupled strong stubborn sets will be extensions thereof.

Consider some plan $\pi = \langle a_1, \dots, a_n \rangle$ for s. Then the following properties hold:

(a) There exists an action shared between A_s and π, i. e., $A_s \cap \{a_1, \dots, a_n\} \neq \emptyset$.

This is because by (i) A_s contains a necessary enabling set for $\mathcal{G}$ in s.

Given (a), let a_k be the shared action with smallest index, i. e., say that $a_k \in A_s$ and $\{a_1, \dots, a_{k-1}\} \cap A_s = \emptyset$.

(b) a_k is applicable to s.

This is because, otherwise, by (ii) A_s contains a necessary enabling set A for $\mathsf{pre}(a_k)$ in s, and one $a \in A$ must precede a_k on π, in contradiction to a_k being the first shared action.

(c) a_k does not interfere with any of the preceding actions a_i, $1 \leq i \leq k-1$, where $\mathsf{pre}(a) \parallel \mathsf{pre}(a_i)$.

This is because, if it did, then by (iii) we would have $a_i \in A_s$, again in contradiction to a_k being the first shared action.

(d) a_k can be moved to the front of π, i. e., $\pi' := \langle a_k, a_1, \dots, a_{k-1}, a_{k+1}, \dots, a_n \rangle$ is a plan for s.

To see this, consider that, by (b), a_k is applicable in s. As a_1 is applicable in s as well, we must have $\mathsf{pre}(a_k) \parallel \mathsf{pre}(a_1)$, so by (c) a_1 does not interfere with a_k, and hence a_k is still applicable in $s[\![a_1]\!]$. But then, the same argument applies to a_2 and $s[\![a_1]\!]$, so iterating the argument we obtain that, for $1 \leq i \leq k-1$, $\mathsf{pre}(a_k) \parallel \mathsf{pre}(a_i)$ and a_i does not interfere with a_k.

The claim follows directly from (d).

10.2 Decoupled Strong Stubborn Sets

We now show how to apply the idea of strong stubborn set construction to decoupled search, for arbitrary star topologies. Chapter 10.2.1 introduces basic concepts used in all our constructions, lifting, amongst others, the notions of safety and necessary enabling sets to decoupled search. To guarantee optimality, decoupled search must continue the search on goal decoupled states. For our purposes here, the main implication is that trying to reach the goal (non-goal decoupled state) is different from trying to decrease the prices of compliant goal leaf paths (goal decoupled state). Our stubborn set constructions will distinguish these two cases. Chapter 10.2.2 introduces decoupled strong stubborn sets for non-goal decoupled states $s^{\mathcal{F}}$, where $\mathcal{G}$ has yet to be reached; and Chapter 10.2.3 shows how to handle goal decoupled states $s^{\mathcal{F}}_G$, where $\mathcal{G}$ is already reached yet may be reached with cheaper compliant leaf paths below $s^{\mathcal{F}}_G$. In each of these two sections, we prove the safety of our constructions.

Throughout, we assume a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ and a factoring $\mathcal{F}$ with center C and leaves $L \in \mathcal{L}$.

10.2.1 Basic Concepts

In the standard setting, i. e., in explicit state-space search, safety of an action set A_s for a non-goal state s means that there exists a strongly optimal plan for s starting with an action from A_s. For decoupled search, i. e., decoupled states $s^{\mathcal{F}}$, this definition changes in two ways. First, instead of plans for s we need to talk about decoupled plans for $s^{\mathcal{F}}$. Second, as decoupled search has to continue below $s^{\mathcal{F}}$ even if $s^{\mathcal{F}}$ is a goal decoupled state, instead of "non-goal" we need to say that the empty decoupled plan is not optimal for $s^{\mathcal{F}}$. We hence define:

Definition 27 (Safety). *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}$ *be a solvable decoupled state for which* $\langle\rangle$ *is not an optimal decoupled plan, and let* $A^{\mathcal{F}}_s \subseteq \mathcal{A}^C$ *be a set of center actions.*

We say that $A^{\mathcal{F}}_s$ *is* safe *for* $s^{\mathcal{F}}$ *if there exists a strongly optimal decoupled plan* $\pi^{\mathcal{F}}$ *for* $s^{\mathcal{F}}$ *such that* $\pi^C(\pi^{\mathcal{F}}) = \langle a^C_1, \dots, a^C_n \rangle$ *where* $a^C_1 \in A^{\mathcal{F}}_s$.

Clearly, a safe $A^{\mathcal{F}}_s$ for $s^{\mathcal{F}}$ plays the same role in search as a safe A_s for s: pruning outgoing transitions from $s^{\mathcal{F}}$ induced by actions outside $A^{\mathcal{F}}_s$ preserves optimality and completeness of the search. Note that, if $\langle\rangle$ is an optimal decoupled plan for $s^{\mathcal{F}}$, then any pruning method preserves optimality and completeness on $s^{\mathcal{F}}$, so such $s^{\mathcal{F}}$ can be disconsidered in the analysis of safety (though not in the generation of stubborn sets, as we cannot recognize such states up front).

To identify safe $A^{\mathcal{F}}_s$, we will rely on commutativity, i. e., on plan permutations. Whereas previously this simply referred to plans for s, the notion of permutations now becomes significantly more complicated, as we have to consider both, decoupled plans and their associated global plans.

Recall that, given a decoupled state $s^{\mathcal{F}}$, a decoupled plan $\pi^{\mathcal{F}}$ for $s^{\mathcal{F}}$ merely is a path in a reformulated search space (the decoupled state space). The meaning of $\pi^{\mathcal{F}}$ for the actual input task is defined in terms of the global plans π given $\pi^{\mathcal{F}}$. These consist of the center action sequence $\pi^C(s^{\mathcal{F}}) \circ \pi^C(\pi^{\mathcal{F}})$, augmented with a cheapest compliant goal leaf path π^L for each leaf factor L. We define permutations over both, $\pi^{\mathcal{F}}$ and π simultaneously:

Definition 28 (Plan Permutation)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}$ *be a solvable decoupled state, and let* $\pi^{\mathcal{F}}$ *and* $\pi^{\mathcal{F}\prime}$ *be decoupled plans for* $s^{\mathcal{F}}$*.*

We say that $\pi^{\mathcal{F}}$ *and* $\pi^{\mathcal{F}\prime}$ *are* permutations *if (i)* $\pi^C(\pi^{\mathcal{F}})$ *is a permutation of* $\pi^C(\pi^{\mathcal{F}\prime})$*, and (ii) there exist global plans* π *and* π' *for* $\pi^{\mathcal{F}}$*, respectively* $\pi^{\mathcal{F}\prime}$*, such that* π *is a permutation of* π'*.*

Here (ii) essentially says that, on top of the center paths being commutative, the corresponding cheapest compliant goal leaf paths need to be commutative as well. For safety, (i) would be enough in principle, i. e., it would be enough to identify a center action starting a permutation of a strongly optimal decoupled plan. Our construction of strong stubborn sets however, i. e., our sufficient criterion for permutability, relies on both (i) and (ii).

A second major implication of the interplay between decoupled plans and their global plans is that we have to think carefully about what is the "future" for a decoupled state $s^{\mathcal{F}}$. Stubborn set construction reasons about the future, which in the standard setting simply are the possible plans π for the state s in question. But what is the future for $s^{\mathcal{F}}$? A priori, of course the possible decoupled plans $\pi^{\mathcal{F}}$ for $s^{\mathcal{F}}$. Yet, $\pi^{\mathcal{F}}$ merely is a path in a reformulated search space. An associated global plan π is not an action sequence "starting in $s^{\mathcal{F}}$": π starts in the initial state of the input planning task. In particular, the goal leaf paths π^L in π schedule actions along both, the center action subsequence $\pi^C(s^{\mathcal{F}})$ up to $s^{\mathcal{F}}$, and the center action subsequence $\pi^C(\pi^{\mathcal{F}})$ behind $s^{\mathcal{F}}$.

Our natural solution to this issue is to consider the "past" as everything scheduled along $\pi^C(s^{\mathcal{F}})$, and the "future" as everything scheduled along $\pi^C(\pi^{\mathcal{F}})$. Precisely, say that π is a global plan given $\pi^{\mathcal{F}}$, and $\pi = \langle a_1, \dots, a_n\rangle$. Consider the center action subsequence $\pi^C(s^{\mathcal{F}}) \circ \pi^C(\pi^{\mathcal{F}})$ in π, and consider, within that subsequence, the starting action of $\pi^C(\pi^{\mathcal{F}})$. Denote by t the index of that action occurence in π, i. e., a_t is the start of $\pi^C(\pi^{\mathcal{F}})$ within π. We capture the "past" as $\pi^{past} := \langle a_1, \dots, a_{t-1}\rangle$, and the "future" as $\pi^{future} := \langle a_t, \dots, a_n\rangle$. In other words, we consider the first center action behind $s^{\mathcal{F}}$—the center action decoupled search applies in $s^{\mathcal{F}}$ to find $\pi^{\mathcal{F}}$—and we use that action as the pivot separating the past from the future. We will be using the notations a_t, π^{past}, and π^{future} throughout.

While this definition of the future is reasonably simple, it leaves us with another subtlety, namely that "the same" π may be scheduled in different ways, leaving us with undesirable ambiguity regarding the difference between past and future. To see this, consider any goal leaf path π^L. Observe that the part of π^L contained within π^{past}—the leaf path's prefix scheduled in the past—must comply with $\pi^C(s^{\mathcal{F}})$. Yet nothing forces this prefix to be maximal: we are free to schedule parts of π^L in the future, even if they comply with $\pi^C(s^{\mathcal{F}})$ so could be scheduled in the past. For illustration, say the center (e. g., a truck) has already done up to $s^{\mathcal{F}}$ what it needs to do in order to support a leaf (e. g., providing transport for a package). But now the leaf needs to do additional tasks independently from the center (e. g., unpacking, assembling package content, . . .). Then π *can* schedule these additional tasks in π^{past}, but may just as well schedule them in π^{future}.

In other words, π^{future} may contain actions that have nothing to do with "the remaining task starting from $s^{\mathcal{F}}$". It will be of service to our analyses to avoid this kind of behavior. We will restrict focus to global plans π that are *past-maximal*, where, intuitively, the leaves do nothing in π^{future} that they could do in π^{past}.

The precise form of past-maximality we will need differs depending on the context: we introduce one notion for star topologies, and another different one for our analysis of fork topologies below. In our notion for star topologies, past-maximality means that no leaf action scheduled in the future can be moved in front of the pivot action a_t:

Definition 29 (Past-Maximality). *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}$ *be a solvable decoupled state, let* $\pi^{\mathcal{F}}$ *be a decoupled plan for* $s^{\mathcal{F}}$*, and let* π *be a global plan given* $\pi^{\mathcal{F}}$*.*

Define a_t*,* π^{past}*, and* π^{future} *as above. We say that* π *is* past-maximal *if, for every leaf factor* L *and every* $k > t$ *where* $a_k \in \mathcal{A}^L_{\not C}$*, if we change* π *by moving* a_k *to any position in front of* a_t*, then* π *is not a plan anymore.*

It is easy to see that a focus on past-maximal global plans is not restricting, in the sense that every global plan can be rescheduled to be past-maximal:

Lemma 8. *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}$ *be a decoupled state, let* $\pi^{\mathcal{F}}$ *be a decoupled plan for* $s^{\mathcal{F}}$*, and let* π *be a global plan given* $\pi^{\mathcal{F}}$*.*

Then there exists a permutation of π *that is a global plan given* $\pi^{\mathcal{F}}$*, and that is past-maximal.*

Proof. Obtain such a permutation π' as follows. Start with $\pi' := \pi$. If π' is past-maximizing, stop. Else, select a counter-example L and k, move a_k in front of a_t in π', and iterate. This algorithm terminates as, after each step, there is one action less behind a_t. The outcome π' satisfies the claim as we have not changed the center-action subsequence, and the permuted leaf paths are still compliant with that sequence. $\square$

We are now ready to introduce the ingredients underlying our stubborn set constructions. As before, we will need to reason about how actions *interfere* with each other, and how they *enable* each other. The former remains exactly the same as before. The latter, however, needs to be adapted substantially given the more complex structure of the search space. We need to distinguish between enabling (a) conditions p not reached in a decoupled state, vs. (b) conditions p reached in a decoupled state. For (b), our constructions will depend on the specific context (stars vs. forks/inverted forks, goal vs. non-goal decoupled states), so we will introduce these in the respective context. For (a), the same construction works in all contexts. Namely, we employ the following extended definition of necessary enabling sets:

Definition 30 (Decoupled Necessary Enabling Set)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}$ *be a decoupled state, and let* p *be a partial variable assignment not reached in* $s^{\mathcal{F}}$*.*

An action set A *is a* decoupled necessary enabling set *for* p *in* $s^{\mathcal{F}}$ *if one of the following conditions holds:*

(i) There exists $v \in \mathsf{vars}(p) \setminus C$ *such that* $p[v]$ *is not reached in* $s^{\mathcal{F}}$*, and* $A = \{a \in \mathcal{A} \mid \mathsf{eff}(a)[v] = p[v]\}$.

(ii) Condition (i) does not apply, and there exists L *such that* $p[L]$ *is not reached in* $s^{\mathcal{F}}$*, and* $A = \bigcup_{v \in \mathsf{vars}(p) \cap L} \{a \in \mathcal{A} \mid \mathsf{eff}(a)[v] = p[v]\}$.

(iii) Neither (i) nor (ii) apply, and there exists $v \in \mathsf{vars}(p) \cap C$ *such that* $\mathsf{center}(s^{\mathcal{F}})[v] \neq p[v]$*, and* $A = \{a \in \mathcal{A} \mid \mathsf{eff}(a)[v] = p[v]\}$.

In other words, a decoupled necessary enabling set first checks whether (i) some leaf-variable value is not reached in $s^{\mathcal{F}}$, then checks whether (ii) some leaf-factor state is not reached in $s^{\mathcal{F}}$, then checks whether (iii) some center-variable value is not reached in $s^{\mathcal{F}}$. In each case, the set of all achieving actions is selected.

As p is not reached in $s^{\mathcal{F}}$, one of (i) – (iii) must necessarily be true. Observe that this is not so for just (i) and (iii) alone, as every single leaf-variable value in p may already be reached in some leaf state of L, yet that may not be so for their combination. In other words, necessary enabling sets become more complicated, relative to standard search, due to the difference between leaf states being true in a unique state s, vs. leaf states being true in one out of the set of states represented by a decoupled state $s^{\mathcal{F}}$.

We remark that the ordering of conditions (i) – (iii) in Definition 30 is arbitrary in the sense that any ordering is possible in principle. The stated ordering encapsulates a preference to regress over open leaf conditions first, the aim being to extract open center conditions (which the decoupled strong stubborn set will be chosen to support) "as close as possible" to the current decoupled state $s^{\mathcal{F}}$. The intuition is to first move the leaves towards their goals—by enabling the required center preconditions—and to care for the center goals only once the leaf goals are achieved.

10.2.2 Non-Goal Decoupled States

Towards our constructions for non-goal decoupled states $s^{\mathcal{F}}$, we next specify how to enable a condition p *reached* in $s^{\mathcal{F}}$.

To understand why this is needed, consider the role of reached leaf states s^L in $s^{\mathcal{F}}$, where for each L one such s^L will be used, and that commitment will be made at plan extraction time. This implies that, just because some condition p on L is reached in $s^{\mathcal{F}}$, it does not have to be true at the corresponding point in the global plan π we will finally commit to. Namely, defining "the corresponding point" in π as the state $\mathcal{I}[\![\pi^{past}]\!]$ before application of the pivot action a_t, p will have to be false at this point if a_t has a precondition contradicting p. Nevertheless, p may have to be true at some later point along π, to enable some other action or part of the goal.

For illustration, consider the decoupled state $s^{\mathcal{F}} := \mathcal{I}^{\mathcal{F}}$ in our NoEmpty example, consider $L := \{p_1\}$, and consider $p := \{p_1 = A\}$. Condition p is reached in $s^{\mathcal{F}}$. Yet, say our decoupled plan $\pi^{\mathcal{F}}$ decides to use $a_t := \mathit{drive}(A, B, p_1)$ in $s^{\mathcal{F}}$. Then p must be false prior to the application of a_t in the corresponding global plan π, due to the incompatible precondition $\{p_1 = T\}$. If p is required later on, e. g., if $\mathcal{G}[p_1] = A$, then p will need to be enabled in the future, behind the application of a_t in π. We need to reason about how to do that.

So, how do we find a set of actions that will necessarily be used to enable a reached L condition p in the future? A simple possibility would be to follow Definition 30 (ii) and just select all actions supporting any variable value in p. Done recursively though, this is likely to collect a very large set of reached actions that affect L. We hence design a definition more tailored to the specific situation: a reached L condition p, false in some reached L state s^L (namely $\mathcal{I}[\![\pi^{past}]\!][L]$), yet true in some L state r^L (namely one r^L visited along π^{future}), where r^L is achieved on a path $\pi^L_{s\to r}$ starting from s^L (namely the respective leaf path segment within π^{future}). It suffices to ensure that we collect at least one action along every such path $\pi^L_{s\to r}$:

Definition 31 (Reached-Enabling Set)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task and $\mathcal{F}$ a factoring. Let $s^{\mathcal{F}}$ be a decoupled state, let L be a leaf factor, and let p be a partial assignment to L reached in $s^{\mathcal{F}}$.*

We say that an L-state s^L is a reached-enabling state *for p in $s^{\mathcal{F}}$ if* *(i) s^L is reached in $s^{\mathcal{F}}$, and $s^L \not\models p$; and*

(ii) there is an L path $\pi^L_{s \to r}$ from s^L to an L state r^L where $r^L \models p$.

An action set A is a reached-enabling set *for p in $s^{\mathcal{F}}$ if every path $\pi^L_{s \to r}$ as in (ii) contains at least one action from A.*

For illustration, in our above example where $p = \{p_1 = A\}$, the only reached s^L where $s^L \not\models p$ is $s^L = \{p_1 = T\}$. That s^L has two outgoing transitions, labeled by unload(T, p_1, A) respectively unload(T, p_1, B). Intuitively, only unload(T, p_1, A) makes sense as an enabler for p. And indeed, that action is a reached-enabling set on its own, because every path $\pi^L_{s \to r}$ achieving p must use it eventually.

It is easy to see that our construction is correct, in the sense that a reached p not true at the end of π^{past}, yet true somewhere on π^{future}, must be enabled on π^{future} with an action from a reached-enabling set:

Lemma 9. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \text{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state, let L be a leaf factor, and let p be a partial assignment to L reached in $s^{\mathcal{F}}$. Let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$, and let $\pi = \langle a_1, \dots, a_n \rangle$ be a global plan given $\pi^{\mathcal{F}}$. Let A be a reached-enabling set for p in $s^{\mathcal{F}}$.*

Define a_t, π^{past}, and π^{future} as before. Denote the states traversed by π as $s_0, \dots, s_n$. If $s_{t-1} \not\models p$ but there exists $k > t$ such that $s_{k-1} \models p$, then $\{a_t, \dots, a_{k-1}\} \cap A \neq \emptyset$.

Proof. Denote by $\pi^L = \langle a^L_1, \dots, a^L_m \rangle$ the leaf path of L in π preceding a_k, i. e., the L leaf path induced by $\langle a_1, \dots, a_{k-1} \rangle$. Say that π^L traverses the L-states $s^L_0, \dots, s^L_m$. Denote by l the index of the L state at the end of π^{past}. Denote by $\pi^L_{l \to m} := \langle a^L_{l+1}, \dots, a^L_m \rangle$ the segment of π^L within π^{future}.

By construction, s^L_l is reached in $s^{\mathcal{F}}$. By prerequisite, $s^L_l \not\models p$. Furthermore, $s^L_m \models p$ because, by construction, a^L_m is the last action preceding a_k that affects L, so s^L_m agrees with s_{k-1} on L, and by prerequisite $s_{k-1} \models p$. Finally, the segment $\pi^L_{l \to m}$ of π^L is an L-path from s^L_l to s^L_m. So s^L_l is a reached-enabling state for p in $s^{\mathcal{F}}$. By Definition 31, A contains at least one action a^L_i from $\pi^L_{l \to m}$, which shows the claim. □

Observe that a reached-enabling set A is essentially a cut between the node sets $\{s^L\}$ and $\{r^L\}$ in L's state space. One could, hence, consider to compute reached-enabling sets via minimum cuts. Observe though that this would be optimizing the wrong objective—we want to minimize, not the number of cut-set transitions, but the number of actions $|A|$ these are labeled with. It is easy to see that optimizing that objective is **NP**-complete.[2] More importantly, reached-enabling sets will need to be computed very frequently—potentially many times on every search state—so their computation must be extremely cheap.

[2]The proof uses a reduction from Hitting Set, where each label subset $\{l_1, \dots, l_n\}$ is represented through a separate path $\pi^L_{s \to r}$ carrying these labels.

An alternative could be a greedy procedure considering only the actions a labeling an outgoing transition $s^L \xrightarrow{a} t^L$ of a reached-enabling state s^L, iteratively collecting and removing one such a until the node sets $\{s^L\}$ and $\{r^L\}$ are disconnected. But this also would be too expensive, requiring repeated reachability checks on the leaf state space. We therefore settle for a trivial approximation, simply collecting all actions a from the transitions $s^L \xrightarrow{a} t^L$. This makes the construction rather inclusive of course, that is, the constructed reached-enabling sets may be large (e. g., we fail to discard unload(T, p_1, B) in the example above). We will show below how more targeted (yet sufficiently cheap) constructions can be designed for goal decoupled states and for fork structures, where enabling p is only relevant if its price decreases strictly.

We are now ready to put the pieces together, and define strong stubborn sets for non-goal decoupled states:

Definition 32 (DSSS: Stars, Non-Goal)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a factoring. Let $s^{\mathcal{F}}$ be a non-goal decoupled state.*

An action set $A_s^{\mathcal{F}}$ is a decoupled strong stubborn set (DSSS) *for $s^{\mathcal{F}}$ if all of the following conditions hold:*

(i) $A_s^{\mathcal{F}}$ contains a decoupled necessary enabling set for $\mathcal{G}$ in $s^{\mathcal{F}}$.

(ii) For all actions $a \in A_s^{\mathcal{F}}$ not reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains a decoupled necessary enabling set for $\mathsf{pre}(a)$ *in $s^{\mathcal{F}}$.*

(iii) For all actions $a \in A_s^{\mathcal{F}}$ reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains all actions a' interfering with a and where $\mathsf{pre}(a) \parallel \mathsf{pre}(a')$.

(iv) For all actions $a \in A_s^{\mathcal{F}}$ reached in $s^{\mathcal{F}}$, and for all L where $\mathsf{pre}(a)[L] \neq \emptyset$, *$A_s^{\mathcal{F}}$ contains a reached-enabling set for* $\mathsf{pre}(a)[L]$ *in $s^{\mathcal{F}}$.*

Items (i) – (iii) of this definition are in obvious correspondence to that for strong stubborn sets in standard search (cf. Chapter 10.1), replacing necessary enabling sets with decoupled necessary enabling sets in (i) and (ii), and replacing applicability with being reached in (ii) and (iii). The new item (iv) is needed to cover enablers for leaf preconditions already reached in $s^{\mathcal{F}}$, as discussed above.

Clearly, the construction of DSSS as per Definition 32 is operational. Viewing the definition as a recursive fixed-point algorithm, all the required constructs—decoupled necessary enabling sets, interfering actions, reached-enabling sets—can be computed in time low-order polynomial in the size of the input task and its leaf state spaces, i. e., in the same size parameters as the computation of the decoupled states themselves.

It remains to prove that Definition 32 is actually correct, i. e., that a DSSS $A_s^{\mathcal{F}}$ for $s^{\mathcal{F}}$ is safe for $s^{\mathcal{F}}$. We do so via extending the proof for strong stubborn sets as given in Chapter 10.1. In particular, we show that for every decoupled plan $\pi^{\mathcal{F}}$ in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$

contains a center action starting a permutation of $\pi^{\mathcal{F}}$ (which by Definition 28 is also a decoupled plan). Note that we actually prove the stronger result that the two decoupled plans induce global plans leading to the *same* goal state, which implies the claim. We include the full proof here, as the underlying analysis is key to understanding why our techniques work.

The proof consists of six successive observations (a) – (e), and a final concluding argument. Observations (a) – (c) correspond to those for the standard setting, namely that (a) there exists a shared action, (b) the first shared action a_k is applicable, (c) a_k does not interfere with the preceding actions. The only major difference here is that, in (b), we can only conclude that a_k is reached in the decoupled state $s^{\mathcal{F}}$, as opposed to being applicable in a state s. Observation (e), moving a_k up front, mirrors observation (d) for the standard setting. Observations (c) and (f), as well as the concluding argument, are new. They are required to: (c) deal with the difference between being reached vs. being applicable; (f) prove that a_k is a center action so we're actually permuting the center path and hence the decoupled plan; (concluding argument) deal with the more complex structure of permutations, i. e., the interplay between decoupled plans and global plans.

Theorem 23. *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}$ *be a solvable non-goal decoupled state, and let* $\pi^{\mathcal{F}}$ *be a decoupled plan for* $s^{\mathcal{F}}$*. Let* $A_s^{\mathcal{F}}$ *be a DSSS for* $s^{\mathcal{F}}$*. Then* $A_s^{\mathcal{F}}$ *contains a center action starting a permutation of* $\pi^{\mathcal{F}}$*.*

Proof. Let π be a global plan for $\pi^{\mathcal{F}}$, and assume, without loss of generality by Lemma 8, that π is past-maximal. Denote $\pi = \langle a_1, \dots, a_n \rangle$. As above, let a_t be the starting action of $\pi^C(\pi^{\mathcal{F}})$ in π, and denote $\pi^{past} := \langle a_1, \dots, a_{t-1} \rangle$ and $\pi^{future} := \langle a_t, \dots, a_n \rangle$. Then the following properties hold:

(a) There must be an action a shared between $A_s^{\mathcal{F}}$ and π^{future}, i. e., $a \in A_s^{\mathcal{F}} \cap \{a_t, \dots, a_n\}$.

First, $\mathcal{G}$ is not reached in $s^{\mathcal{F}}$, as $s^{\mathcal{F}}$ is a non-goal decoupled state. By Definition 32 (i), $A_s^{\mathcal{F}}$ contains a decoupled necessary enabling set A for $\mathcal{G}$ in $s^{\mathcal{F}}$. At least one $a \in A$ must achieve some sub-assignment p of $\mathcal{G}$ that is not reached in $s^{\mathcal{F}}$. Observe that this action a cannot be scheduled on π^{past}, and hence must be on π^{future}: if p pertains to the center, this is trivial; if p pertains to a leaf, this is so because otherwise p would be reached in $s^{\mathcal{F}}$.

Given (a), let a_k be the first shared action, i. e., say that $a_k \in A_s^{\mathcal{F}}$ and $\{a_t, \dots, a_{k-1}\} \cap A_s^{\mathcal{F}} = \emptyset$.

(b) a_k is reached in $s^{\mathcal{F}}$.

This is because, otherwise, by Definition 32 (ii) $A_s^{\mathcal{F}}$ contains a decoupled necessary enabling set A for $\mathsf{pre}(a_k)$ in $s^{\mathcal{F}}$. With the same argument as in (a), applied to $\mathsf{pre}(a_k)$ instead of $\mathcal{G}$, this yields a shared action between $A_s^{\mathcal{F}}$ and π^{future}. That action must precede a_k on π^{future}, in contradiction to a_k being the first shared action.

(c) a_k does not interfere with any of the actions a_i, $t \leq i \leq k-1$, where $\mathsf{pre}(a_k) \parallel \mathsf{pre}(a_i)$.

This is because, if it did, then by Definition 32 (iii) we would have $a_i \in A^{\mathcal{F}}_s$, again in contradiction to a_k being the first shared action.

(d) The leaf precondition of a_k is true at the end of π^{past}, i. e., in $\mathcal{I}[\![\pi^{past}]\!]$.

Assume to the contrary that, for some L, $p := \mathsf{pre}(a_k)[L]$ is not true at the end of π^{past}, i. e., prior to the application of a_k in π. As π is a plan, p is true prior to the application of a_k however. In particular, $k > t$. Furthermore, by (b) and Definition 32 (iv), $A^{\mathcal{F}}_s$ contains a reached-enabling set A for p in $s^{\mathcal{F}}$. We can apply Lemma 9, and get that there exists an action $a_i \in A \subseteq A^{\mathcal{F}}_s$ preceding a_k on π^{future}, again in contradiction.

(e) a_k can be moved to the start of π^{future}. Precisely, $\pi' := \pi^{past} \circ \langle a_k, a_t, \ldots, a_{k-1}, a_{k+1}, \ldots, a_n\rangle$ is a plan for Π.

To see this, consider that, first, a_k is applicable after π^{past}, i. e., a_k is applicable to $\mathcal{I}[\![\pi^{past}]\!]$: by (b) a_k is reached in $s^{\mathcal{F}}$, in particular its center precondition is true at that point; by (d), its leaf precondition is true as well. Second, as both a_k and a_t are applicable in $\mathcal{I}[\![\pi^{past}]\!]$, we must have $\mathsf{pre}(a_k) \parallel \mathsf{pre}(a_t)$, so by (c) a_t does not interfere with a_k, and hence a_k is still applicable in $s[\![\pi^{past} \circ \langle a_t\rangle]\!]$. But then, the same argument applies to a_{t+1} and $s[\![\pi^{past} \circ \langle a_t\rangle]\!]$, so iterating the argument we obtain that, for $t \leq i \leq k-1$, $\mathsf{pre}(a_k) \parallel \mathsf{pre}(a_i)$ and a_i does not interfere with a_k. The claim follows directly from this.

(f) a_k is a center action.

Assume for contradiction that a_k does not affect the center, $a_k \in \mathcal{A}^L_{\not{C}}$ for some L. As π is past-maximal, a_k cannot be moved in front of a_t in π without violating the plan property. However, the plan π' constructed as per (e) does exactly that, in contradiction.

We can now easily construct the desired permutation $\pi^{\mathcal{F}'}$ of $\pi^{\mathcal{F}}$. We construct the underlying center-action sequence $\pi^{C'}$ to be like $\pi^C(\pi^{\mathcal{F}})$, but moving the action a_k, which by (f) is a center action so is part of $\pi^C(\pi^{\mathcal{F}})$, to the front. Consider the plan π' constructed as per (e). The center-action subsequence of π' is $\pi^C(s^{\mathcal{F}}) \circ \pi^{C'}$. So π' consists of that center path, augmented with cheapest goal leaf paths compliant with that center path and ending in cheapest goal leaf states. Therefore, $\pi^{C'}$ induces a decoupled plan $\pi^{\mathcal{F}'}$ for $s^{\mathcal{F}}$, and the permutation π' of π is a global plan for $\pi^{\mathcal{F}'}$. This concludes the argument. □

Corollary 2. *Let Π be a planning task and $\mathcal{F}$ a factoring for Π. Let $s^{\mathcal{F}}$ be a solvable non-goal decoupled state. Let $A^{\mathcal{F}}_s$ be a DSSS in $s^{\mathcal{F}}$. Then $A^{\mathcal{F}}_s$ is safe for $s^{\mathcal{F}}$.*

10.2.3 Goal Decoupled States

Let us now consider goal decoupled states $s^{\mathcal{F}}_G \in \mathcal{S}^{\mathcal{F}}_{\mathcal{G}}$. The only thing that needs to change, relative to the definition of DSSS for non-goal decoupled states, is item (i): How to ensure that there is a shared action, i. e., that at least one action from π^{future} is contained in $A^{\mathcal{F}}_s$?

For non-goal decoupled states, this question was easily answered in terms of a decoupled necessary enabling set, supporting some part p of $\mathcal{G}$ not reached in $s^{\mathcal{F}}$. In a goal decoupled state $\mathcal{G}$, however, such p does not exist. We instead need to capture all ways in which the goal price of some leaf may still be improved. We do so in terms of what we call *goal-price frontier* action sets:

Definition 33 (Goal-Price Frontier Set). *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s^{\mathcal{F}}_G$ *be a goal decoupled state, and let* L *be a leaf factor where* $\mathcal{G}[L] \neq \emptyset$.

We say that an L-transition $s^L \xrightarrow{a} t^L$ *is a* goal-price frontier transition *for* L *in* $s^{\mathcal{F}}_G$ *if*

(i) s^L *is reached in* $s^{\mathcal{F}}_G$ *and* $\mathsf{prices}(s^{\mathcal{F}}_G)[s^L] + \mathsf{cost}(a) < \mathsf{prices}(s^{\mathcal{F}}_G)[t^L]$*; and*

(ii) $s^L \xrightarrow{a} t^L$ *is part of an* L *path* π^L *from* $\mathcal{I}[L]$ *to an L-state* r^L *where* $r^L \models \mathcal{G}[L]$ *and* $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s^{\mathcal{F}}_G)[L]$.

An action set A *is a* goal-price frontier set *for* L *in* $s^{\mathcal{F}}_G$ *if, for every path* π^L *as in (ii), the segment* $\pi^L_{s \to r}$ *of* π^L *starting with* $s^L \xrightarrow{a} t^L$ *contains at least one action from* A.

Here, by $\mathsf{gprice}(s^{\mathcal{F}}_G)[L]$ we denote the goal price of leaf L, so, extending the definition in Chapter 3.2.2: $\mathsf{gprice}(s^{\mathcal{F}}_G)[L] := \min\{\mathsf{prices}(s^{\mathcal{F}}_G)[s^L] \mid s^L \in S^L, s^L \models \mathcal{G}[L]\}$.

Example 11. *Figure 10.1 shows the leaf state space of an example task with a car and a manager, where* $\mathcal{V} = \{job, have\text{-}car, location\}$*, with* $\mathcal{D}(job) = \{worker, manager\}$*,* $\mathcal{D}(have\text{-}car) = \{T, F\}$*, and* $\mathcal{D}(location) = \{A, B\}$*. The actions are as follows:* $\mathcal{A} = \{get\text{-}manager\text{-}job, walk, drive, get\text{-}company\text{-}car, buy\text{-}car\}$*, where the action* $get\text{-}manager\text{-}job$ *changes the value of the variable* job *from* $worker$ *to* $manager$ *for cost* 1*,* $walk$ *and* $drive$ *set position from* A *to* B *at a cost of* 100*, respectively* 1*, where* $drive$ *has the additional precondition* $\{have\text{-}car{=}T\}$*. The actions buy-car and* $get\text{-}company\text{-}car$ *set* $have\text{-}car$ *from* F *to* T *at a cost of* 20000*, respectively* 0*, where* $get\text{-}company\text{-}car$ *has the additional precondition* $\{job{=}manager\}$*. The initial state is* $\mathcal{I} = \{job{=}worker, have\text{-}car{=}F, location{=}A\}$*, the goal is* $\mathcal{G} = \{location{=}B\}$*.*

We set the factoring $\mathcal{F}$ *with a single leaf to* $C = \{job\}$ *and* $L = \{have\text{-}car, location\}$*. The pricing function of the initial decoupled state* $\mathcal{I}^{\mathcal{F}}$ *is shown in the red numbers next to the states in Figure 10.1, where we abbreviate, e. g., state* $\{have\text{-}car = \mathbf{F}, location = \mathbf{A}\}$ *by* FA*. Observe that* $\mathcal{I}^{\mathcal{F}}$ *is a goal decoupled state, where the extracted global plan is* $\langle walk \rangle$ *with a cost of* 100*. The goal-price frontier set for* $\mathcal{I}^{\mathcal{F}}$ *is* $A = \{get\text{-}company\text{-}car\}$*,*

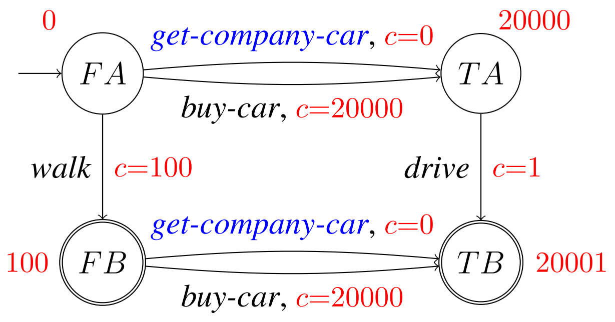


Figure 10.1: Illustration of the leaf state space of the task in Example 11. The actions in the goal-price frontier set are highlighted in blue.

which captures the only way to reduce the goal price. In terms of the definition, both FA and FB qualify as s^L, where the respective t^L are TA and TB.

We use the goal-price frontier sets for all leaf factors that have a goal to ensure progress towards a cheaper *goal decoupled state (replacing the decoupled necessary enabling set for the goal). In our example, the action $get\text{-}company\text{-}car$ in the goal-price frontier set is not applicable because of the unsatisfied center precondition $\{have\text{-}car = T\}$. This leads to the center action $get\text{-}manager\text{-}job$ being added to $A_s^{\mathcal{F}}$. Applying it to $\mathcal{I}^{\mathcal{F}}$ results in the state $s^{\mathcal{F}} := \mathcal{I}^{\mathcal{F}}[\![get\text{-}manager\text{-}job]\!]$ which has a global plan $\langle get\text{-}manager\text{-}job, get\text{-}company\text{-}car, drive\rangle$ with cost* 2.

Similarly as for reached-enabling sets above, we prove that this construction indeed captures all potential enabling actions. From this property, the proof of safety will follow immediately. As the construction of a frontier is more selective than that of reached-enabling sets—targeted at price-improving transitions—its correctness proof is more complicated though.

Lemma 10. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task and $\mathcal{F}$ a factoring. Let $s_G^{\mathcal{F}}$ be a goal decoupled state, and let L be a leaf factor where $\mathcal{G}[L] \neq \emptyset$. Let $\pi^{\mathcal{F}}$ be a decoupled plan for $s_G^{\mathcal{F}}$, and let π be a global plan given $\pi^{\mathcal{F}}$. Let A be a goal-price frontier set for L in $s_G^{\mathcal{F}}$.*

Define a_t, π^{past}, and π^{future} as before, and denote the L-path within π by π^L. If $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s_G^{\mathcal{F}})[L]$, then π^{future} contains an action from A.

Proof. Denote by $s_0^L, \dots, s_m^L$ the leaf states traversed by π^L. As π^L strictly decreases the goal price for L, π^L cannot be fully contained in π^{past}. Denote by l the index of the first action on π^L where a_l^L is on π^{future}.

By construction, s_m^L is a goal leaf state for L, and $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s_G^{\mathcal{F}})[L]$. In particular, $\mathsf{cost}(\pi^L) = \mathsf{cost}(\langle a_1^L, \dots, a_m^L\rangle) < \mathsf{prices}(s_G^{\mathcal{F}})[s_m^L]$, and hence there exists an index $i \geq l$ where $\mathsf{cost}(\langle a_1^L, \dots, a_i^L\rangle) < \mathsf{prices}(s_G^{\mathcal{F}})[s_i^L]$. Let i be the smallest such index. Consider the transition $s_{i-1}^L \xrightarrow{a_i^L} s_i^L$ on π^L. As $i \geq l$, this transition is part of π^{future}.

Because i is the smallest index where $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) < \mathsf{prices}(s_G^{\mathcal{F}})[s_i^L]$, it follows that $\mathsf{prices}(s_G^{\mathcal{F}})[s_{i-1}^L] \leq \mathsf{cost}(\langle a_1^L, \dots, a_{i-1}^L \rangle)$. Observe that, hence, s_{i-1}^L is reached in $s_G^{\mathcal{F}}$, as its price is upper-bounded by a finite value. Further, observe that, adding the cost of a_i^L on both sides of the inequality $\mathsf{prices}(s_G^{\mathcal{F}})[s_{i-1}^L] \leq \mathsf{cost}(\langle a_1^L, \dots, a_{i-1}^L \rangle)$, we get $\mathsf{prices}(s_G^{\mathcal{F}})[s_{i-1}^L] + \mathsf{cost}(a_i^L) \leq \mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle)$. As, by construction, we have $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) < \mathsf{prices}(s_G^{\mathcal{F}})[s_i^L]$, we obtain that $\mathsf{prices}(s_G^{\mathcal{F}})[s_{i-1}^L] + \mathsf{cost}(a_i^L) < \mathsf{prices}(s_G^{\mathcal{F}})[s_i^L]$. Furthermore, $s_{i-1}^L \xrightarrow{a_i^L} s_i^L$ lies on the L path π^L from $\mathcal{I}[L]$ to s_m^L where, as already observed, $s_m^L \models \mathcal{G}[L]$ and $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s_G^{\mathcal{F}})[L]$. Putting these observations together, $s_{i-1}^L \xrightarrow{a_i^L} s_i^L$ is a goal-price frontier transition for L in $s_G^{\mathcal{F}}$.

Therefore, as A is a goal-price frontier set for L in $s_G^{\mathcal{F}}$, we know that the segment $\langle a_i^L, \dots, a_m^L \rangle$ of π^L between s_{i-1}^L and s_m^L must contain an action $a_j^L \in A$. The claim follows as $\langle a_i^L, \dots, a_m^L \rangle$ is fully contained in π^{future}. □

How to compute a goal-price frontier set? Like for reached-enabling sets, this computation is highly time-critical, so we settle for a similar very simple solution: We collect all actions a labeling the goal-price frontier transitions $s^L \xrightarrow{a} t^L$. Note that this is more targeted still than our solution for reached-enabling sets, as the definition of goal-price frontier transitions is more restrictive. For effective implementation, we precompute, within every leaf state space and for every leaf state s^L, the minimum cost $g^*(s^L)$ to reach s^L from $\mathcal{I}[L]$, and the minimum cost $h^*(s^L)$ to reach $\mathcal{G}[L]$ from s^L. Given $s_G^{\mathcal{F}}$, the goal-price frontier transitions then are exactly those $s^L \xrightarrow{a} t^L$ that qualify for Definition 33 (i) which is cheap to test, and where $g^*(s^L) + \mathsf{cost}(a) + h^*(t^L) < \mathsf{gprice}(s_G^{\mathcal{F}})[L]$.

We can now state the definition of DSSS for goal decoupled states very easily:

Definition 34 (DSSS: Stars, Goal)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a factoring. Let $s_G^{\mathcal{F}}$ be a goal decoupled state.*

An action set $A_s^{\mathcal{F}}$ is a decoupled strong stubborn set (DSSS) *for $s_G^{\mathcal{F}}$ if all of the following conditions hold:*

(i) For every L where $\mathcal{G}[L] \neq \emptyset$, $A_s^{\mathcal{F}}$ contains a goal-price frontier set for L in $s_G^{\mathcal{F}}$.

(ii) – (iv) as for non-goal decoupled states, Definition 32, replacing $s^{\mathcal{F}}$ with $s_G^{\mathcal{F}}$.

The proof of safety also is very similar to that for non-goal decoupled states, i. e., that of Theorem 23:

Theorem 24. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a factoring. Let $s_G^{\mathcal{F}}$ be a goal decoupled state for which $\langle\rangle$ is not an optimal decoupled plan, and let $\pi^{\mathcal{F}}$ be a strongly optimal decoupled plan for $s_G^{\mathcal{F}}$. Let $A_s^{\mathcal{F}}$ be a DSSS for $s_G^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ contains a center action starting a permutation of $\pi^{\mathcal{F}}$.*

Proof. The only difference to the proof of Theorem 23 is in argument (a), showing that there is a shared action a contained in both π^{future} and $A_s^{\mathcal{F}}$.

Observe that, with $\langle\rangle$ not being an optimal decoupled plan, $\pi^{\mathcal{F}}$ must strictly decrease the price of the goal for at least one leaf factor L. That is, there must exist L where $\mathcal{G}[L] \neq \emptyset$ and, denoting the L-path within π by π^L, $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s_G^{\mathcal{F}})[L]$. By Definition 34 (i) $A_s^{\mathcal{F}}$ contains a goal-price frontier set A for L in $s_G^{\mathcal{F}}$. We can apply Lemma 10, and get that π^{future} contains an action from A, proving the claim. □

Corollary 3. *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ *be a planning task and* $\mathcal{F}$ *a factoring. Let* $s_G^{\mathcal{F}}$ *be a goal decoupled state for which* $\langle\rangle$ *is not an optimal decoupled plan. Let* $A_s^{\mathcal{F}}$ *be a DSSS for* $s_G^{\mathcal{F}}$. *Then* $A_s^{\mathcal{F}}$ *is safe for* $s_G^{\mathcal{F}}$.

Note that, if there is no goal-price frontier transition for L in $s_G^{\mathcal{F}}$, then $\emptyset$ is a goal-price frontier set for L in $s_G^{\mathcal{F}}$. If this is so for all L on which a goal is defined, then, consequently, $\emptyset$ is a DSSS for $s_G^{\mathcal{F}}$. This is safe, i. e., we can prune all outgoing transitions of $s_G^{\mathcal{F}}$, because in this situation the goal price cannot be improved. In other words, goal-price frontier sets encompass a sufficient criterion for safely stopping the search at $s_G^{\mathcal{F}}$.

10.3 Special Cases Facilitating More Effective Handling

As we have seen, the construction of safe action sets in general star topologies requires to be rather inclusive, collecting potentially many actions which is detrimental to pruning power. This is specifically so for non-goal decoupled states (and hence most of the search). The main difficulty is the handling of reached leaf conditions p, i. e., reached-enabling sets. As p is already fully reached, in contrast to necessary enabling sets we cannot focus on some small part of p that is still open.

A major remedy is the consideration of more restricted topologies, namely fork and inverted-fork topologies. These are practically relevant, since they can be easily identified and occur regularly in planning domains. The key property of these topologies is *monotonicity* of the pricing function:

- *Positive monotonicity: In a fork topology, the price of any one leaf state* s^L *can only decrease along a search path.*

  This is because the center has no precondition or effect on the leaves, so whenever a leaf path becomes center-compliant (all its preconditions on C have been provided), it remains so forever after.

- *Negative monotonicity: In an inverted-fork topology, the price of any one leaf state* s^L *can only increase along a search path.*

  This is because the only cross-factor interaction consists in preconditions of the center on the leaves. So anything the center does can only pose more requirements on the leaves, and whenever a leaf path becomes non-center-compliant, it remains so forever after.

Positive monotonicity is useful because, for reached leaf conditions p, the only future event of interest is one that strictly decreases the price of some leaf state that satisfies p. Hence necessary enablers for such p can now be identified in a more targeted manner, similar to the goal-price frontier sets above. Negative monotonicity is useful because, if a leaf condition p is not reached, then this will remain so and anything requiring p can be pruned.

For the sake of conciseness, we give summary presentations only, deferring some details, in particular the full proofs, to Appendix B.1. We consider fork topologies in Chapter 10.3.1 and inverted-fork topologies in Chapter 10.3.2. In Chapter 10.3.3, we show how to combine the respective techniques, as special-case treatments for individual fork/inverted-fork leaves as part of general star topologies.

10.3.1 Forks

As previously hinted, the major ingredient of our treatment for fork topologies is a concept capturing how the price of a reached leaf condition p may be improved. This being the key construction here, we introduce and discuss it in detail. The concept is similar, but not identical to, the definition of goal-price frontier sets:

Definition 35 (Fork-Price Frontier Set)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task and* $\mathcal{F}$ *a fork factoring. Let* $s^{\mathcal{F}}$ *be a decoupled state, let* L *be a leaf factor, and let* p *be a partial assignment to* L *reached in* $s^{\mathcal{F}}$*.*

We say that an L*-transition* $s^L \xrightarrow{a} t^L$ *is a* fork-price frontier transition *for* p *in* $s^{\mathcal{F}}$ *if*

(i) s^L *is reached in* $s^{\mathcal{F}}$, $\mathsf{prices}(s^{\mathcal{F}})[s^L] + \mathsf{cost}(a) < \mathsf{prices}(s^{\mathcal{F}})[t^L]$*, and* $\mathsf{center}(s^{\mathcal{F}}) \not\models \mathsf{pre}(a)[C]$*; and*

(ii) $s^L \xrightarrow{a} t^L$ *is part of a simple* L *path* π^L *from* $\mathcal{I}[L]$ *to an* L *state* r^L *where* $r^L \models p$ *and* $\mathsf{cost}(\pi^L) < \mathsf{prices}(s^{\mathcal{F}})[r^L]$*.*

An action set A *is a* fork-price frontier set *for* p *in* $s^{\mathcal{F}}$ *if, for every path* π^L *as in (ii), the segment* $\pi^L_{s \to r}$ *of* π^L *starting with* $s^L \xrightarrow{a} t^L$ *contains at least one action from* A*.*

There are three differences to goal-price frontier sets (Definition 33): (a) the additional requirement in (i) that $\mathsf{center}(s^{\mathcal{F}}) \not\models \mathsf{pre}(a)[C]$; (b) that π^L is required to be simple in (ii), i. e., visit each L-state at most once; (c) the weaker condition $\mathsf{cost}(\pi^L) < \mathsf{prices}(s^{\mathcal{F}})[r^L]$ in (ii). Differences (a) and (b) are sound in fork topologies but not in general, so are benefits of the focus on forks. Difference (c) is due to the difference between non-goal vs. goal decoupled states: whereas on the latter the goal price must be reduced globally, on non-goal decoupled states the price reduction will pertain only to individual leaf states r^L. On goal decoupled states $s^{\mathcal{F}}_G$ in fork topologies, we can get rid of difference (c), i. e., instead of $\mathsf{cost}(\pi^L) < \mathsf{prices}(s^{\mathcal{F}})[r^L]$ we can use $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s^{\mathcal{F}}_G)[L]$ as in Definition 33. We refer to the variant as *fork-goal-price frontier sets*.

As before, we show the correctness of our construction, i. e., that it indeed captures all potential enabling actions. The proof is similar to that for goal-price frontier sets in Lemma 10, with simple additions tackling the differences (a) – (c):

Lemma 11. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state, let L be a leaf factor, and let p be a partial assignment to L reached in $s^{\mathcal{F}}$. Let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$, and let π be a global plan given $\pi^{\mathcal{F}}$. Let A be a fork-price frontier set for p in $s^{\mathcal{F}}$.*

Define a_t, π^{past}, and π^{future} as before. If there exists $k > t$ such that, denoting by $\langle a^L_1, \dots, a^L_i\rangle$ and $s^L_0, \dots, s^L_i$ the L-actions, respectively states, in π prior to a_k, we have $\mathsf{cost}(\langle a^L_1, \dots, a^L_i\rangle) < \mathsf{prices}(s^{\mathcal{F}})[s^L_i]$ and $s^L_i \models p$, then $\{a_t, \dots, a_{k-1}\} \cap A \neq \emptyset$.

Proof (sketch). Similarly to the proof of Lemma 10, we consider the smallest index j such that a^L_j is on π^{future} and $\mathsf{cost}(\langle a^L_1, \dots, a^L_j\rangle < \mathsf{prices}(s^{\mathcal{F}})[s^L_j]$, and we prove that $s^L_{j-1} \xrightarrow{a^L_j} s^L_j$ is a fork-price frontier transition for p in $s^{\mathcal{F}}$, from which the claim follows directly.

Difference (c) does not change anything as it is merely a different condition on the L-path end state, s^L_i in our case here, provided by prerequisite. It remains to prove that (a) $\mathsf{center}(s^{\mathcal{F}}) \not\models \mathsf{pre}(a^L_j)[C]$ and (b) π^L is simple.

Regarding (a), as a^L_j affects L, in a fork topology a^L_j cannot affect the center. Further more, from $\mathsf{prices}(s^{\mathcal{F}})[s^L_{j-1}] + \mathsf{cost}(a^L_j) < \mathsf{prices}(s^{\mathcal{F}})[s^L_j]$ which is proved exactly as in Lemma 10, we can therefore conclude that a^L_j cannot be reached in $s^{\mathcal{F}}$. We do know, however, that s^L_{j-1} is reached in $s^{\mathcal{F}}$. As the only non-L precondition of a^L_j must be on C, (a) follows.

Regarding (b), in a fork topology, any cheapest compliant leaf path is simple, because without center preconditions nor effect on the leaf there is no reason to visit the same leaf state twice. □

A similar claim holds for fork-goal-price frontier sets A, so if it holds that $\mathsf{cost}(\pi^L) < \mathsf{gprice}(s^{\mathcal{F}}_G)[L]$, then π^{future} contains an action from A. The proof is a straightforward combination of those for Lemmas 10 and 11, essentially applying arguments (a) and (b) from the latter as an extension of the former to tackle the definition differences (a) and (b) explained above.

From a practical perspective, in difference to goal-price frontier sets, item (ii) in Definition 35 is moot. First, we cannot exploit (b) that π^L must be simple, as the question whether there exist vertex-disjoint paths from $\mathcal{I}[L]$ to s^L and from t^L to r^L is the 2-vertex-disjoint paths problem for directed graphs, which is known to be **NP**-complete [Fortune et al., 1980]. Second, in the weaker condition in (c), $\mathsf{cost}(\pi^L) <$

$\mathsf{prices}(s^{\mathcal{F}})[r^L]$, the value $\mathsf{prices}(s^{\mathcal{F}})[r^L]$ we are comparing the cost of π^L to is not a constant, i. e., depends on π^L. One could check the condition through a search below t^L, or through precomputing all-pairs shortest paths on the leaf state spaces, but given the associated overhead neither is promising. So we settle for the naïve approximation collecting all actions a labeling the fork-price frontier transitions $s^L \xrightarrow{a} t^L$. For fork-goal-price frontier sets, of course, we can apply the same filter as before, i. e., collect a only where $g^*(s^L) + \mathsf{cost}(a) + h^*(t^L) < \mathsf{gprice}(s^{\mathcal{F}}_G)[L]$.

We are now ready to define decoupled strong stubborn sets for fork topologies. For conciseness, we subsume both, the non-goal and goal cases, into a single definition:

Definition 36 (DSSS: Forks)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a decoupled state.*

An action set $A^{\mathcal{F}}_s$ is a fork-decoupled strong stubborn set (FDSSS) *for $s^{\mathcal{F}}$ if all of the following conditions hold:*

(i) If $\mathcal{G}$ is not reached in $s^{\mathcal{F}}$, then $A^{\mathcal{F}}_s$ contains a decoupled necessary enabling set for $\mathcal{G}$ in $s^{\mathcal{F}}$.

If $\mathcal{G}$ is reached in $s^{\mathcal{F}}$, then, for every L where $\mathcal{G}[L] \neq \emptyset$, $A^{\mathcal{F}}_s$ contains a fork-goal-price frontier set for L in $s^{\mathcal{F}}_G$.

(ii) For all actions $a \in A^{\mathcal{F}}_s$ not reached in $s^{\mathcal{F}}$, $A^{\mathcal{F}}_s$ contains a decoupled necessary enabling set for $\mathsf{pre}(a)$ *in $s^{\mathcal{F}}$.*

(iii) For all center actions $a^C \in A^{\mathcal{F}}_s \cap \mathcal{A}^C$ reached in $s^{\mathcal{F}}$, $A^{\mathcal{F}}_s$ contains all actions a' interfering with a^C and where $\mathsf{pre}(a^C) \parallel \mathsf{pre}(a')$.

(iv) For all leaf actions $a^L \in A^{\mathcal{F}}_s \cap \mathcal{A}^L$ reached in $s^{\mathcal{F}}$, $A^{\mathcal{F}}_s$ contains a fork-price frontier set for $\mathsf{pre}(a^L)[L]$ *in $s^{\mathcal{F}}$.*

Recall here that, for fork and inverted-fork topologies, the center actions $\mathcal{A}^C$ and the leaf actions $\mathcal{A}^L$ are disjoint, $\mathcal{A}^C \cap \mathcal{A}^L = \emptyset$. Item (i) is an obvious combination handling non-goal $s^{\mathcal{F}}$ like Definition 32, and handling goal $s^{\mathcal{F}}$ like Definition 34 but with fork-goal-price frontier sets. Item (ii) is as before, item (iii) as well except that, as it turns out, we need to select interfering actions only for center actions $a^C \in A^{\mathcal{F}}_s \cap \mathcal{A}^C$. Item (iv) replaces reached-enabling sets with fork-price frontier sets as advertised, doing so only for leaf actions as center actions do not have leaf preconditions in a fork.[3]

[3]We remark that, in the latter, a fork-price frontier set needs to be collected even if $\mathsf{pre}(a^L)[L] = \emptyset$. This is because a^L may need to be on π^{future} to correct a detrimental effect of some other, not-yet-reached, leaf action on π^{future}. In the special case where L consists of a single variable though, the latter cannot happen, so in that special case a fork-price frontier set is not required for $\mathsf{pre}(a^L)[L] = \emptyset$. We exploit that special case in our implementation.

As mentioned before, our safety proof makes use of a notion of past-maximality different from that used for star topologies (Definition 29). We defer the details to Appendix B.1. Essentially, π is *fork-past-maximal* if, whenever on π^{future} we apply a leaf action a_k to a leaf state s_i^L already reached in $s^{\mathcal{F}}$, then we do so on a leaf path achieving s_i^L at a price cheaper than that in $s^{\mathcal{F}}$. Intuitively, this makes sense as, otherwise, we could have as well used a compliant path for s_i^L in π^{past} already. Formally, every global plan can be transformed into a fork-past-maximal global plan of equal or cheaper cost.

Theorem 25. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state for which $\langle\rangle$ is not an optimal decoupled plan, and let $\pi^{\mathcal{F}}$ be a strongly optimal decoupled plan for $s^{\mathcal{F}}$. Let $A_s^{\mathcal{F}}$ be an FDSSS for $s^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ contains a center action starting a permutation of $\pi^{\mathcal{F}}$.*

Proof (sketch). The proof is similar to that of Theorems 23 and 24. There is one major difference.

As before, we consider a global plan π for $\pi^{\mathcal{F}}$. We assume without loss of generality that π is fork-past-maximal. We denote a_t, π^{past}, and π^{future} as before.

The argument (a) that there must be an action a shared between $A_s^{\mathcal{F}}$ and π^{future} is the same as in the proof of Theorem 23 in the case of non-goal $s^{\mathcal{F}}$. For goal $s^{\mathcal{F}}$, it is the same as in the proof of Theorem 24, replacing Lemma 10 with the variant of Lemma 11 for fork-goal-price frontier sets.

Denoting by a_k the first shared action, that (b) a_k is reached in $s^{\mathcal{F}}$ follows with the same argument as in the proof of Theorem 23.

The one major difference is that we have to prove next that *(c) a_k is a center action*, because Definition 36 (iii) includes interfering actions for center actions $a^C \in A_s^{\mathcal{F}} \cap \mathcal{A}^C$ only.

Once (c) is shown, the same argument as in Theorem 23 (c) shows that a_k does not interfere with any of the actions a_i, $t \leq i \leq k-1$. The same argument as in Theorem 23 (e) shows that a_k can be moved to the start of π^{future}. The same concluding argument as in Theorem 23 shows the claim.

So, how do we show (c)? If a_k is a leaf action, $a_k \in \mathcal{A}^L$, then by Definition 36 (iv) $A_s^{\mathcal{F}}$ contains a fork-price frontier set A for $p := \mathsf{pre}(a_k)[L]$ in $s^{\mathcal{F}}$. As a_t is a center action, given the fork topology we know that $a_k \neq a_t$ and hence $k > t$. Hence fork-past-maximality applies, showing that $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_i^L]$ where $\langle a_1^L, \dots, a_i^L \rangle$ and $s_0^L, \dots, s_i^L$ denote the L actions, respectively states, in π prior to a_k. Applying Lemma 11 yields that $\{a_t, \dots, a_{k-1}\} \cap A \neq \emptyset$, in contradiction to a_k being the first shared action, concluding the proof. $\square$

Corollary 4. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state for which $\langle\rangle$ is not an optimal decoupled plan. Let $A_s^{\mathcal{F}}$ be an FDSSS in $s^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ is safe for $s^{\mathcal{F}}$.*

10.3.2 Inverted Forks

Our modifications for inverted-fork topologies are much simpler than those for forks. This has two main reasons:

- On goal decoupled state $s^{\mathcal{F}}_G$, we do not need to do anything at all. This is because, with negative monotonicity—pricing functions increasing monotonically along search paths—the prices below $s^{\mathcal{F}}_G$ can never improve. So there is no need to search below $s^{\mathcal{F}}_G$, and we do not need to define any pruning techniques for that case (in other words: just set $A^{\mathcal{F}}_s := \emptyset$).
- On non-goal decoupled states, the intricacies of enabling reached leaf conditions p remain essentially the same as in general star toplogies—all we know is that p is false after π^{past} but may need to be made true in the future. Hence we need to include reached-enabling sets just like for general star topologies.

Given this, our modifications are confined to discarding any actions a from $A^{\mathcal{F}}_s$ whose leaf preconditions aren't reached in $s^{\mathcal{F}}$—which preserves safety simply because such a can never occur on π^{future}. For the sake of completeness, let us state the latter formally:

Proposition 9. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ an inverted-fork factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state. Let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$, and let π be a global plan given $\pi^{\mathcal{F}}$. Let a be an action where $\mathsf{pre}(a)[\mathcal{V} \setminus C]$ is not reached in $s^{\mathcal{F}}$. Define a_t, π^{past}, and π^{future} as before. Then a does not occur on π^{future}.*

Proof. Assume to the contrary that a does occur on π^{future}. There must be at least one leaf factor where the precondition of a on that factor is not reached in $s^{\mathcal{F}}$. Let L be such a leaf factor. Then $\mathsf{pre}(a)[L]$ must be supported, in π, by a $\pi^C(s^{\mathcal{F}}) \circ \pi^C(\pi^{\mathcal{F}})$-compliant L path π^L, ending in s^L where $s^L \models \mathsf{pre}(a)[L]$. Given the inverted-fork structure, the actions in π^L have preconditions on L only. Therefore, π^L also is $\pi^C(s^{\mathcal{F}})$-compliant. But then, $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$ in contradiction. $\square$

The simplified definitions are as follows. First, necessary enabling sets need to worry about the center only, as an unreached leaf condition is unsolvable; and actions whose leaf preconditions are not reached can be discarded:

Definition 37 (Center-Necessary Enabling Set)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ an inverted-fork factoring. Let $s^{\mathcal{F}}$ be a decoupled state, and let p be a partial variable assignment to C not reached in $s^{\mathcal{F}}$.*

An action set A is a center-necessary enabling set *for p in $s^{\mathcal{F}}$ if there exists $v \in \mathsf{vars}(p)$ such that $\mathsf{center}(s^{\mathcal{F}})[v] \neq p[v]$, and $A = \{a \in \mathcal{A} \mid \mathsf{eff}(a)[v] = p[v], \mathsf{pre}(a)[\mathcal{V} \setminus C]$ is reached in $s^{\mathcal{F}}\}$.*

For reached-enabling sets, nothing changes. This is because, cf. Definition 31, these catch actions from leaf paths π^L starting in reached-enabling states s^L, one of whose properties is that s^L is reached in $s^{\mathcal{F}}$. But then, as an L leaf path has preconditions only on L itself, all actions on π^L are already reached, and there are no unreached leaf preconditions to prune.

The definition of DSSS now mirrors that of DSSS for general star topologies:

Definition 38 (DSSS: Inverted Forks)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ an inverted-fork factoring. Let $s^{\mathcal{F}}$ be a non-goal decoupled state where $\mathcal{G}[\mathcal{V} \setminus C]$ is reached in $s^{\mathcal{F}}$.*

An action set $A_s^{\mathcal{F}}$ is an inverted-fork decoupled strong stubborn set (IFDSSS) *for $s^{\mathcal{F}}$ if all of the following conditions hold:*

(i) $A_s^{\mathcal{F}}$ contains a center-necessary enabling set for $\mathcal{G}[C]$ in $s^{\mathcal{F}}$.

(ii) For all center actions $a \in A_s^{\mathcal{F}}$ not reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains a center-necessary enabling set for $\mathsf{pre}(a)[C]$ in $s^{\mathcal{F}}$.

(iii) For all actions $a \in A_s^{\mathcal{F}}$ reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains all actions a' interfering with a where $\mathsf{pre}(a) \parallel \mathsf{pre}(a')$ and $\mathsf{pre}(a')[\mathcal{V} \setminus C]$ is reached in $s^{\mathcal{F}}$.

(iv) For all actions $a \in A_s^{\mathcal{F}}$ reached in $s^{\mathcal{F}}$, and for all L where $\mathsf{pre}(a)[L] \neq \emptyset$, $A_s^{\mathcal{F}}$ contains a reached-enabling set for $\mathsf{pre}(a)[L]$ in $s^{\mathcal{F}}$.

Compared to DSSS for general star topologies (Definition 32), we can restrict focus to decoupled states $s^{\mathcal{F}}$ where the leaf goal is reached, because otherwise $s^{\mathcal{F}}$ is unsolvable. We use center-necessary enabling sets instead of decoupled necessary enabling sets. We discard actions whose leaf preconditions are not reached, and given this, unreached leaf actions are never included, so only unreached center actions need to be supported.

Safety of this construction follows immediately from safety of DSSS for general star topologies, together with Proposition 9:

Corollary 5. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ an inverted-fork factoring. Let $s^{\mathcal{F}}$ be a solvable non-goal decoupled state. Let $A_s^{\mathcal{F}}$ be an IFDSSS in $s^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ is safe for $s^{\mathcal{F}}$.*

Proof. The proof of Theorem 23 remains intact almost exactly as stated. The only addition we need to make is in arguments (a) – (c), where we invoke Proposition 9 to show that the shared action identified on π^{future} (a,b), respectively the actions preceding a_k on π^{future} (c), do not have unreached leaf preconditions. $\square$

We remark that, in the special case of inverted-fork topologies where all leaf factors consist of a single state variable only, one can modify Definition 38, using only center actions in item (iii), and replacing item (iv) with a rule that includes center actions a' whose leaf preconditions compete with those of a center action a already included into $A_s^{\mathcal{F}}$. We use this more targeted definition as an optimization in our implementation. Formal details are provided in Definition 69 in Appendix B.1. Although this special case is interesting from a theoretical point of view, we did not observe a major impact in most of the planning domains that we ran our analysis on, as we shall see in Chapter 10.5.

10.3.3 Fork/Inverted-Fork Leaves in General Star Topologies

One can also combine the observations just made for fork and inverted-fork topologies, to obtain an enhanced variant of DSSS for general star topologies, exploiting the properties of individual fork/inverted-fork leaves. This is important as, in practice, it often happens that even in general star topologies there are (inverted-)fork leaves that can be treated more efficiently by the special-case algorithms.

Recall that fork/inverted-fork leaves behave like the leaves in the respective structure. That is, a fork leaf is one on which the center has neither precondition nor effect; and an inverted-fork leaf is one on which the center has no effect, and that has no precondition on the center. If such leaves occur in a general star topology, then we do not get positive/negative monotonicity at the global level, but we do get it for each of these individual leaves. This can be exploited in the same manner as specified above.

We define *enhanced decoupled strong stubborn sets (EDSSS)* towards this end. As the details are simple yet cumbersome to spell out formally, we defer them to Appendix B.1. Here is a summary:

(i) *Goal enablers:* For non-goal $s^{\mathcal{F}}$, use decoupled necessary enabling sets. For goal $s^{\mathcal{F}}$, use goal-price frontier sets for non-fork leaves, and use fork-goal-price frontier sets for fork leaves. Discard actions with unreached inverted-fork preconditions.

(ii) *Unreached action enablers:* Use decoupled necessary enabling sets. Discard actions with unreached inverted-fork preconditions.

(iii) *Reached action interference:* As before, but only for non-fork-leaf actions, and discarding actions with unreached inverted-fork preconditions.

(iv) *Reached action enablers:* For non-fork-leaf actions, use reached-enabling sets. For fork-leaf actions, use fork-price frontier sets.

The proof of safety mirrors that of Theorem 23, the only major addition being a new proof item showing that a_k must be a non-fork action. The argument used for that purpose is the same one used in Theorem 25 to prove that a_k must be a center action. The notions of past-maximality and fork-past-maximality are combined in these arguments, in the sense of assuming these properties for the individual leaves L as appropriate.

10.4 Exponential Separation from Base Methods

Before proceeding to the empirical part of our research, we look into the theoretical properties of the power of DSSS. We only give the proof sketch in this section, the full proof is given in Appendix B.1.2.

We have already seen that decoupled search and SSS are complementary in that each is exponentially separated from the other, see Chapter 6.2. Trivially, DSSS is exponentially separated from each of decoupled search and SSS, simply because DSSS naturally generalizes each of these components, so we can use the same task families to show this. As a much stronger testimony to the power of DSSS, there are cases where it is exponentially separated from *both* its components:

Theorem 26. *DSSS is exponentially separated from both, decoupled search and SSS.*

Two suitable families $\{\Pi^n\}$ arise from simple modifications of our logistics example. First, say we have M trucks and $N * M$ packages, where each truck T_i is associated with a group of N packages that only T_i can transport. The number of reachable decoupled states is exponential in M because all trucks must be in the center factor. The SSS-pruned reachable standard state space has size exponential in N because including an (un)load action into A_s necessitates, due to interference via the truck move as above, to include *all* applicable (un)load actions for the respective package group. However, *in decoupled search with DSSS pruning, there are only* $M + 1$ *reachable states*. This is because the two sources of pruning power combine gracefully. Decoupling gets rid of the blow-up in N (the packages within a group become independent leaves), while DSSS gets rid of the blow-up in M (only a single truck is committed to at a time).

In our second example, DSSS even is exponentially *more* than the sum of its components: stubborn sets have exponentially more impact on the decoupled search space than on the standard one. Say we have N packages and M trucks (where every truck may transport every package). Then decoupled search blows up in M, and SSS does not do anything because any package may require any truck. Applying DSSS to decoupled search, no truck move is pruned in $\mathcal{I}^{\mathcal{F}}$. However, after applying any one drive(T_i, A, B) action, all package prices are the cheapest possible ones, the goal-price frontier is empty, and DSSS stops the search. So, again, there are only $M + 1$ reachable states. As we shall see next, similar phenomena seem to occur in the standard Logistics benchmarks.

10.5 Experimental Evaluation

We implemented all variants of decoupled strong stubborn sets in our decoupled search planner based on Fast Downward [Helmert, 2006b]. The general settings, benchmarks, algorithms, and heuristics used in the evaluation are as described in Chapter 7.2. We focus on optimal planning, where stubborn-sets pruning has been most commonly employed. For satisficing planning, the pruning does just not make sense, since typical approaches here never generate all successors of an expanded state, anyway. The overhead of the optimality-preserving stubborn-set computation then does not pay off. For proving unsolvability, we did not observe any interesting results, since on the subset of domains where a star factoring can be computed stubborn sets yield only very little pruning. We use the IA factoring strategy throughout. The source code and evaluation data are publicly available [Gnad, 2021a].

The definitions in Chapter 10.2 do not fully specify *how* to compute decoupled strong stubborn sets, but rather state the properties such sets need to have. In particular, there are several ways to compute (1) decoupled necessary enabling sets, (2) reached-enabling sets, and (3) goal-price frontier sets. For (1), we stick to the ordering given in Definition 30, namely we first choose unreached leaf variable assignments, then unreached center assignments. If this still does not result in a unique choice, we stick to the variable-ordering of Fast Downward, which has also been used in prior work on strong stubborn sets [Wehrle and Helmert, 2014]. In combination with our use of action interference, where only interfering actions with *agreeing* preconditions are added to the DSSS, our selection strategy corresponds to the "full/mutex + FD" strategy of Wehrle and Helmert [2014]. For the standard search variant of strong stubborn sets pruning that we compare to (S3), we use the strategy "full-syntactic-SSS-EC", which is the strongest variant of SSS pruning that is based on the same approach of computing stubborn sets. Recently, a computation of stubborn sets based on atoms, instead of actions, has been proposed [Röger et al., 2020]. We do not include that in the comparison, since it could potentially be adapted to decoupled search, too. Regarding (2) and (3), we use the approximations as described in Chapter 10.2.2 and Chapter 10.2.3, respectively.

The special case definitions from Chapter 10.3 also leave some freedom, namely in how to compute fork(-goal)-price frontier sets and center-necessary enabling sets. For the latter, we stick to the aforementioned static selection of yet unreached assignments. Regarding the former, we implemented the algorithms outlined in Chapter 10.3.1.

We show results for two different configurations of decoupled strong stubborn sets, the plain variant implementing the definitions from Chapter 10.2 (DS3), and a variant that uses all optimizations from Chapter 10.3 whenever they are applicable ($\mathrm{DS3}^o$).

As a preview on the empirical results, it turns out that just like in standard search [Alkhazraji et al., 2012; Wehrle et al., 2013; Wehrle and Helmert, 2014], there exist domains where stubborn sets pruning does not result in any reduction of the search space size. Yet, computing a (decoupled) strong stubborn set for every state expanded in the search incurs a significant runtime overhead. Therefore, we implement a simple *safety belt* mechanism to disable the pruning in planning instances where stubborn sets are not effective. To do so, we switch the stubborn sets off if after the first 1000 expanded states less than 1% of the transitions have been pruned (a similar mechanism has been used in Torralba and Hoffmann [2015]). We keep the safety belt rather permissive to only disable the pruning in instances where effectively no transitions are being pruned. This allows for a detailed analysis on how much pruning is needed to make up for the computational overhead of computing a (decoupled) strong stubborn set in every search state. Expanding 1000 states usually takes only little time, so we expect the total overhead to be small if the pruning does not pay off. On the other hand, if only a small fraction of the transitions could be pruned during the first 1000 expansions, chances are high that it remains like this throughout the entire search. We shall see in the next two sections that both conjectures are valid in almost all domains.

In Figure 10.2 we report coverage results (number of instances solved). For blind search, $\mathrm{DS3}$ mostly inherits the highest coverage of its two components. Out of the 28 domains listed in the table, there are only 7 in which the combination is worse than DS, and only 4 where coverage is lower than for $\mathrm{S3}$. While this sounds like a negative result, observe that also $\mathrm{S3}$ drops in coverage compared to Base in 11 domains. On the positive side, $\mathrm{DS3}$ performs better than both baselines in Logistics, Rovers, and Woodworking.

Enabling the optimizations ($\mathrm{DS3}^o$) increases coverage over $\mathrm{DS3}$ in Elevators, Logistics, and Openstacks by up to 2 instances. This somewhat disappointing result is not mainly due to using the IA factoring strategy, which often produces fork or inverted-fork factorings, as we observed in Figure 7.2 in Chapter 7.3. Even when using the factorings obtained by Fork or IFork the results are the same.

Like in explicit-state blind search, where stubborn sets pruning only improves in total coverage when enabling the safety-belt mechanism, decoupled strong stubborn sets pruning has a significant per-state overhead that does not pay off if there is little pruning.

When using A^* with $h^{\text{LM-cut}}$, the overall picture is similar. In this setting, we also compare to the state-of-the-art planners symbolic bidirectional search (SBD) and Complementary ($\mathrm{C2}$). In general, the results look better for stubborn sets pruning in both settings. We conjecture that the reason for this is the non-trivial per-state runtime of the $h^{\text{LM-cut}}$ heuristic, so the relative overhead to compute the strong stubborn set is lower per state. Still, even the best decoupled strong stubborn sets configuration with the optimizations and the safety belt only improves over both baselines in a single domain (Tetris) by one instance. However, it quite reliably performs as good as its best component, similar to the blind search setting.

In the scatter plots in Figure 10.3, we see that enabling strong stubborn sets pruning leads to very similar behaviour in explicit-state search and decoupled search. In both blind search and A^* with $h^{\text{LM-cut}}$, for both state representations the pruning is able to re-

			Blind Search							A* with $h^{\text{LM-cut}}$								
Domain	#	#$\mathcal{F}$	B	S3	S3$_\text{s}$	DS	DS3	DS3^o	DS3^o_s	B	S3	S3$_\text{s}$	DS	DS3	DS3^o	DS3^o_s	SB	C2
Agricola	20	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**5**	0
Childsnack	20	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**4**	0
DataNet	20	20	7	5	7	**9**	**9**	**9**	**9**	12	12	12	**14**	**14**	13	**14**	13	13
Depots	22	22	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	5	**7**
Driverlog	20	20	7	7	7	**11**	**11**	**11**	**11**	13	13	13	13	13	13	13	12	**15**
Elevators	30	30	13	12	12	**16**	14	15	**16**	22	22	22	23	23	23	23	**25**	**25**
Floortile	40	40	**2**	**2**	**2**	**2**	**2**	**2**	**2**	13	13	13	9	9	9	9	**34**	28
Freecell	80	42	**3**	0	2	0	0	0	0	2	1	2	2	0	0	2	2	**3**
GED	20	20	**15**	**15**	**15**	**15**	**15**	**15**	**15**	15	15	15	15	15	15	15	19	**20**
Grid	5	5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	2	2	2	2	2	2	2	2	**3**
Logistics	63	63	12	12	12	25	26	**28**	**28**	26	27	27	**36**	35	35	**36**	24	28
Miconic	150	145	45	40	45	**46**	**46**	**46**	**46**	**136**	**136**	**136**	135	135	135	135	107	98
Mprime	35	6	**6**	**6**	**6**	4	4	4	4	**6**	**6**	**6**	4	4	4	4	**6**	**6**
NoMystery	20	20	8	8	8	**20**	14	14	19	14	14	14	**20**	18	18	**20**	14	**20**
Openstacks	80	50	**25**	22	22	20	19	20	20	24	22	22	20	20	20	20	**50**	40
Org-Split	20	16	**6**	4	5	3	2	2	2	**11**	9	9	9	7	7	7	5	5
ParcPrinter	20	13	1	**13**	**13**	3	6	6	6	4	**13**	**13**	7	**13**	**13**	**13**	2	4
PSR	50	48	47	47	47	**48**	**48**	**48**	**48**	47	47	47	**48**	**48**	**48**	**48**	**48**	**48**
Rovers	40	40	5	7	7	7	**9**	**9**	**9**	7	9	9	8	9	9	9	**14**	13
Satellite	36	36	5	**6**	**6**	5	**6**	**6**	**6**	7	11	11	9	**12**	11	11	8	9
Scanalyzer	30	9	**6**	3	4	3	3	3	3	5	3	3	5	4	4	5	**6**	**6**
Tetris	17	13	4	4	**6**	5	4	3	5	5	4	4	4	4	4	5	2	**10**
Tidybot	30	30	14	5	8	**16**	5	5	**16**	**18**	17	**18**	17	17	17	**18**	7	**18**
TPP	30	29	**5**	**5**	**5**	**5**	**5**	**5**	**5**	5	5	5	5	5	5	5	7	**14**
Transport	59	30	**15**	13	14	**15**	13	13	**15**	14	14	14	14	14	14	14	**16**	**16**
Trucks	30	27	**5**	4	**5**	4	4	4	4	9	9	9	10	10	10	10	10	**12**
Woodwork	30	26	6	10	10	7	**11**	**11**	**11**	14	**23**	**23**	17	20	20	20	19	17
Zenotravel	20	20	8	7	7	**9**	**9**	**9**	**9**	**13**	**13**	**13**	**13**	**13**	**13**	12	10	**13**
Other	593	36	10	10	10	10	10	10	10	11	11	11	11	11	11	11	11	11
Total	1630	896	285	272	290	313	300	303	**324**	462	478	480	477	482	480	488	487	**502**

Figure 10.2: Coverage data (number of solved tasks) in optimal planning, on instances where IA does not abstain. #$\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage (separately for blind search and A* with $h^{\text{LM-cut}}$) in **bold face**.

duce search space size by several orders of magnitude. Where this happens, there is also a significant reduction in runtime. In general, there is a significant runtime overhead, though. This, however, can effectively be tackled with the safety-belt mechanism.

In Figure 10.4, in the two left-most columns we see that for both blind search and

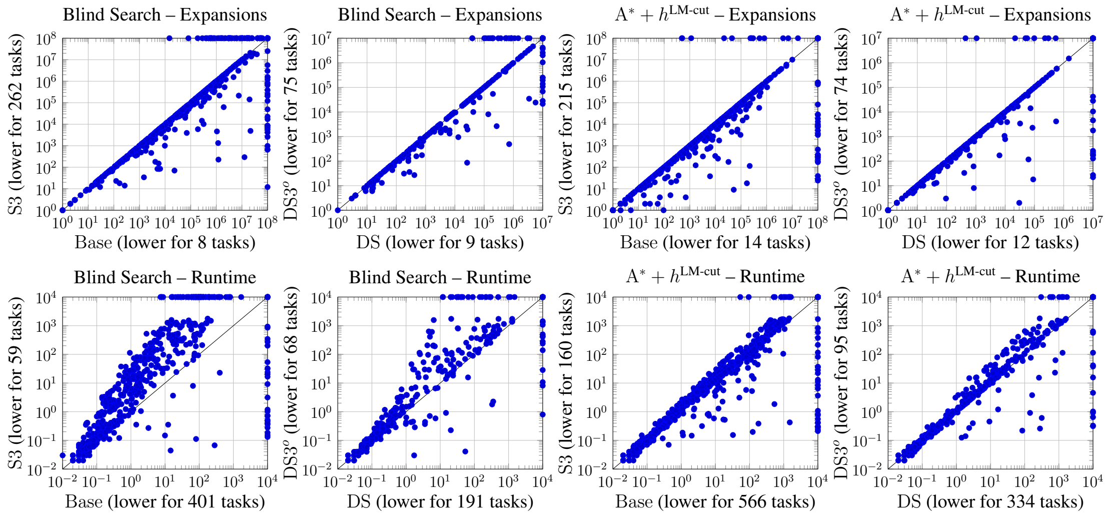


Figure 10.3: Scatter plots comparing the number of expanded states until the last f-layer in A* (top), and runtime (bottom) of (from left to right) Base to $\mathrm{S3}$ and DS to $\mathrm{DS3}^o$ in blind search (the two left-most columns) and in A* with $h^{\text{LM-cut}}$ (the two right-most columns). In all plots we use factorings obtained by IA.

A* with $h^{\text{LM-cut}}$ decoupled strong stubborn sets generally outperform the pruning in explicit-state search. There are still few instances, though, where the search spaces are larger for decoupled search. We conjecture that decoupled stubborn sets lose pruning power in these cases, since due to the more complex state structure more actions need to be included. In comparison to Figure 7.25 in Chapter 7.6, where we compared basic decoupled search to the same strong stubborn sets pruning (POR), we see a clear improvement in that $\mathrm{DS3}^o$ seems to catch up on instances where previously POR had an advantage.

In the right of Figure 10.4, we compare $\mathrm{DS3}$ to the $\mathrm{DS3}^o$ variant with the optimizations from Chapter 10.3. There is no significant difference between the versions, with only few instances where the pruning is stronger or the runtime improves.

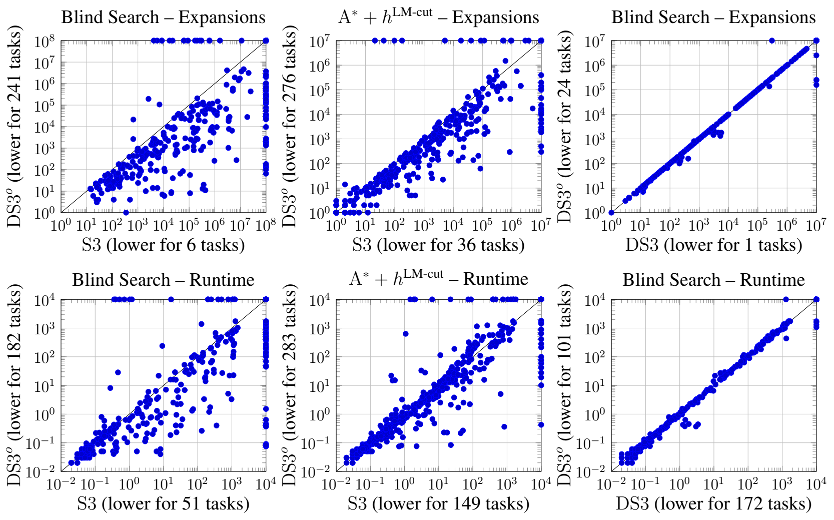


Figure 10.4: Scatter plots comparing the number of expanded states until the last f-layer in A^* (top), and runtime (bottom) of $\mathrm{S3}$ to $\mathrm{DS3}^o$ in blind search (left), in A^* with $h^{\text{LM-cut}}$ (middle), and $\mathrm{DS3}$ to $\mathrm{DS3}^o$ in blind search (right). In all plots we use factorings obtained by IA.

Chapter 11

Symmetry Breaking

Symmetry breaking is well-established and well-explored across several sub-areas of computer science, including AI planning (e. g. Starke, 1991; Emerson and Sistla, 1996; Fox and Long, 1999; Rintanen, 2003; Pochter et al., 2011; Domshlak et al., 2012). It allows to prune (parts of) the exponential search resulting from, e. g., the presence of objects with symmetric behavior. Variants of symmetry breaking have been proposed by Fox and Long [1999] in the context of Graphplan [Blum and Furst, 1997; Long and Fox, 1999], for SAT-based planning [Rintanen, 2003] and, most recently, for state-space search [Pochter et al., 2011; Domshlak et al., 2012, 2013; Wehrle et al., 2015; Sievers et al., 2015; Shleyfman et al., 2015]. In the latter setting, the symmetries take the form of symmetry groups across states, called *orbits*. If several states from an orbit are encountered, only one of these is explored. Upon finding a goal state, a reconstruction procedure takes care of any discontinued paths in the solution. Symmetric states may cause blow-ups in decoupled search as well, so the question is whether the two methods can be integrated. We devise such an integration in this chapter, and demonstrate the theoretical and practical benefits.

We start by introducing the required background and notation for symmetry breaking in classical planning, in particular *structural symmetries* [Shleyfman et al., 2015], which capture previously proposed concepts of symmetry. These can be derived from an input task's syntax in a simple declarative manner. Based on this, we extend the notion of symmetry relations to decoupled states and show how to compute decoupled structural symmetries. We then introduce *decoupled orbit space search*, which exploits symmetries in the decoupled state space, and prove its correctness. Finally, we show that decoupled orbit space search is exponentially separated from its baseline components, and evaluate our algorithm empirically.

This chapter is based on Gnad et al. [2017c]. The author of this work, Álvaro Torralba, and Alexander Shleyfman contributed equally to that publication. All three collaborated on developing the basic algorithm and showing its correctness. The implementation was done by the author of this work, who contributed the application of symmetries to decoupled states and the solution reconstruction; and Álvaro Torralba, who was responsible for the lexicographic orderings and adapting the existing code of symmetry breaking in explicit-state search to provide an interface for decoupled search.

11.1 Background

Symmetry breaking considers equivalence classes of symmetric states in the search space, and allows for using representative states of each equivalence class. Shleyfman et al. [2015] introduced the notion of structural symmetries, which are a relabeling of the factored representation of a given planning task Π. Actions are mapped to actions, variables to variables, and values to values (preserving the variable/value pairs structure). This relabeling induces an automorphism of the state space Θ_Π. We follow the definition of structural symmetries for planning tasks as defined by Wehrle et al. [2015].

Given a directed graph $\mathcal{G} = \langle N, E\rangle$, a permutation σ on the vertices N, s. t. $(n, n') \in E$ iff $(\sigma(n), \sigma(n')) \in E$, is called an *automorphism*. The automorphisms of a graph $\mathcal{G}$ form a group under composition. We call this group an *automorphism group*, and denote it by $\mathrm{Aut}(\mathcal{G})$. For every permutation $\sigma \in \mathrm{Aut}(\mathcal{G})$ there exists an inverse permutation $\sigma^{-1} \in \mathrm{Aut}(\mathcal{G})$ s.t. $\sigma \circ \sigma^{-1}$ is the identity permutation.

Definition 39 (Structural Symmetry)**.** *For a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$, let P be the set of Π's* facts, *i. e., pairs $\langle v, d\rangle$ with $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$. A* structural symmetry *for Π is a permutation $\sigma : P \cup A \to P \cup \mathcal{A}$ such that:*

(1) $\sigma(Q) = Q$, where $Q := \{\{\langle v, d\rangle \mid d \in \mathcal{D}(v)\} \mid v \in \mathcal{V}\}$.

(2) $\sigma(\mathcal{A}) = \mathcal{A}$, and, for all $a \in \mathcal{A}$, $\mathsf{pre}(\sigma(a)) = \sigma(\mathsf{pre}(a))$, $\mathsf{eff}(\sigma(a)) = \sigma(\mathsf{eff}(a))$, and $\mathsf{cost}(\sigma(a)) = \mathsf{cost}(a)$,

(3) $\sigma(\mathcal{G}) = \mathcal{G}$.

where, for a set X, we define $\sigma(X) := \{\sigma(x) \mid x \in X\}$.

Condition (1) ensures that states are mapped to proper states, in particular that a variable is mapped to a variable with same domain size, facts of a variable are not mapped to multiple variables, and that the fact mapping is a bijection. For example, for a (partial) state s, $s' = \sigma(s)$ is the (partial) state obtained from s such that for all $\langle v, d\rangle$ with $v \in \mathsf{vars}(s)$ and $s[v] = d$, we have $\sigma(\langle v, d\rangle) = \langle v', d'\rangle$ and $s'[v'] = d'$. These properties are enforced by condition (2), which allows action permutation only when condition (1) is satisfied for the permuted preconditions and effects, and if the action costs are identical. Condition (3) *stabilizes* the goal, i. e., maps $\mathcal{G}$ to itself.

A set of structural symmetries Σ for a planning task Π induces a subgroup Γ of the automorphism group $\mathrm{Aut}(\Theta_\Pi)$, which in turn defines an equivalence relation over the states S of Π. Namely, we say that a state s is *symmetric* to a state s' iff there exists an automorphism $\sigma \in \Gamma$ such that $\sigma(s) = s'$.

State-space search algorithms with symmetry elimination do not consider all states $s \in S$, but only a single representative element of the equivalence class of s. These equivalence classes are called *orbits* and are usually represented by one of its member states that is called the *canonical* state. The search then explores all applicable actions, and prunes the resulting successor states if another representative of their orbit has already been encountered during search. Due to the properties of structural symmetries, this reduced state transition graph is guaranteed to still contain an optimal plan for every state. However, determining if two states are symmetric is **NP**-hard [Luks, 1993]. To overcome this, one can perform symmetry elimination by computing an approximated canonical representative with an incomplete ad-hoc procedure that is not guaranteed to detect all symmetries [Pochter et al., 2011]. We use the *orbit space search* (OSS) algorithm introduced by Domshlak et al. [2015b]. OSS replaces each state by its approximated canonical representative, resulting in a search over the state transition graph induced by the (approximated) canonical states.

11.2 Symmetry Relations over Decoupled States

To define symmetries in the decoupled state space, we restrict the allowed class of structural symmetries by imposing additional requirements on their properties. Similar to before, this is required to ensure that a decoupled state $s^{\mathcal{F}}$ of a task Π decomposed via $\mathcal{F}$ is again mapped a proper decoupled state that can be represented with the same factoring.

Definition 40 (Decoupled Structural Symmetry)**.** *Let Π be a planning task, and let $\mathcal{F}$ be a factoring for Π. Then σ is a* decoupled structural symmetry *if and only if σ is a structural symmetry and in addition it holds that:*

(i) $\sigma(C) = C$, *and*

(ii) $\forall L \in \mathcal{L} : \sigma(L) \in \mathcal{L}$.

In words, decoupled structural symmetries are the subset of structural symmetries that (i) stabilize the center (center facts are only mapped to center facts), and (ii) stabilize the leaves (when permuting a fact of a leaf L_1 to a fact of L_2, all facts of L_1 must be permuted to facts of L_2). Note that property (i) follows from property (ii) and the fact that σ is a structural symmetry. Nevertheless, we include property (i) into the definition for better readability. These properties are not tautological, i. e., there exist structural symmetries that do not satisfy them:

Proposition 10. *Not every structural symmetry is a decoupled structural symmetry.*

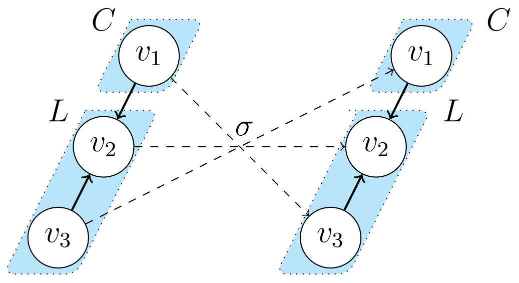


Figure 11.1: A structural symmetry σ that is not a decoupled structural symmetry for the depicted factoring. Solid edges between variables are causal-graph connections.

Proof. Let Π be a planning task, s.t. $\mathcal{V} = \{v_1, v_2, v_3\}$ is a set of binary variables, $\mathcal{A} = \{a_1, a_2\}$ is a set of unit-cost actions, where $\mathsf{pre}(a_1) = \{\langle v_1, 0\rangle\}$, $\mathsf{eff}(a_1) = \{\langle v_2, 1\rangle\}$, $\mathsf{pre}(a_2) = \{\langle v_3, 0\rangle\}$, $\mathsf{eff}(a_2) = \{\langle v_2, 1\rangle\}$, $\mathcal{I} = \{\langle v_1, 0\rangle, \langle v_2, 0\rangle\langle v_3, 0\rangle\}$, and $\mathcal{G} = \{\langle v_2, 1\rangle\}$. There is only one structural symmetry σ, where $\sigma(a_1) = a_2, \sigma(v_1) = v_3$, and $\sigma(v_2) = v_2$. Let $C = \{v_1\}$ and $\mathcal{L} = \{\{v_2, v_3\}\}$ be a factoring with a single leaf L. Figure 11.1 illustrates that σ is not a decoupled structural symmetry, because property (i) of Definition 40 is violated. Namely $\sigma(C) \neq C$. $\square$

The example in Figure 11.1 shows that such symmetries cannot be exploited in decoupled search.[1]

Applying a decoupled permutation to a decoupled state requires permuting its center state and pricing function. We obtain the permuted pricing function by assigning the price of each leaf state s^L to $\sigma(s^L)$:

Definition 41 (Permuted Decoupled States)**.** *Let* Π *be a planning task, and let* $\mathcal{F}$ *be a factoring for* Π*. Let* $s^{\mathcal{F}}$ *be a decoupled state, and let* σ *be a decoupled structural symmetry. We define* $\sigma(s^{\mathcal{F}})$ *as a decoupled state with center* $\mathsf{center}(\sigma(s^{\mathcal{F}})) := \sigma(\mathsf{center}(s^{\mathcal{F}}))$*, and prices* $\mathsf{prices}(\sigma(s^{\mathcal{F}}))$*, where for each leaf state* $s^L \in S^{\mathcal{L}}$: $\mathsf{prices}(\sigma(s^{\mathcal{F}}))[\sigma(s^L)] := \mathsf{prices}(s^{\mathcal{F}})[s^L]$.

Note that $\sigma(s^{\mathcal{F}})$ is a valid decoupled state thanks to the properties of decoupled structural symmetries: Property (i) ensures that $\sigma(\mathsf{center}(s^{\mathcal{F}}))$ is indeed a center state, because center variables are only mapped to center variables. Property (ii) ensures that leaf states are always mapped to leaf states, and the price of a leaf state is mapped to the price of its permuted counterpart. Additionally, (ii) ensures that all states in a leaf L_1 are permuted into states of the same target leaf L_2, so the permuted decoupled state can be represented with $\mathcal{F}$.

[1]We remark that, in practice as far as reflected by the IPC benchmarks and our factoring strategies, this is typically not a serious limitation.

Next, we show that decoupled structural symmetries induce an automorphism group in the decoupled state space. We start by showing that decoupled structural symmetries are a relabeling of the decoupled-state transitions.

Lemma 12. *Let Π be an planning task, $\mathcal{F}$ a factoring for Π, $s^{\mathcal{F}}$ a decoupled state, and let σ be a decoupled structural symmetry. Then, $s^{\mathcal{F}} \xrightarrow{a} t^{\mathcal{F}}$ is a transition in the decoupled state space $\Theta^{\mathcal{F}}_{\Pi}$ if and only if $\sigma(s^{\mathcal{F}}) \xrightarrow{\sigma(a)} \sigma(t^{\mathcal{F}})$ is a transition in $\Theta^{\mathcal{F}}_{\Pi}$.*

Proof. First, observe that by Definition 41 both $\sigma(s^{\mathcal{F}})$ and $\sigma(t^{\mathcal{F}})$ are decoupled states in $\Theta^{\mathcal{F}}_{\Pi}$. Further, because σ stabilizes the center, $\sigma(a)$ has a center effect, so is a center action. It remains to show that $\sigma(s^{\mathcal{F}}) \xrightarrow{\sigma(a)} \sigma(t^{\mathcal{F}})$ is a transition in $\Theta^{\mathcal{F}}_{\Pi}$.

We first show that $\sigma(a)$ is applicable in $\sigma(s^{\mathcal{F}})$. For the center preconditions, because σ is a decoupled structural symmetry, we get that $\mathsf{center}(\sigma(s^{\mathcal{F}})) = \sigma(\mathsf{center}(s^{\mathcal{F}})) \models \sigma(\mathsf{pre}(a))[C] = \mathsf{pre}(\sigma(a))[C]$. For the leaves, there must be a leaf state s^L for every leaf $L \in \mathcal{L}$ such that $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$ and $s^L \models \mathsf{pre}(a)[L]$. Thus, again because of the properties of decoupled structural symmetries, and Definition 41, for every L there exists a leaf state $\sigma(s^L) =: t^{\sigma(L)} \in S^{\sigma(L)}$ such that $\mathsf{prices}(\sigma(s^{\mathcal{F}}))[t^{\sigma(L)}] < \infty$ and $t^{\sigma(L)} \models \mathsf{pre}(\sigma(a))[\sigma(L)]$.

It remains to show that $\sigma(s^{\mathcal{F}})[\![\sigma(a)]\!] = \sigma(t^{\mathcal{F}})$. For the center state, this is obvious, $\mathsf{center}(\sigma(s^{\mathcal{F}}))[\![\sigma(a)]\!] = \mathsf{center}(\sigma(t^{\mathcal{F}}))$ because σ is a decoupled structural symmetry.

For the pricing function, by Definition 41, it holds that $\mathsf{prices}(\sigma(s^{\mathcal{F}}))[\sigma(s^L)] = \mathsf{prices}(s^{\mathcal{F}})[s^L]$ for all $s^L \in S^{\mathcal{L}}$. Because σ is a decoupled structural symmetry, it holds that $s^L \models \mathsf{pre}(a)[L]$ iff $\sigma(s^L) \models \mathsf{pre}(\sigma(a))[\sigma(L)]$. Thus, the set of leaf states $S^{\sigma(L)}_c$ that complies with $\sigma(a)$ is a permutation via σ of the set of leaf states S^L_c that complies with a. What is left to show is that, behind a, respectively $\sigma(a)$, the set of leaf paths that are enabled by $\mathsf{center}(t^{\mathcal{F}})$, respectively $\mathsf{center}(\sigma(s^{\mathcal{F}}))[\![\sigma(a)]\!]$, are permutations of each other and are in one-to-one correspondence. This holds by the same arguments why $\sigma(a)$ is applicable in $\sigma(s^{\mathcal{F}})$ and by induction over the length of the compliant leaf paths.

The other direction follows by the same arguments because σ^{-1} is also a decoupled structural symmetry. □

This result extends to sequences of actions as follows:

Proposition 11. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Let $s^{\mathcal{F}}$ be a decoupled state reachable from $\mathcal{I}^{\mathcal{F}}$, and let σ be a decoupled structural symmetry. Then $\sigma(s^{\mathcal{F}})$ is a decoupled state reachable from $\sigma(\mathcal{I}^{\mathcal{F}})$ s.t. for each leaf state $s^L \in S^{\mathcal{L}}$, $\mathsf{prices}(\sigma(s^{\mathcal{F}}))[\sigma(s^L)] = \mathsf{prices}(s^{\mathcal{F}})[s^L]$.*

Proof. By induction on the path length. The base case, for $\mathcal{I}^{\mathcal{F}}$, directly follows from the definition of permuted decoupled states. The inductive case follows from Lemma 12. □

$$\begin{tikzcd}
s \in [s^{\mathcal{F}}] \arrow[r, dashed, "\sigma"'] \arrow[d, leftrightarrow] & \sigma(s) \in [\sigma(s^{\mathcal{F}})] \arrow[d, leftrightarrow] \\
s^{\mathcal{F}} \arrow[r, dashed, "\sigma"'] & \sigma(s^{\mathcal{F}})
\end{tikzcd}$$

Figure 11.2: Illustration of how the applications of a permutation to a decoupled state corresponds to applying it on its hypercube.

We now prove one of our main results, namely that decoupled structural symmetries induce an automorphism group over the decoupled state space:

Theorem 27. *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π*. If* σ *is a decoupled structural symmetry of* Π *and* $\mathcal{F}$*, then* σ *is an automorphism of* $\Theta^{\mathcal{F}}_{\Pi}$.

Proof. Let Σ be the set of all decoupled structural symmetries over Π using the factoring $\mathcal{F}$. First, we show that Σ is a group. By Lemma 1 in Shleyfman et al. [2015] structural symmetries form a finite group. By definition, Σ is a subset of this group, so it is enough to show that Σ is closed under composition, i. e., if $\sigma_1, \sigma_2 \in \Sigma$ then $\sigma_1 \circ \sigma_2 \in \Sigma$. Let C be the center variables of the factoring $\mathcal{F}$, then $\sigma_1 \circ \sigma_2(C) = \sigma_1(\sigma_2(C)) = \sigma_1(C) = C$. By the same argument, the leaves of the factoring are preserved under the composition. Second, to prove that a decoupled structural symmetry σ induces a graph symmetry of the decoupled state space $\Theta^{\mathcal{F}}_{\Pi}$, we need to show that:

(i) $\sigma(\mathcal{S}^{\mathcal{F}}) = \mathcal{S}^{\mathcal{F}}$,

(ii) $\sigma(\mathcal{A}^{C}) = \mathcal{A}^{C}$,

(iii) $s^{\mathcal{F}} \xrightarrow{a} t^{\mathcal{F}}$ with $a \in \mathcal{A}^{C}$ iff $\sigma(s^{\mathcal{F}}) \xrightarrow{\sigma(a)} \sigma(t^{\mathcal{F}})$ with $\sigma(a) \in \mathcal{A}^{C}$

(iv) $\sigma(\mathcal{S}^{\mathcal{F}}_{\mathcal{G}}) = \mathcal{S}^{\mathcal{F}}_{\mathcal{G}}$.

Property (i) holds by Definitions 40 and 41, since σ is a relabeling of the variable/value pairs that preserves the factoring. For property (ii), suppose that $a \in \mathcal{A}^{C}$, then a affects at least one variable in C. Thus, as $\sigma(C) = C$, $\sigma(a)$ also affects at least one variable in C and $\sigma(a) \in \mathcal{A}^{C}$. Property (iii) holds by Lemma 12, and property (ii).

Regarding property (iv), suppose that $s^{\mathcal{F}}_{G} \in \mathcal{S}^{\mathcal{F}}_{\mathcal{G}}$, then as $\mathsf{center}(s^{\mathcal{F}}_{G})$ is a goal center state, this is also true for $\mathsf{center}(\sigma(s^{\mathcal{F}}_{G}))$ since σ stabilizes the center and the goal. For every $L \in \mathcal{L}$, there exists a goal leaf state $s^{L} \in S^{L}$ s. t. $\mathsf{prices}(s^{\mathcal{F}}_{G})[s^{L}] = c^{L} < \infty$. Then, $\sigma(L) \in \mathcal{L}$ and there exists a goal leaf state $\sigma(s^{L}) \in S^{\sigma(L)}$ s. t. $\mathsf{prices}(\sigma(s^{\mathcal{F}}_{G}))[\sigma(s^{L})] = c^{L}$, again because σ is a decoupled structural symmetry. □

Applying a permutation to a decoupled state is equivalent to applying that permutation to each of the member states of its hypercube, as illustrated in Figure 11.2:

Theorem 28. *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π. *Let* $s^{\mathcal{F}}$ *be a decoupled state, and let* σ *be a decoupled structural symmetry. Then* $\sigma(s) \in [\sigma(s^{\mathcal{F}})]$ *if and only if* $s \in [s^{\mathcal{F}}]$, *and* $\mathsf{price}(\sigma(s^{\mathcal{F}}), \sigma(s)) = \mathsf{price}(s^{\mathcal{F}}, s)$.

Proof. Let s be a state in $[s^{\mathcal{F}}]$ with price $p = \mathsf{price}(s^{\mathcal{F}}, s)$. Then $s = \mathsf{center}(s^{\mathcal{F}}) \cup \bigcup_{L \in \mathcal{L}} s[L]$. If we apply the permutation σ to s, we obtain $\sigma(s) = \sigma(\mathsf{center}(s^{\mathcal{F}})) \cup \bigcup_{L \in \mathcal{L}} \sigma(s[L])$. By Definition 41, we have $\mathsf{center}(\sigma(s^{\mathcal{F}})) = \sigma(\mathsf{center}(s^{\mathcal{F}}))$, and for every leaf state $s^L \in S^{\mathcal{L}} : \mathsf{prices}(\sigma(s^{\mathcal{F}}))[\sigma(s^L)] = \mathsf{prices}(s^{\mathcal{F}})[s^L]$. Hence, we get $\sum_{L \in \mathcal{L}} \mathsf{prices}(\sigma(s^{\mathcal{F}}))[\sigma(s[L])] = \sum_{L \in \mathcal{L}} \mathsf{prices}(s^{\mathcal{F}})[s[L]] = p$, and therefore $\sigma(s) \in [\sigma(s^{\mathcal{F}})]$ with price p.

The other direction follows by the same arguments because σ^{-1} is also a decoupled structural symmetry. □

11.3 Finding Decoupled-State Symmetries

It is obviously infeasible to compute the automorphism group of the state space Θ_Π of a planning task Π directly on the state space. The state space symmetries need to be inferred from the compact representation. Prior work introduced the *problem description graph* (PDG) of a task Π, and showed that the automorphism group of this graph induces a subgroup of $\mathrm{Aut}(\Theta_\Pi)$ [Pochter et al., 2011]. Later, Domshlak et al. [2012] made some modifications to the definition of the PDG, mainly to allow support of general-cost actions. Considering that the number of PDG vertices is linear in the size of Π, its automorphism group can be found efficiently using off-the-shelf tools. Our definition loosely follows that by Domshlak et al. [2012]:

Definition 42 (Problem Description Graph)**.** *Let* Π *be a planning task. The* problem description graph *of* Π, *PDG*(Π), *is the colored digraph* $\langle N, E\rangle$ *with nodes* N, *node colors* $col(n)$ *for* $n \in N$, *and edges* E:

$$N = \{n_v \mid v \in \mathcal{V}\} \cup \bigcup_{v \in \mathcal{V}} \{n_{\langle v,d\rangle} \mid d \in \mathcal{D}(v)\} \cup \{n_a \mid a \in \mathcal{A}\},$$

$$col(n) = \begin{cases} 1 & \text{if } n = n_{\langle v,d\rangle} \text{ and } \langle v,d\rangle \in \mathcal{G} \\ 2 + \mathsf{cost}(a) & \text{if } n = n_a \text{ and } a \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases}$$

$$E = \bigcup_{v \in \mathcal{V}} \{\langle n_v, n_{\langle v,d\rangle}\rangle \mid d \in \mathcal{D}(v)\} \cup \bigcup_{a \in \mathcal{A}} \left(E_a^{\mathsf{pre}()} \cup E_a^{\mathsf{eff}()}\right)$$

Where $E_a^{\mathsf{pre}()}$ and $E_a^{\mathsf{eff}()}$ are defined as follows:

$$E_a^{\mathsf{pre}()} = \{\langle n_{\langle v,d\rangle}, n_a\rangle \mid \langle v,d\rangle \in \mathsf{pre}(a)\},$$
$$E_a^{\mathsf{eff}()} = \{\langle n_a, n_{\langle v,d\rangle}\rangle \mid \langle v,d\rangle \in \mathsf{eff}(a)\}.$$

The automorphism group of PDG(Π) induces a set of structural symmetries of Π. By the same recipe we create a slightly modified version of the PDG, which induces the decoupled structural symmetries of Π:

Definition 43 (Factored Problem Description Graph)**.** *Let Π be a planning task, $\mathcal{F}$ a factoring for Π, and $\langle N, E\rangle$ the PDG of Π. The* factored problem description graph *of Π given $\mathcal{F}$, PDG$(\Pi, \mathcal{F})$, is the colored digraph $\langle N', E'\rangle$ with: $N' = N \cup \{n_L \mid L \in \mathcal{L}\}$, node colors extending those of PDG by col$(n_L) = -1$ for all $L \in \mathcal{L}$, and edges $E' = E \cup \bigcup_{L \in \mathcal{L}}\{\langle n_L, n_v\rangle \mid v \in L\}$.*

We ensure the properties required to only obtain decoupled structural symmetries by adding the n_L nodes. By attaching each n_L node to the variables in L, leaf variables can only be mapped to leaf variables (so center variables can only be mapped to center variables). By coloring all n_L nodes in the *same* color, we allow permutations across leaf factors. Thus, because all variables of a particular leaf L are connected to exactly one n_L, we guarantee that *if* there exist symmetries that permute a variable of L_1 to one of L_2, then all variables of L_1 must be mapped into variables of L_2. The structural symmetries obtained from a factored PDG are obviously a subset of those that can be obtained from the PDG of the same planning task.

Proposition 12. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Every automorphism of PDG$(\Pi, \mathcal{F})$ corresponds to a decoupled structural symmetry of Π and $\mathcal{F}$.*

Proof. PDG symmetries are structural symmetries of Π. The two additional properties of decoupled structural symmetries are ensured by the new nodes n_L for each leaf L. These nodes ensure that center variables $v \in C$ can only be mapped to center variables. Further more, they ensure that if cross-leaf permutations exist, then the entire leaves have to be permuted. Alternatively, leaf states $s^L \in S^L$ are permuted within the same leaf L. $\square$

11.4 Symmetry Breaking in Decoupled Search

Exploiting the symmetries of a planning task by pruning states has proved to be highly beneficial in explicit-state search. We propose *decoupled orbit space search (*DOSS*)*, that applies symmetry pruning in the decoupled state space. Like orbit space search, DOSS replaces each new decoupled state by its *canonical* representative. We next provide further details about how exactly a canonical decoupled state is computed and, once the search is completed, how a valid (optimal) plan can be reconstructed from the decoupled orbit state space.

11.4.1 Mapping to Canonical Representatives

In orbit space search, every state is replaced by its canonical representative. The *perfect* canonical state is defined as the one with minimal lexicographic ordering, given an arbitrary total order on variable/value pairs. Because finding the coarsest relation is **NP**-hard [Luks, 1993], OSS approximates canonical representatives through a local search procedure, where search nodes correspond to states and search transitions correspond to the application of one of the generators of the structural symmetry group. Search states are ranked based on the lexicographical order. The search stops at a local minimum.

We adapt this idea to handle decoupled states. We define the perfect canonical decoupled state to be one with a lexicographically minimal center state given an arbitrary total order on the values of center variables, and among these center-minimal decoupled states, one whose pricing function is lexicographically minimal according to an arbitrary total order on the set $S^{\mathcal{L}}$ of leaf states.[2]

Like in OSS, we approximate canonical representatives via a local search procedure. For efficiency reasons, we divide this local search into two phases. A permutation σ is called *center-affecting* for a decoupled state $s^{\mathcal{F}}$, if $\sigma(\mathsf{center}(s^{\mathcal{F}})) \neq \mathsf{center}(s^{\mathcal{F}})$. Otherwise, we say that σ is *center-stable* for $s^{\mathcal{F}}$. We first perform a local search only considering center-affecting permutations, in order to obtain a decoupled state that is a local minimum with respect to the center state. Note that these center-affecting permutations may also affect the leaves, but are only applied if they "improve" the current center state. We then perform a second local search from that state, using only center-stable permutations. Dividing the search into two phases reduces the computational overhead to obtain the canonical decoupled state, since the more expensive checks of whether a permutation produces a lexicographically smaller pricing function are only performed for center-stable permutations.

11.4.2 Solution Reconstruction

When the search stops once a decoupled goal state $s^{\mathcal{F}}_G$ is found, the sequence of center actions leading to $s^{\mathcal{F}}_G$ is not guaranteed to be valid, because a permutation can have been applied in every step. Let $\sigma_0(s^{\mathcal{F}}_0) \xrightarrow{a_1} \sigma_1(s^{\mathcal{F}}_1[\![a_1]\!]) \dots \xrightarrow{a_k} \sigma_k(s^{\mathcal{F}}_k[\![a_k]\!])$ be a sequence of decoupled transitions, where $s^{\mathcal{F}}_i = \sigma_{i-1}(s^{\mathcal{F}}_{i-1})$, $s^{\mathcal{F}}_0 = \mathcal{I}^{\mathcal{F}}$, and $\sigma_k(s^{\mathcal{F}}_k[\![a_k]\!]) \models \mathcal{G}$. To obtain a plan for the given planning task, we need to compute a valid center path, reconstruct the compliant leaf paths, and embed them into the center path.

We obtain a valid center path similarly to standard plan reconstruction in OSS [Domshlak et al., 2012]. This procedure retrieves all permutations $(\sigma_0, \dots, \sigma_k)$ applied during the search and then reconstructs a plan from the (real) initial decoupled state to

[2]Preferring center-minimality reduces the number of different center states among canonical decoupled states, which tends to be beneficial given the focus of decoupled search on the center.

a goal state by finding the applicable actions a'_i that produce the unpermuted state in every step. More formally, the reconstruction retrieves the following path:

$$s_0^{\mathcal{F}} \xrightarrow{a'_1} \sigma_0^{-1}(s_1^{\mathcal{F}}[\![a_1]\!]) \dots \xrightarrow{a'_k} (\sigma_0^{-1} \circ \sigma_1^{-1} \circ \dots \circ \sigma_{k-1}^{-1})(s_k^{\mathcal{F}}[\![a_k]\!])$$

With a valid center path $\pi^C = \langle a'_1, \dots, a'_k \rangle$, we apply the standard solution construction process of decoupled search. This is guaranteed to return the optimal global plan that extends π^C.

11.4.3 Completeness and Optimality

Herein, we prove that DOSS preserves the completeness and optimality of the underlying search algorithm.

Lemma 13. *Let Π be a planning task, $\mathcal{F}$ a factoring for Π, and σ a decoupled structural symmetry. Let $s^{\mathcal{F}}$ be a decoupled state. Then $h^*_{\mathcal{F}}(\sigma(s^{\mathcal{F}})) = h^*_{\mathcal{F}}(s^{\mathcal{F}})$.*

Proof. By Theorem 28, for every $s \in [s^{\mathcal{F}}]$ with price p we have $\sigma(s) \in [\sigma(s^{\mathcal{F}})]$ with the same price. By the properties of structural symmetries, $h^*(s) = h^*(\sigma(s))$ for all $s \in [s^{\mathcal{F}}]$. Therefore, $h^*_{\mathcal{F}}(s^{\mathcal{F}}) = \min_{s \in [s^{\mathcal{F}}]} \mathsf{price}(s^{\mathcal{F}}, s) + h^*(s) = \min_{s \in [\sigma(s^{\mathcal{F}})]} \mathsf{price}(\sigma(s^{\mathcal{F}}), s) + h^*(s) = h^*_{\mathcal{F}}(\sigma(s^{\mathcal{F}}))$. □

Completeness and optimality now follow by an argument similar to that for OSS [Domshlak et al., 2015b]:

Theorem 29. DOSS *preserves completeness and optimality of the search algorithm employed.*

Proof. Let Π be a planning task, and let $\mathcal{F}$ be a factoring for Π. DOSS performs decoupled search by replacing each decoupled state $s^{\mathcal{F}}$ with its canonical representative $\sigma(s^{\mathcal{F}})$, where σ is a decoupled structural symmetry. By Lemma 13, the goal distance of $\sigma(s^{\mathcal{F}})$ and $s^{\mathcal{F}}$ is the same (recall that $h^*_{\mathcal{F}}(s^{\mathcal{F}})$ takes into account both the center and leaf cost). Thus, an optimal plan for $s^{\mathcal{F}}$ has the same cost as an optimal plan for $\sigma(s^{\mathcal{F}})$ and we can safely replace $s^{\mathcal{F}}$ by $\sigma(s^{\mathcal{F}})$. Since structural symmetries preserve action costs, we furthermore get that for each step taking a transition in the decoupled state space $\mathsf{cost}(a_i) = \mathsf{cost}(\sigma(a_i))$. The claim follows as decoupled search preserves completeness and optimality. □

11.5 Exponential Separation from Base Methods

We analyze the theoretical differences between DOSS and its two base components, orbit-space search (OSS) and decoupled search (DS). This extends the results from

Chapter 6. Herein, we show that DOSS can be exponentially more efficient than both of its components *on the same family* of planning tasks, so the combination is actually more than the sum of its components.

Theorem 30. *There exist families of planning tasks $\{\Pi^n\}$ with factorings $\mathcal{F}^n$, structural symmetries Γ^n, and decoupled structural symmetries $\Gamma^n_d \subseteq \Gamma^n$ such that the reachable decoupled search space under* DOSS *has size polynomial in n, while both the reachable state space under* OSS *and the reachable decoupled state space $\Theta_\Pi^{\mathcal{RF}}$ have size exponential in n.*

Proof. Let Π^n be a planning task s.t. $\mathcal{V} = \{v^C_0, \dots, v^C_n, v^L_1, \dots, v^L_n\}$. $\mathcal{F}^n$ is a factoring with n leaves $\mathcal{L} = \{L_1, \dots, L_n\}$. The $n+1$ center variables $v^C_0, \dots, v^C_n$ are binary. Each leaf variable v^L_i has domain $\mathcal{D}(v^L_i) = \{0, \dots, i\}$, and is assigned to a leaf factor $L_i = \{v^L_i\}$. The initial state $\mathcal{I}$ assigns 0 to all variables, the goal is $\mathcal{G} = \{\langle v^C_i, 1\rangle \mid 0 \leq i \leq n\} \cup \{\langle v^L_j, j\rangle \mid 0 \leq j \leq n\}$. There are three types of (unit-cost) actions:

1. a^C_0, where $\mathsf{pre}(a^C_0) = \{\langle v^C_0, 1\rangle\}$, and $\mathsf{eff}(a^C_0) = \{\langle v^C_0, 0\rangle\}$,
2. $a^C_{i,1}$, where $\mathsf{pre}(a^C_{i,1}) = \{\langle v^C_0, 0\rangle, \langle v^C_i, 0\rangle\}$, and $\mathsf{eff}(a^C_{i,1}) = \{\langle v^C_0, 1\rangle, \langle v^C_i, 1\rangle\}$, for $i \in \{1, \dots, n\}$,
3. $a^L_{i,j}$, where $\mathsf{pre}(a^L_{i,j}) = \{\langle v^L_i, j\rangle\} \cup \{\langle v^C_k, 0\rangle \mid 1 \leq k \leq n\}$, and $\mathsf{eff}(a^L_{i,j}) = \{\langle v^L_i, j+1\rangle\}$, for $i \in \{1, \dots, n\}, j \in \{0, \dots, i-1\}$.

The structural symmetries Γ^n in this domain are such that all center variables except v^C_0 are symmetric to one another, so v^C_i is symmetric to v^C_j for any $i, j \in \{1, \dots, n\}$. There are no structural symmetries among the leaves because they have a different number of facts. Since there are $n!$ combinations of leaf states that are reachable from the initial state, the number of states considered by OSS is exponential in n.

Since the center precondition of all leaf actions $a^L_{i,j}$ holds in the initial center state, all decoupled states have the same pricing function in all leaves. The minimum possible prices are obtained in the initial decoupled state and do not change later on. Decoupled search still has to explore all different center states, of which there are 2^{n+1}.

Finally, consider the search space of DOSS. The structural symmetries Γ^n only affect center variables in $\mathcal{F}^n$. Hence, $\Gamma^n_d = \Gamma^n$ are decoupled structural symmetries under factoring $\mathcal{F}^n$. As before, all decoupled states have the same pricing function, so the state space explored by DOSS is isomorphic to the one explored by OSS projected on the center variables, which has $2n$ different states. Therefore, the search space of DOSS has polynomial size in n. □

Note that the exponential separation does not imply that the search space under DOSS will always be smaller than that of its components. While this is true for OSS compared to standard search, decoupled search does not dominate Base, as we have seen in Chapter 3.4.2. This of course also applies to the comparison to OSS, which is hence exponentially separated from DOSS. However, as we shall see in the next section, in practice the search space under DOSS tends to be smaller than that of decoupled search and OSS.

11.6 Experimental Evaluation

We implemented DOSS in our decoupled search planner based on Fast Downward [Helmert, 2006b]. The general settings, benchmarks, algorithms, and heuristics used in the evaluation are as described in Chapter 7.2. We evaluate DOSS in satisficing planning, optimal planning, and proving unsolvability. To obtain the decoupled structural symmetries of a planning task Π and factoring $\mathcal{F}$, we use the BLISS tool [Junttila and Kaski, 2007] on the factored problem description graph. We use the IA factoring strategy throughout, since it is a simple strategy that returns good factorings in many instances. The source code and evaluation data are publicly available [Gnad, 2021a].

We start by showing the results in satisficing planning, where we run explicit-state search without pruning (Base), orbit-space search (OSS), and decoupled orbit-space search (DOSS), with greedy best-first search and the h^{FF} heuristic. We include coverage data (number of solved instances) in Figure 11.3 for search with and without the use of preferred-operator pruning (PO). For the latter, we also compare to the LAMA and BFWS planners. In this setting, we only apply the center-affecting symmetries, because the use of all symmetries to compute the canonical representative incurs too much overhead.

Without preferred-operator pruning (left part of the table), DOSS outperforms all other methods overall. Looking at individual domains, there is no clear picture. DOSS often inherits the strength of one of its base methods, but often performs worse than both, too. There are, however, domains where DOSS in fact solves more instances than both baselines, most remarkably in Childsnack, and in Depots. With PO, the results are similar, in general, although there is no more domain where DOSS beats both baselines. In contrast, there are more domains where the combination performs worse, most visibly in TPP. As a result, total coverage is even lower than for Base. We conjecture that even only applying the center-affecting symmetries incurs too much overhead.

In the scatter plots in Figure 11.4, we compare the search space size (number of state evaluations) and runtime. On the left, we show the differences between OSS and Base as a basis for the comparison of DOSS to DS (middle). In general, the results are similar for decoupled search, although in contrast to explicit-state search all differences are less pronounced. In terms of state evaluations, the use of symmetries in decoupled search seems to be consistently better, compared to explicit-state search. For runtime, however, we see that the overhead is apparently higher in decoupled search. For the comparison between OSS and DOSS, the decoupled search variant is clearly ahead in

			GBFS with h^{FF}				+ PO Pruning					
Domain	#	$\#\mathcal{F}$	Base	OSS	DS	DOSS	Base	OSS	DS	DOSS	LAMA	BFWS
Agricola	20	20	**10**	8	**10**	8	**12**	10	**12**	8	**12**	10
Barman	40	2	0	**1**	0	0	0	**2**	0	0	**2**	**2**
Childsnack	20	20	0	7	2	**20**	6	17	**20**	**20**	6	8
DataNetwork	20	20	**7**	**7**	6	6	10	10	7	7	**13**	9
Depots	22	22	15	18	19	**21**	19	**22**	21	**22**	20	**22**
Driverlog	20	20	18	**19**	**19**	**19**	**20**	**20**	**20**	**20**	**20**	**20**
Floortile	40	40	**8**	**8**	5	6	8	**9**	6	6	8	5
Freecell	80	42	**42**	**42**	41	36	**42**	**42**	**42**	37	**42**	**42**
Grid	5	5	**4**	**4**	3	**4**	4	4	**5**	**5**	**5**	**5**
Logistics	63	63	51	57	**63**	**63**	61	60	**63**	**63**	**63**	62
Maintenance	20	9	**1**	**1**	**1**	0	2	2	2	2	3	**8**
Mprime	35	6	**6**	**6**	**6**	4	**6**	**6**	**6**	4	**6**	**6**
NoMystery	20	20	8	9	**19**	**19**	9	9	**19**	**19**	11	17
Organic-Split	20	19	**12**	**12**	11	10	11	**12**	11	9	**12**	**12**
ParcPrinter	30	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	19
Pathways	30	30	11	11	**14**	**14**	21	21	21	21	23	**30**
Rovers	40	40	22	**23**	21	21	**40**	**40**	**40**	38	**40**	**40**
Satellite	36	36	25	**32**	26	28	**36**	**36**	33	31	**36**	31
Tetris	20	17	4	**5**	4	**5**	9	10	11	10	**14**	13
Thoughtful	20	13	**5**	**5**	**5**	**5**	7	7	6	5	11	**12**
Tidybot	20	20	**16**	**16**	15	15	16	16	15	15	17	**18**
TPP	30	29	22	24	**26**	19	**29**	**29**	25	16	**29**	28
Trucks	30	27	13	**15**	13	13	16	**17**	15	15	16	14
Woodworking	40	37	31	31	**33**	**33**	**37**	**37**	**37**	**37**	**37**	21
Other	965	322	322	322	322	322	322	322	322	322	322	322
Total	1686	899	673	703	704	**711**	763	780	779	752	**788**	776

Figure 11.3: Coverage data, i. e., the number of solved tasks, in satisficing planning, on instances where IA does not abstain. $\#\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in “Other”. We highlight the best coverage (separately for search with vs. without preferred-operator pruning) in **bold face**.

terms of state evaluations. For runtime, there is no advantage for either method, similar advantages exist in both directions.

Next, we look into optimal planning. Here, we distinguish two DOSS configurations: DOSS uses all symmetries, while DOSS$_C$ only uses the center-affecting symmetries to compute the canonical. We show results for blind search and A* with $h^{\text{LM-cut}}$. Coverage is shown in Figure 11.5. DOSS clearly outperforms all baseline planners and even beats the state-of-the art planners SBD and C2 when used with $h^{\text{LM-cut}}$. It often

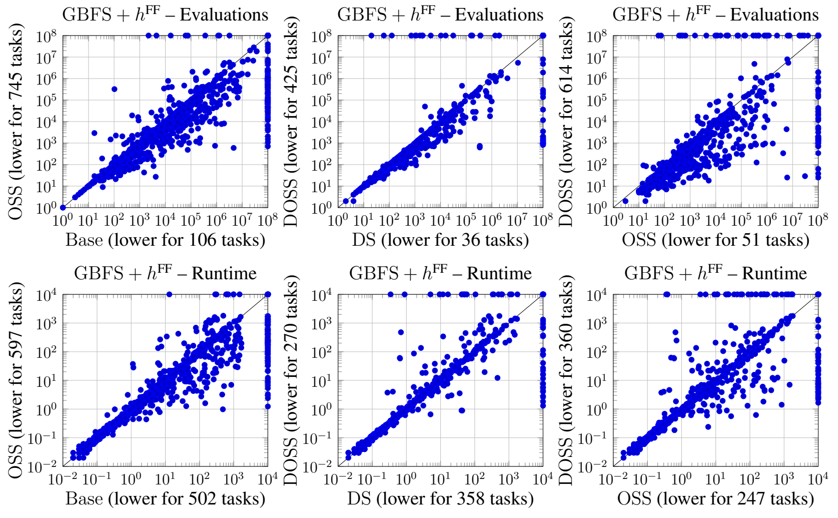


Figure 11.4: Scatter plots comparing the number of evaluated states (top), and runtime (bottom) of OSS to Base (left), and DOSS to decoupled search (middle), and OSS (right). In all plots we use factorings obtained by IA and run GBFS with h^{FF}.

inherits the best coverage of OSS or DS, though sometimes performs worse. With blind search, there are again two domains (Elevators and Logistics) where DOSS improves over both base methods. When using A^* with $h^{\text{LM-cut}}$, there are even *six* domains where this is the case, most notably in Childsnack. Comparing DOSS to DOSS_C, in most domains there is no big difference in coverage. The exception is Openstacks, where the advantage of four/five instances leads to an overall win of DOSS.

In the four leftmost plots in Figure 11.6 we see that, where OSS almost always improves search space size (top) and runtime (bottom) over Base, and does often do so by a large margin, this effect is significantly less visible when enabling symmetry pruning in decoupled search. Still, the additional pruning almost always pays off. Comparing OSS to DOSS (third column), we see that DOSS consistently explores a smaller search space. There are also often big advantages in terms of runtime, although there are also some exceptions. Comparing the use of different symmetries to compute canonical decoupled states, we see only a minor difference. There is a minor positive trend for DOSS in terms of search space size. This typically also translates to a runtime advantage.

We conclude our evaluation by showing results for proving unsolvability. We compare the baselines to DOSS, using blind search and A^* with $h^{\max}$ for deadend detection. With $h^{\max}$, we also include Sympa in the comparison. Looking at coverage (number of instances proved unsolvable) in Figure 11.7, the picture is similar to optimal planning. Good performance of the base methods is often inherited by DOSS, though not always. Again, there are domains where DOSS improves over both DS and OSS, namely DocTransfer and Rovers. Although DOSS can outperform Sympa in some domains with

			Blind Search					A* with $h^{\text{LM-cut}}$						
Domain	#	#$\mathcal{F}$	Base	OSS	DS	DOSS$_C$	DOSS	Base	OSS	DS	DOSS$_C$	DOSS	SBD	C2
Agricola	20	20	0	**7**	0	4	4	0	0	0	0	0	**5**	0
Childsnack	20	20	0	**6**	0	**6**	**6**	0	6	0	**8**	**8**	4	0
DataNet	20	20	7	7	**9**	**9**	**9**	12	12	**14**	**14**	**14**	13	13
Depots	22	22	4	**6**	4	5	**6**	7	8	7	**9**	**9**	5	7
Driverlog	20	20	7	7	**11**	**11**	**11**	13	13	13	13	13	12	**15**
Elevators	30	30	13	15	16	**18**	**18**	22	22	23	23	23	**25**	**25**
Floortile	40	40	**2**	**2**	**2**	**2**	**2**	13	16	9	11	11	**34**	28
Freecell	80	42	**3**	**3**	0	0	0	2	2	2	2	2	2	**3**
GED	20	20	**15**	**15**	**15**	**15**	**15**	15	15	15	15	15	19	**20**
Grid	5	5	**1**	**1**	**1**	**1**	**1**	2	2	2	2	2	2	**3**
Logistics	63	63	12	14	25	**28**	27	26	26	36	36	**37**	24	28
Miconic	150	145	45	**51**	46	47	47	136	**137**	135	135	135	107	98
Mprime	35	6	**6**	**6**	4	4	4	**6**	**6**	4	4	4	**6**	**6**
NoMystery	20	20	8	8	**20**	19	19	14	15	**20**	**20**	**20**	14	**20**
Openstacks	80	50	25	**30**	20	21	25	24	29	20	20	25	**50**	40
Org-Split	20	16	6	**7**	3	3	4	11	**15**	9	11	11	5	5
ParcPrinter	20	13	1	1	**3**	**3**	**3**	4	4	**7**	**7**	**7**	2	4
PSR	50	48	47	**48**	**48**	**48**	**48**	47	**48**	**48**	**48**	**48**	**48**	**48**
Rovers	40	40	5	5	**7**	**7**	**7**	7	7	8	9	9	**14**	13
Satellite	36	36	5	**6**	5	5	5	7	**13**	9	**13**	**13**	8	9
Scanalyzer	30	9	**6**	**6**	3	3	3	5	5	5	5	5	**6**	**6**
Tetris	17	13	4	**7**	5	**7**	**7**	5	7	4	5	5	2	**10**
Tidybot	30	30	14	14	**16**	**16**	**16**	18	**19**	17	18	17	7	18
TPP	30	29	5	**6**	5	5	5	5	7	5	6	6	7	**14**
Transport	59	30	**15**	**15**	**15**	**15**	**15**	14	14	14	15	15	**16**	**16**
Trucks	30	27	5	**7**	4	4	5	9	11	10	**12**	**12**	10	**12**
Woodwork	30	26	6	**7**	**7**	**7**	**7**	14	16	17	17	17	**19**	17
Zenotravel	20	20	8	8	**9**	8	8	**13**	**13**	**13**	11	11	10	**13**
Other	593	36	10	10	10	10	10	11	11	11	11	11	11	11
Total	1630	896	285	325	313	331	**337**	462	499	477	500	**505**	487	502

Figure 11.5: Same setup as in Figure 11.3 for optimal planning. We highlight the best coverage (separately for blind search and A* with $h^{\text{LM-cut}}$) in **bold face**.

$h^{\max}$, it is not competitive overall; Sympa is too strong in Rovers and TPP.

In the scatter plots in Figure 11.8, observe that symmetry pruning almost always pays off in proving unsolvability for both explicit-state and decoupled search. DOSS also cleary outperforms OSS in terms of search space size and runtime.

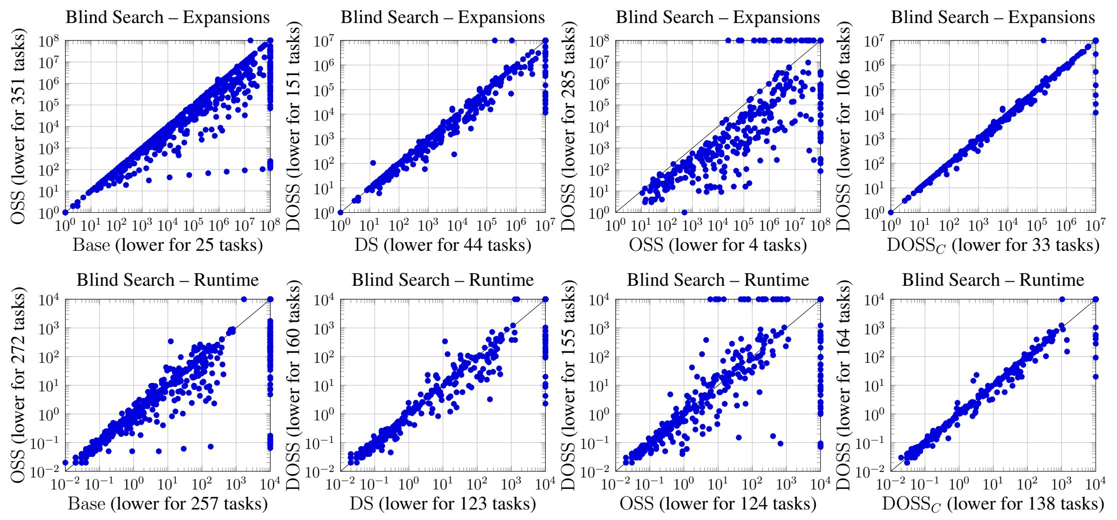


Figure 11.6: Scatter plots comparing the number of expanded states until the last f-layer in A^* (top), and runtime (bottom) of OSS to Base (left), and DOSS to decoupled search (second column) and OSS (third column). In the rightmost column, we compare the use of difference lexicographic orderings. In all plots we use factorings obtained by IA and run blind search.

			Blind Search				A^* with $h^{\max}$				
Domain	#	$\#\mathcal{F}$	Base	OSS	DS	DOSS	Base	OSS	DS	DOSS	Sympa
BagBarman	20	16	**12**	**12**	**12**	**12**	8	8	8	8	**11**
Cavediving	25	21	3	**7**	4	4	3	**7**	4	4	3
DocTransfer	20	20	5	5	5	**8**	7	12	13	**14**	10
NoMystery	23	23	2	2	**12**	**12**	2	3	**12**	**12**	**12**
Rovers	19	19	6	6	**9**	**9**	6	6	9	10	**16**
Tetris	20	20	5	**10**	5	**10**	5	**10**	5	5	5
TPP	30	30	**17**	**17**	12	12	16	16	15	15	**24**
Other	161	46	26	26	26	26	26	26	26	26	26
$\sum$	318	195	76	85	85	**93**	73	88	92	94	**107**
Unsolvable Benchmarks from Hoffmann et al. [2014]											
3-SAT	30	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
NoMystery	25	25	0	0	**24**	**24**	0	0	**25**	**25**	24
Rovers	25	25	0	0	0	**1**	1	2	2	2	**18**
TPP	25	25	**6**	**6**	0	0	6	6	1	1	**18**
Other	11	0	0	0	0	0	0	0	0	0	0
$\sum$	116	76	7	7	25	**26**	8	9	29	29	**60**
Total	434	271	83	92	110	**119**	81	97	121	123	**167**

Figure 11.7: Same setup as in Figure 11.3 for proving unsolvability. We highlight the best coverage (separately for blind search and A^* with $h^{\max}$) in **bold face**.

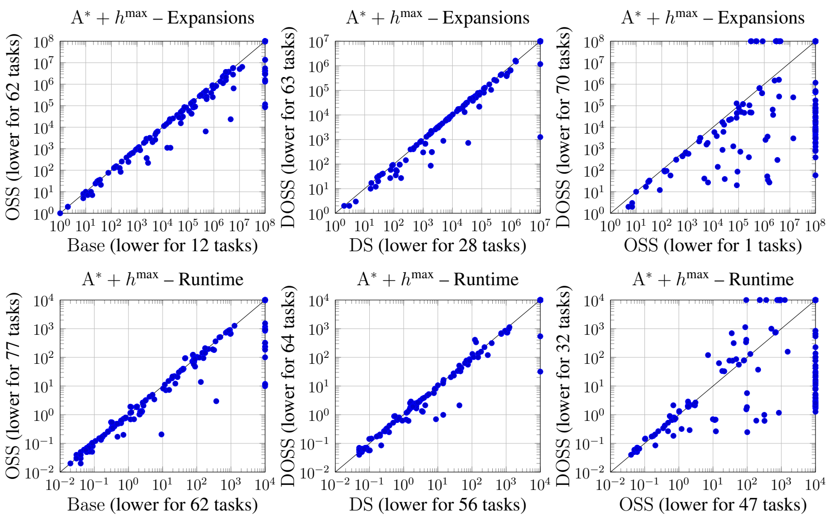


Figure 11.8: Scatter plots comparing the number of expanded states (top), and runtime (bottom) of OSS to Base (left), and DOSS to decoupled search (middle) and OSS (right). In all plots we use factorings obtained by IA and run A^* with h^{max}.

Chapter 12

Symbolic Leaf Representation

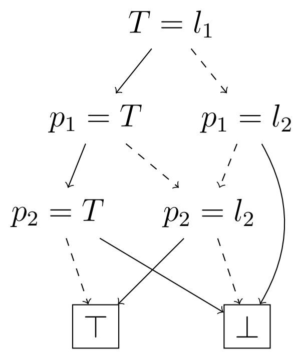
In this chapter we address a weakness of decoupled search: large leaf components whose state space needs to be enumerated explicitly. We develop a symbolic representation of the leaf state spaces via decision diagrams, which can be significantly smaller than the explicit enumeration of all reachable leaf states, and also more runtime efficient.

The pricing function for a leaf factor can be maintained in time low-order polynomial in the size of the leaf factor's state space. While this works well for small leaf factors, it can be prohibitive for large leaves that contain many state variables. We address this through symbolic leaf-state-space representation, resulting in a new hybrid of explicit and symbolic search, performing explicit search on the center component and symbolic search on the leaf state spaces.

Binary decision diagrams (BDDs) have been used in the past as an alternative to explicit search, as they allow to represent large state spaces compactly [Bryant, 1986; McMillan, 1993; Edelkamp and Helmert, 1999]. We build on prior work in planning that has derived efficient techniques to run symbolic search [Jensen et al., 2008; Kissmann and Edelkamp, 2011; Torralba et al., 2017].

Heuristic search is essential for the performance of planning as explicit-state search. The compilation introduced in Chapter 4.1 that allows to plug any planning heuristic into decoupled search, however, does not suit our symbolic leaves, as it requires to enumerate all leaf states. We overcome this here, by an alternative compilation to leaf *facts*, which avoids the explicit enumeration and benefits our compact symbolic leaf representation. We showcase this with the $h^{\max}$ [Bonet and Geffner, 2001], h^{FF} [Hoffmann and Nebel, 2001], and $h^{\text{LM-cut}}$ heuristic [Helmert and Domshlak, 2009].

This chapter is based on Gnad et al. [2017b]. The author of this work and Álvaro Torralba contributed equally to the conceptual development of the symbolic leaf representation and connection to heuristics. Álvaro Torralba implemented support for partial states in the existing symbolic-search extension of a version of the Fast Downward planner. The author of this work implemented the interface to decoupled states and the heuristics that utilize the symbolic representation.



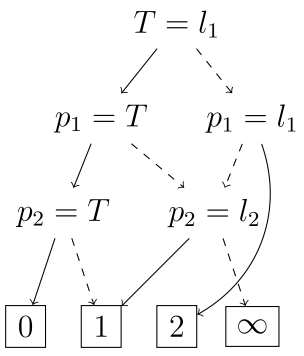


Figure 12.1: Illustration of two decision diagrams, a BDD (left) and an ADD (right) representing sets of states in our running example. Solid edges correspond to "true" assignments, dashed edges to "false" ones.

12.1 Background

Symbolic search is a state-space exploration technique that uses efficient data structures to represent and manipulate sets of states [McMillan, 1993]. *Binary Decision Diagrams* (BDDs), in particular, often yield exponential gains compared to explicit enumeration [Bryant, 1986]. BDDs can be manipulated efficiently to maintain and process sets of states, which is necessary to run the search. Algebraic Decision Diagrams [Bahar et al., 1997] are an extension of BDDs that represent functions mapping each state to a finite set of values, e. g., integers representing costs. In the context of planning, they have been used to represent heuristic functions and to perform A* search [Hansen et al., 2002]. We will use ADDs to represent the pricing function and to manipulate the prices when computing heuristics.

Figure 12.1 shows illustrations of a BDD and an ADD that represent a set of states in our logistics example. Nodes in the decision diagram correspond to variable assignments, the edges indicate if the respective fact is satisfied (solid) or not (dashed). The leaf nodes specify the set of reached states. All assignments along paths in the decision diagram from the root to a leaf represent the states that are described by the BDD (paths ending in $\top$), respectively the cost of these states for the ADD.

To perform search, actions are represented via *transition relations* (TRs). A transition relation T represents a set of actions $A' \subseteq \mathcal{A}$ of the same cost, through a BDD that contains the set of all pairs of states (s, s') such that s' is reachable from s by applying an action $a \in A'$. Given a set of states S and a transition relation T, the *image* operation computes the set of successor states of S through T, i. e., the set of all states reachable from a state in S by an action represented in T. The image operation has worst-case complexity exponential in the number of state variables, but is usually more efficient than expanding the states in S one by one. The symbolic variant of standard search algorithms is implemented by starting from the BDD representation of the initial state, and iteratively computing the image. Once the set of reached states intersects with a BDD that represents the set of goal states, the search stops and a plan can be extracted.

For further details regarding the use of BDDs and ADDs for symbolic search in classical planning tasks, we refer to [Torralba et al., 2017].

12.2 Symbolic Leaf Representation

When dealing with leaves that have a small state space, the pricing function can be kept explicitly. This requires a single entry per reachable leaf state that stores its price. Caching transitions between leaf states then allows to efficiently update the pricing function. This becomes prohibitive, however, both memory and runtime wise, for large leaf state spaces. We next introduce a symbolic representation of the pricing function using decision diagrams.

We use different types of decision diagrams to address the different requirements of the operations working on the pricing function. For each leaf L and price p, we keep a BDD B^L_p that represents all leaf states of L with price p.[1] Additionally, we compute a BDD $B^L_{\mathcal{R}}$ that represents all leaf states of L reached with finite price, as well as an ADD P^L that represents all B^L_p in a single data structure. As we will describe next, different data structures ease the computation of certain operations required to perform decoupled search. We next describe these operations and then explain how we use the decision diagrams to implement these efficiently.

(1) *CheckPre*$(s^{\mathcal{F}}, a^C)$: check if a center action a^C is applicable in a decoupled state $s^{\mathcal{F}}$.

(2) *Apply*$(s^{\mathcal{F}}, a^C)$: apply center action a^C to $s^{\mathcal{F}}$, in particular, computing the set of leaf states that satisfy the leaf preconditions of a^C, and apply a^C's leaf effects.

(3) *UpdatePrices*$(s^{\mathcal{F}})$: update the pricing function for the newly generated decoupled state $s^{\mathcal{F}}$, i. e., run a fixed-point computation with the leaf-only actions $\mathcal{A}^{\mathcal{L}}_{\not{C}}$ until no new leaf states are reached and the pricing function is stable.

(4) *CheckDominance*$(s^{\mathcal{F}}, t^{\mathcal{F}})$: check if a new state is dominated by, or dominates an existing state.

These operations must be performed directly on the symbolic representation, without enumerating all leaf states at any point, so that the complexity of each operation depends on the size of the decision diagram, not on the number of represented leaf states.

[1]We remark that the action costs are always natural numbers, and usually not very diverse, in all standard classical planning benchmarks, so having a separate BDD per price p is not an issue. For more general cost functions, it might make sense to switch to more suitable decision diagrams, like edge-valued multi-valued decision diagrams (EVMDDs) [Ciardo and Siminiceanu, 2002; Speck et al., 2018].

Given leaf independence, the operations are always performed for each leaf separately. We next describe how to do so using standard BDD and ADD operations.

To compute *CheckPre*$(s^{\mathcal{F}}, a^C)$ of a center action a^C, we encode the leaf preconditions of a^C on a leaf L as a BDD $B^L_{\mathsf{pre}(a^C)}$ that describes the set of leaf states that satisfy such preconditions. Then, a^C is applicable if the intersection of $B^L_{\mathsf{pre}(a^C)}$ and the set of reached leaf states $B^L_{\mathcal{R}}$ is non-empty.

Apply$(s^{\mathcal{F}}, a^C)$ applies a^C to the pricing function of a decoupled state $s^{\mathcal{F}}$. If a^C has preconditions on a leaf, we compute the intersection of the B^L_p BDDs representing $\mathsf{prices}(s^{\mathcal{F}})$ with $B^L_{\mathsf{pre}(a^C)}$. If a^C has an effect on a leaf, this can be applied to each B^L_p using the image operation with respect to the transition relation of a^C projected to L.

UpdatePrices$(s^{\mathcal{F}})$ is a fundamental operation in decoupled search. The price updates correspond to performing a symbolic uniform-cost search for every leaf factor $L \in \mathcal{L}$, using only those transition relations that correspond to leaf-only actions with an effect on L whose center preconditions are satisfied by $\mathsf{center}(s^{\mathcal{F}})$. The open list of this search is initialized with the previous pricing function, i. e. the B^L_p, inserting a leaf state s^L with a g-value of p. The search is run until the open list is empty, exhausting the leaf state space reachable with the center preconditions provided by $\mathsf{center}(s^{\mathcal{F}})$. After the search, the closed list represents the desired pricing function, with a new BDD B^L_p for each cost layer containing the set of leaf states with this price.

The computation of *CheckDominance*$(s^{\mathcal{F}}, t^{\mathcal{F}})$ makes use of the ADD representation of pricing functions. Dominance corresponds to the standard "lower or equal" operation on ADDs, which checks whether one ADD has lower or equal value than the other for every possible assignment. Observe that this is only efficiently doable for the basic dominance pruning $\preceq$. For augmented-cost dominance, $\preceq_{\text{aug}}$, we would need to compare the price of individual leaf states, defeating the purpose of the compact representation.

The described implementation is suitable for optimal planning, where we need to keep the price of each leaf state. If cost is not of interest, e. g., for satisficing planning or proving unsolvability, there is no need to keep the actual price of every leaf state. In that setting, it suffices to keep a single BDD $B^L_{\mathcal{R}}$ to represent the reached leaf states of each leaf $L \in \mathcal{L}$. The operations are similar, except that updates can be done with the simpler symbolic breadth first search, and dominance is performed by checking whether the set of states represented by one BDD is a subset of another.

12.3 Connecting Symbolic Leaves to Heuristics

The buy-leaves compilation introduced in Chapter 4.1 in principle allows the usage of any planning heuristic in decoupled search. This compilation depends on the decoupled state $s^{\mathcal{F}}$ for which the heuristic is computed, where a new auxiliary action $a[s^L]$ for each reached leaf state s^L is introduced, with effect s^L and cost $\mathsf{prices}(s^{\mathcal{F}})[s^L]$. Thereby, the heuristic has to "buy" a leaf state before being able to perform any other operation on the variables of the corresponding leaf. This compilation does not easily extend to the symbolic leaf representation, since we need to avoid the explicit enumeration of leaf states. Instead, we compute the heuristic based on reached leaf *facts*, and use the ADD representation of the pricing function to compute the heuristic.

A rather straightforward case are delete-relaxation heuristics, like $h^{\max}$ and h^{FF}, where it suffices to add an auxiliary action for each leaf fact. We set the cost of these actions to the minimum price of any leaf state containing that fact. This can be computed easily for each leaf by a single traversal over the ADD that represents its pricing function. The auxiliary actions are formally defined in lines 5–8 in the algorithm in Figure 12.2. Note that the adapted task (specified in line 10) is like the buy-leaves compilation, except that the leaf-state actions $a[s^L]$ are split into leaf-fact actions $a_{v,d}$ for each fact $\langle v, d\rangle \in s^L$. For delete-relaxation heuristics, the splitting does not incur any information loss. This is because the $\mathsf{bought}[L]$ facts remain true even after a $a[s^L]$ action is applied, so the heuristic can simply combine the cheapest required leaf facts across leaf states, anyway.

The case of $h^{\text{LM-cut}}$ is more complicated. The heuristic iteratively runs $h^{\max}$, and computes a disjunctive action landmark [Zhu and Givan, 2003; Hoffmann et al., 2004; Richter et al., 2008; Richter and Westphal, 2010] after each iteration. This landmark is called the *cut*, a set of non-zero cost actions at least one of which must be applied in any plan. The cost of the actions in the cut is decreased by the minimum c_{min} of their costs. This process is repeated until the value of $h^{\max}$ is 0. The final $h^{\text{LM-cut}}$ value is the sum of all c_{min}. We provide the full details of our symbolic $h^{\text{LM-cut}}$ variant in Figure 12.2.

The computation of $h^{\max}$ within $h^{\text{LM-cut}}$ can be performed as described above, by adding an auxiliary action per leaf fact. The difficulty is to determine for which of the auxiliary actions we have to reduce the cost in each iteration. For single-variable leaves, this is just the set of actions in the cut, as usual. For multi-variable leaves, however, there is no mapping from auxiliary-fact actions to leaf states, and it can be necessary to reduce the cost of several auxiliary actions that are not in the cut. For example, consider a planning task with a single leaf of two variables, where $\mathcal{D}(v_1) = \{q_1, q'_1\}$, $\mathcal{D}(v_2) = \{q_2, q'_2\}$. Let $s^{\mathcal{F}}$ be a decoupled state with two finite-price leaf states, $\mathsf{prices}(s^{\mathcal{F}})[\{v_1 = q_1, v_2 = q_2\}] = 0$, $\mathsf{prices}(s^{\mathcal{F}})[\{v_1 = q'_1, v_2 = q'_2\}] = 1$. The goal is $\{v_1 = q'_1, v_2 = q'_2\}$, so the second leaf state needs to be bought for a price of 1. Say the first iteration of $h^{\text{LM-cut}}$ finds the cut $\{a_{v_1,q'_1}\}$ with a single auxiliary action, so $c_{min} = 1$. In this case, the cost of the auxiliary action a_{v_2,q'_2} must be reduced as well, because otherwise the cost of buying the leaf state $\{v_1 = q'_1, v_2 = q'_2\}$ would be counted more than once, resulting in an inadmissible heuristic. This is because the cost of both a_{v_1,q'_1}, and a_{v_2,q'_2} is due to the same leaf state. Thus, decreasing the cost of one auxiliary-fact action may cause a reduced cost of other auxiliary actions.

Instead of directly reducing the cost of the auxiliary actions involved in the cut,

```
1  Decoupled symbolic-leaf LM-cut($\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$, $s^{\mathcal{F}}$)
2      $h \leftarrow 0$
3      $P \leftarrow ADD(\mathsf{prices}(s^{\mathcal{F}}))$
4      $\mathcal{A}_{aux} \leftarrow \{a_{v,d} \mid d \in \mathcal{D}(v), v \notin C\}$
5      $\forall d \in \mathcal{D}(v), v \in L, L \in \mathcal{L}$ :
6          $\mathsf{pre}(a_{v,d}) \leftarrow \{\mathsf{bought}[L] = \bot\}$
7          $\mathsf{eff}(a_{v,d}) \leftarrow \{v = d, \mathsf{bought}[L] = \top\}$
8          $\mathsf{cost}(a_{v,d}) \leftarrow \min_{s^L \in S^L, s^L[v]=d} P[s^L]$
9      $\mathcal{I}' \leftarrow \mathcal{I}[C] \cup \{v = \mathsf{none} \mid v \notin C\} \cup \{\mathsf{bought}[L] = \bot \mid L \in \mathcal{L}\}$
10     $\Pi' \leftarrow \langle \mathcal{V} \cup \{\mathsf{bought}[L] \mid L \in \mathcal{L}\}, \mathcal{A} \cup \mathcal{A}_{aux}, \mathcal{I}', \mathcal{G} \cup \{\mathsf{bought}[L]{=}\top \mid L \in \mathcal{L}\} \rangle$
11     while $h^{\max}(\Pi', \mathsf{center}(s^{\mathcal{F}})) > 0$ do
12         $cut \leftarrow$ Compute disjunctive action landmark
13         $c_{min} \leftarrow \min_{a \in cut} \mathsf{cost}(a)$
14         $h \leftarrow h + c_{min}$
           // Decrease cost of actions in the cut
15         foreach $a \in cut, a \notin \mathcal{A}_{aux}$ do
16             $\mathsf{cost}(a) \leftarrow \mathsf{cost}(a) - c_{min}$
17         end
           // Decrease cost of auxiliary actions
18         foreach $L \in \mathcal{L}$ do
19             $BDDcut \leftarrow \bigvee_{a_{v,d} \in cut, v \in L} BDD(v, d)$
20             $ADDcut \leftarrow c_{min} \cdot ADD(BDDcut)$
21             $P[L] \leftarrow P[L] - ADDcut$
22             $\forall d \in \mathcal{D}(v), v \in L : \mathsf{cost}(a_{v,d}) \leftarrow \min_{s^L \in S^L, s^L[v]=d} P[s^L]$
23         end
24     end
25     return $h$
```

Figure 12.2: A variant of $h^{\text{LM-cut}}$ for decoupled search with symbolic leaves.

we reduce the prices of the corresponding states in the symbolic pricing function, and recompute the cost of the auxiliary actions with respect to the new pricing function. Given a cut, we construct an ADD that assigns a value of c_{min} to all leaf states containing a fact whose corresponding auxiliary action is in the cut and 0 elsewhere. We subtract this ADD from the pricing function and recompute the cost of each auxiliary action by a new traversal over the resulting ADD. So whenever the cost of an auxiliary action a_q for a leaf fact q is decreased by c_{min}, we subtract c_{min} from every leaf state that satisfies q. This results in a non-negative price, i. e., $P[L](s^L) - c_{min} \geq 0$ for all s^L, because c_{min} is the minimum action cost of all actions in the cut so $c_{min} \leq \mathsf{cost}(a_q) \leq P[L](s^L)$.

Furthermore, this procedure will reduce the price of exactly those leaf states containing a fact whose corresponding auxiliary leaf actions would be in the cut when using the explicit $h^{\text{LM-cut}}$ implementation (modulo different tie-breaking). This is due to the fact that each leaf fact q in the initial layer of $h^{\text{LM-cut}}$'s landmark graph can only result from the application of an auxiliary leaf action. So, if the auxiliary achiever of q is in the cut in the symbolic version, so are all auxiliary leaf actions whose effect contains q in the explicit variant, and vice versa.

12.4 Experimental Evaluation

We implemented the symbolic leaf representation in our decoupled search planner based on Fast Downward [Helmert, 2006b]. To do so, we integrated the symbolic search code from the Symba planner [Torralba et al., 2014], and adopt the usage of the *CUDD* library to store and manage the symbolic leaf representations [Somenzi, 2021], as done in Symba. We reuse the standard configuration, in particular the variable ordering, of the existing symbolic search planner. The general settings, benchmarks, algorithms, and heuristics used in the evaluation are as described in Chapter 7.2. We evaluate in optimal planning, satisficing planning, and proving unsolvability. We use the IA factoring strategy throughout, since it is a simple strategy that returns good factorings in many instances. The source code and evaluation data are publicly available [Gnad, 2021a].

We start our evaluation by looking into optimal planning. We compare our new symbolic-leaf representation (sl) to explicit-state search (Base) as a reference, to our standard decoupled search configuration (DS), and a variant thereof ($\mathrm{DS}_{\preceq}$) that employs basic dominance pruning $\preceq$ instead of augmented-cost dominance $\preceq_{\mathrm{aug}}$, which, as mentioned before, is not supported for the symbolic leaves. We run these planners with blind search and A^* with $h^{\text{LM-cut}}$. In the latter setting, we include symbolic bidirectional search (SBD) and the Complementary planner ($\mathrm{C2}$) in the comparison.

Figure 12.3 shows coverage data (number of instances solved) on all planning tasks where the IA factoring strategy is successful. Observe that the symbolic leaves configuration consistently outperforms the base variant with explicit-state leaves. It only solves one instance less than that baseline $\mathrm{DS}_{\preceq}$ with $h^{\text{LM-cut}}$ in Logistics. Otherwise, it improves coverage by 1 to 2 instances across several domains, leading to an overall superior coverage. Coverage increases mostly on instances where the leaves are too big, i. e., where the construction of the leaf state spaces prior to the search runs out of memory with the explicit representation. Compared to our standard decoupled search configuration (DS), the picture is mixed. sl does still improve by up to 2 instances in several domains, but also loses coverage in five domains—most notably in DataNetwork and Elevators with blind search, and in Logistics with $h^{\text{LM-cut}}$.

In the scatter plots in Figure 12.4, we see the initialization phase of the CUDD library again, that we already observed in the evaluation in Chapter 7. CUDD preallocates a lot of memory and the initialization takes a while to complete. For non-trivial

			Blind Search				A* with $h^{\text{LM-cut}}$					
Domain	#	$\#\mathcal{F}$	Base	DS	DS$_{\preceq}$	sl	Base	DS	DS$_{\preceq}$	sl	SBD	C2
Agricola	20	20	0	0	0	0	0	0	0	0	**5**	0
Childsnack	20	20	0	0	0	0	0	0	0	0	**4**	0
DataNetwork	20	20	7	**9**	5	5	12	**14**	12	12	13	13
Depots	22	22	**4**	**4**	**4**	**4**	**7**	**7**	**7**	**7**	5	**7**
Driverlog	20	20	7	**11**	**11**	**11**	13	13	13	13	12	**15**
Elevators	30	30	13	**16**	9	9	22	23	22	22	**25**	**25**
Floortile	40	40	**2**	**2**	**2**	**2**	13	9	9	9	**34**	28
Freecell	80	42	**3**	0	0	2	2	2	2	2	2	**3**
GED	20	20	**15**	**15**	**15**	**15**	15	15	15	15	19	**20**
Grid	5	5	**1**	**1**	**1**	**1**	2	2	2	2	2	**3**
Logistics	63	63	12	**25**	**25**	**25**	26	**36**	35	34	24	28
Miconic	150	145	45	46	**47**	**47**	**136**	135	135	135	107	98
Mprime	35	6	**6**	4	4	**6**	**6**	4	4	**6**	**6**	**6**
NoMystery	20	20	8	**20**	**20**	**20**	14	**20**	**20**	**20**	14	**20**
Openstacks	80	50	**25**	20	21	21	24	20	20	20	**50**	40
Organic-Split	20	16	**6**	3	3	5	**11**	9	9	**11**	5	5
ParcPrinter	20	13	1	3	3	**4**	4	**7**	**7**	**7**	2	4
PSR	50	48	47	**48**	46	46	47	**48**	47	47	**48**	**48**
Rovers	40	40	5	**7**	**7**	**7**	7	8	8	9	**14**	13
Satellite	36	36	**5**	**5**	**5**	**5**	7	**9**	**9**	**9**	8	**9**
Scanalyzer	30	9	**6**	3	3	3	5	5	5	5	**6**	**6**
Tetris	17	13	4	**5**	**5**	**5**	5	4	4	4	2	**10**
Tidybot	30	30	14	**16**	**16**	**16**	**18**	17	**18**	**18**	7	**18**
TPP	30	29	**5**	**5**	**5**	**5**	5	5	5	5	7	**14**
Transport	59	30	**15**	**15**	13	13	14	14	14	14	**16**	**16**
Trucks	30	27	**5**	4	4	4	9	10	10	10	10	**12**
Woodworking	30	26	6	7	7	**8**	14	17	16	17	**19**	17
Zenotravel	20	20	8	**9**	8	8	**13**	**13**	12	12	10	**13**
Other	593	36	10	10	10	10	11	11	11	11	11	11
Total	1630	896	285	**313**	299	307	462	477	471	476	487	**502**

Figure 12.3: Coverage data (number of solved tasks) in optimal planning, on instances where IA does not abstain. $\#\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage (separately for blind search and A* with $h^{\text{LM-cut}}$) in **bold face**.

instances we see that the symbolic representation eventually always pays off in terms of memory usage. Concerning runtime, there is still a clear trend that favors the symbolic leaves with blind search. For A* with $h^{\text{LM-cut}}$, though, there is no clear picture, and the differences are small in general. This might be due to the repeated creation and

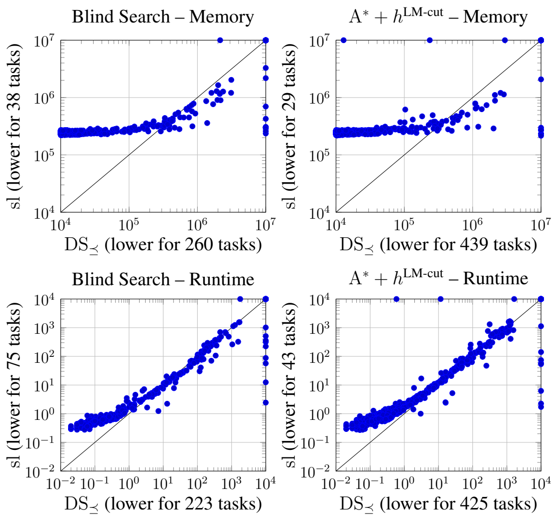


Figure 12.4: Scatter plots comparing the memory usage (top), and runtime (bottom) of sl to $\mathrm{DS}_{\preceq}$ with blind search (left), and A^* with $h^{\text{LM-cut}}$ (right). In all plots we use factorings obtained by IA.

manipulation of the ADDs that are used within the heuristic.

In satisficing planning we run greedy best-first search with the h^{FF} heuristic, both with and without preferred-operator pruning (PO). We compare to LAMA and BFWS when PO pruning is enabled. Coverage data is shown in Figure 12.5. We observe similar results as in optimal planning: sl improves over the explicit-leaf decoupled search baseline in several domains, by up to 3 instances. Without preferred-operator pruning, it loses 1 instance in coverage only in two domains, leading to an overall advantage. With PO, overall it solves 6 more instances, loses 1 in Tetris and 2 in Trucks, but gains 9 instances across five domains. We conclude that the symbolic representation pays off nicely in satisficing planning, where we only need to keep a single BDD per leaf.

This is confirmed by the scatter plots in Figure 12.6, where there is a clear runtime advantage of the symbolic representation starting from around 10s search time. In terms of memory, the advantage is rather small overall, although there are several instances that show a clear memory reduction.

Finally, we look into proving unsolvability. We compare in blind search and A^* with $h^{\max}$, including SBD and the Sympa planner in the comparison for the latter. For coverage, Figure 12.7, there is almost no difference. The main exception being the Tetris domain when running blind search. Here, the symbolic leaf representation clearly outperforms its explicit counterpart, solving 5 additional instances. Here, blind search with explicit leaves runs out of memory. With $h^{\max}$, both variants run out of time, so the deadend pruning of the heuristic does not seem to be very effective, causing a significant overhead without adequately reducing the size of the search space.

			GBFS with h^{FF}			+ PO Pruning				
Domain	#	# $\mathcal{F}$	Base	DS	sl	Base	DS	sl	LAMA	BFWS
Agricola	20	20	**10**	**10**	**10**	**12**	**12**	**12**	**12**	10
Barman	40	2	0	0	0	0	0	0	**2**	**2**
Childsnack	20	20	0	**2**	**2**	6	**20**	**20**	6	8
DataNetwork	20	20	**7**	6	6	10	7	9	**13**	9
Depots	22	22	15	**19**	**19**	19	21	21	20	**22**
Driverlog	20	20	18	**19**	**19**	**20**	**20**	**20**	**20**	**20**
Floortile	40	40	**8**	5	6	**8**	6	**8**	**8**	5
Freecell	80	42	**42**	41	41	**42**	**42**	**42**	**42**	**42**
Grid	5	5	**4**	3	3	4	**5**	**5**	**5**	**5**
Logistics	63	63	51	**63**	**63**	61	**63**	**63**	**63**	62
Maintenance	20	9	**1**	**1**	**1**	2	2	2	3	**8**
NoMystery	20	20	8	19	**20**	9	19	**20**	11	17
Organic-Split	20	19	**12**	11	10	11	11	11	**12**	**12**
ParcPrinter	30	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	19
Pathways	30	30	11	14	**15**	21	21	21	23	**30**
Rovers	40	40	**22**	21	**22**	**40**	**40**	**40**	**40**	**40**
Satellite	36	36	25	26	**29**	**36**	33	**36**	**36**	31
Tetris	20	17	**4**	**4**	3	9	11	10	**14**	13
Thoughtful	20	13	**5**	**5**	**5**	7	6	6	11	**12**
Tidybot	20	20	**16**	15	15	16	15	15	17	**18**
TPP	30	29	22	**26**	**26**	**29**	25	26	**29**	28
Trucks	30	27	**13**	**13**	**13**	**16**	15	13	**16**	14
Woodworking	40	37	31	**33**	**33**	**37**	**37**	**37**	**37**	21
Other	1000	328	328	328	328	328	328	328	328	328
Total	1686	899	673	704	**709**	763	779	785	**788**	776

Figure 12.5: Same setup as in Figure 12.3 for satisficing planning. We highlight the best coverage (separately for search with vs. without the use of preferred-operator pruning, PO) in **bold face**.

The scatter plots in Figure 12.8 also indicate that in general there is little difference between the explicit and symbolic leaf representation in this setting.

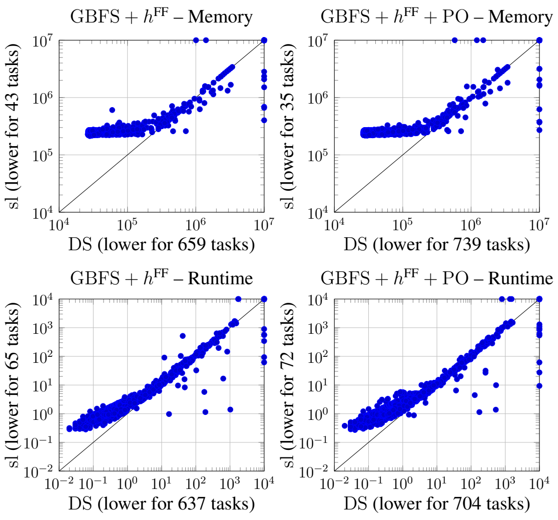


Figure 12.6: Scatter plots comparing the memory usage (top), and runtime (bottom) of sl to DS with (left), vs. without the use of preferred-operator pruning (right). In all plots we use factorings obtained by IA.

			Blind Search			A* with $h^{\max}$			Sym.	
Domain	#	#$\mathcal{F}$	Base	DS	sl	Base	DS	sl	SBD	Sympa
BagBarman	20	16	**12**	**12**	**12**	8	8	8	**14**	11
Cavediving	25	21	3	**4**	**4**	3	4	4	**6**	3
DocTransfer	20	20	**5**	**5**	**5**	7	**13**	**13**	5	10
NoMystery	23	23	2	**12**	**12**	2	**12**	**12**	5	**12**
Rovers	19	19	6	**9**	**9**	6	9	9	12	**16**
Tetris	20	20	5	5	**10**	**5**	**5**	**5**	**5**	**5**
TPP	30	30	**17**	12	12	16	15	14	21	**24**
Other	161	46	26	26	26	26	26	26	26	26
$\sum$	318	195	76	85	**90**	73	92	91	94	**107**
Unsolvable Benchmarks from Hoffmann et al. [2014]										
3-SAT	30	1	**1**	**1**	**1**	**1**	**1**	**1**	0	0
NoMystery	25	25	0	**24**	**24**	0	**25**	**25**	5	24
Rovers	25	25	0	0	**1**	1	2	3	5	**18**
TPP	25	25	**6**	0	0	6	1	0	7	**18**
Other	11	0	0	0	0	0	0	0	0	0
$\sum$	116	76	7	25	**26**	8	29	29	17	**60**
Total	434	271	83	110	**116**	81	121	120	111	**167**

Figure 12.7: Same setup as in Figure 12.3 for proving unsolvability. We highlight the best coverage (separately for blind search and A^* with $h^{\max}$) in **bold face**.

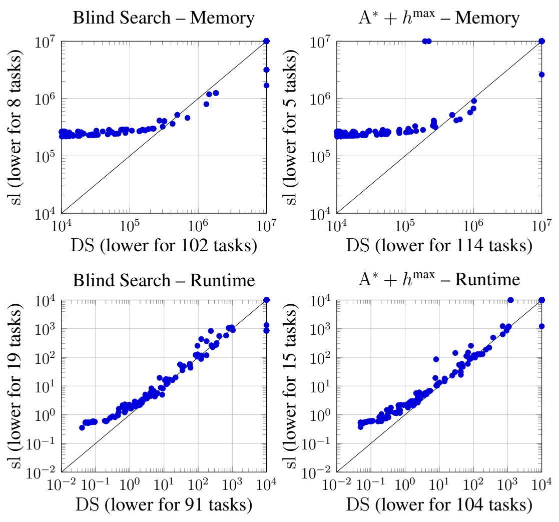


Figure 12.8: Scatter plots comparing the memory usage (top), and runtime (bottom) of sl to DS with blind search (left), and A^* with $h^{\max}$ (right). In all plots we use factorings obtained by IA.

Chapter 13

Dominance Pruning for Fork Topologies

In Part II of this work, we observed that the reachable decoupled state space can be exponentially larger than the explicit state space, even on simple toy benchmarks.[1] In this chapter, we tackle this issue by extending the concept of decoupled-state dominance relations, introducing new variants that are exponentially separated from the previous notions of dominance. We devise several more powerful criteria to identify dominated decoupled states, show that they preserve optimality, and establish their interrelations.

The worst-case exponential blow-ups often result from irrelevant distinctions in pricing functions. One means to combat this, and more generally to improve search, is dominance pruning, pruning a state $s^{\mathcal{F}}$ if a better state $t^{\mathcal{F}}$ has already been seen. But, given the complex structure of decoupled states, when is one "better" than another? For the basic criterion presented in Chapter 3.4.1, $s^{\mathcal{F}}$and $t^{\mathcal{F}}$must have the same center state and $t^{\mathcal{F}}$ needs to have cheaper prices than $s^{\mathcal{F}}$ for all leaf states. Here, we introduce advanced methods, analyzing the structure of decoupled states to identify (and then, disregard) irrelevant distinctions. We devise several such methods, using different sources of information. We characterize their relative pruning power, and show that they can yield exponential search reductions. Experiments on standard planning benchmarks attest to the possible practical benefits. The main text only outlines our proof arguments, full proofs are provided in Appendix B.2.

This chapter is based on Torralba et al. [2016]. The author of this work and Álvaro Torralba contributed equally to the adaptation of the notions of simulation relations to decoupled states and the correctness proofs of the new relations. The implementation of the novel dominance relations was done with equal contributions by both, where Álvaro Torralba focused on providing an interface to connect leaf state spaces to the existing code that identified dominance relations in explicit-state search. The author of

[1]See Chapter 3.4.2, in particular Example 5 for details.

this work took care of the implementation details for decoupled states. An early version of the implementation that is no longer utilized is due to Patrick Dubbert.

13.1 Decoupled State Dominance

In this chapter, we adopt the notion of dominance based on simulation relations from prior work [Milner, 1971; Gentilini et al., 2003; Torralba and Hoffmann, 2015], where a state s dominates a state t if s is at least as close to the goal as t. Before we introduce the new dominance relations, we extend this concept to decoupled states.

Let $\mathcal{P}^{\mathcal{F}}(s^{\mathcal{F}})$ the set of all augmented-optimal decoupled plans $\pi^{\mathcal{F}}$ for a decoupled state $s^{\mathcal{F}}$. By $d^{\mathcal{F}}(s^{\mathcal{F}}) := \min_{\pi^{\mathcal{F}} \in \mathcal{P}^{\mathcal{F}}(s^{\mathcal{F}})} |\pi^{\mathcal{F}}|$, we denote the minimum number of center actions in any augmented-optimal plan for $s^{\mathcal{F}}$.

Definition 44 (Decoupled Dominance Relation)**.** *Let* Π *be a planning task,* $\mathcal{F}$ *a factoring for* Π*, and* $s^{\mathcal{F}}$*,* $t^{\mathcal{F}}$ *two decoupled states. A binary relation* $\preceq \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{F}}$ *over decoupled states is a* decoupled dominance relation *if* $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$ *implies that* $h^*_{\mathcal{F}}(s^{\mathcal{F}}) \geq h^*_{\mathcal{F}}(t^{\mathcal{F}})$ *and* $d^{\mathcal{F}}(s^{\mathcal{F}}) \geq d^{\mathcal{F}}(t^{\mathcal{F}})$*.*

Intuitively, $t^{\mathcal{F}}$ dominates $s^{\mathcal{F}}$ if it has an at least equally good global plan and a decoupled plan that is at most as long. The decoupled plan condition is needed only in the presence of 0-cost actions, and ensures that the global plan for $t^{\mathcal{F}}$ does not have to traverse $s^{\mathcal{F}}$. In dominance pruning, given such a relation $\preceq$, we prune a state $s^{\mathcal{F}}$ at generation time if we have already seen another state $t^{\mathcal{F}}$ (i. e., $t^{\mathcal{F}}$ is in the open or closed list) such that $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$ and $g(s^{\mathcal{F}}) \geq g(t^{\mathcal{F}})$. If $t^{\mathcal{F}}$ can be reached with equal or better g-cost, pruning $s^{\mathcal{F}}$ preserves completeness and optimality of the search algorithm.

We remark that the notion of decoupled dominance relations can be generalized to augmented-cost dominance pruning against all visited states. This is easy to see since the dominance relations we introduce here only consider the remaining cost and distance of the respective decoupled states to the goal, not the cost of the paths these were reached on. Thus, cheaper-visited pruning with any relation $\preceq_X$ introduced in this chapter can be replaced by all-visited pruning with the augmented-cost variant $\preceq^X_{\text{aug}}$ of such $\preceq_X$ without sacrificing completeness or optimality. In our definitions and proofs in this chapter, we stick to $\preceq$ for simplicity.

We derive practical decoupled dominance relations by efficiently testable sufficient criteria. The relations differ in terms of their pruning power. We capture their relative power with two simple notions. First, we say that $\preceq'$ *subsumes* $\preceq$ if $\preceq' \supseteq \preceq$, i. e., if $\preceq'$ recognizes every occurrence of dominance recognized by $\preceq$. We will also show exponential separation between dominance relations, i. e., families of planning tasks in which the decoupled state space is exponential in the size of the input task under dominance pruning using $\preceq$ and polynomial when using $\preceq'$. We will devise several decoupled dominance relations, weaker and stronger ones. Weaker relations are useful in practice (only) when they cause less computational overhead.

Recall the definition of dominance pruning from Chapter 3.4: Let $s^{\mathcal{F}}$ and $t^{\mathcal{F}}$ be two decoupled states. We say that $t^{\mathcal{F}}$ *dominates* $s^{\mathcal{F}}$, denoted $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$, iff $\mathsf{center}(t^{\mathcal{F}}) = \mathsf{center}(s^{\mathcal{F}})$ and, for every leaf state $s^L \in S^{\mathcal{L}}$, $\mathsf{prices}(t^{\mathcal{F}})[s^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$.

This method simply does a point-wise comparison between $\mathsf{prices}(s^{\mathcal{F}})$ and $\mathsf{prices}(t^{\mathcal{F}})$, whenever both have the same center state. Basic dominance pruning often helps to reduce search effort, but is unnecessarily restrictive in its insistence on *all* leaf prices being cheaper. This is inappropriate in cases where $s^{\mathcal{F}}$ has some irrelevant cheaper prices. It may, indeed, cause exponential blow-ups.

Note that $\preceq$ is obviously a decoupled dominance relation. Since it requires that the set of member states of $s^{\mathcal{F}}$ is a subset of the member states of $t^{\mathcal{F}}$, *all* plans are trivially preserved and the price of the member states is cheaper in $t^{\mathcal{F}}$.

The blow-up observed in Example 5 arises because through the leaf state prices, the decoupled states "remember" the locations visited by the truck in the past. Recall that in that variant of our logistics example, the truck can drive between any pair of locations. Say there only exists a single package with goal location l_2. Then, for example the decoupled state reached through the decoupled path $\langle \text{drive}(T, l_1, l_3), \text{drive}(T, l_3, l_4)\rangle$ has finite prices for (p, l_1), (p, T), (p, l_3), and (p, l_4), and price ∞ elsewhere; while the decoupled state reached through the decoupled path $\langle \text{drive}(T, l_1, l_4)\rangle$ has finite prices for (p, l_1), (p, T), and (p, l_4). Intuitively, the difference between the two pricing functions does not matter, because, with initial location l_1, the prices for $(p, l_i), i > 2$ are irrelevant. But without recognizing this fact, the decoupled state space enumerates (pricing functions corresponding to) every combination of visited locations. We next devise several new dominance relations that tackle this shortcoming.

Throughout the remainder of this chapter, for simplicity, and without loss of generality, we will assume that every leaf $L \in \mathcal{L}$ has a single goal leaf state, denoted s^L_G. We will use $s^L_{\mathcal{I}} := \mathcal{I}[L]$ to denote the initial leaf state for each leaf $L \in \mathcal{L}$.

13.2 Frontier-Based Dominance

Our first dominance relation is based on the idea that differing prices on a leaf state s^L do not matter if "s^L has no purpose". In our example, say that we are checking whether $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$ and $\mathsf{prices}(s^{\mathcal{F}})[\{(p, l_3)\}] = 2$ while $\mathsf{prices}(t^{\mathcal{F}})[\{(p, l_3)\}] = \infty$, and thus $s^{\mathcal{F}} \not\preceq t^{\mathcal{F}}$. However, say that $\mathsf{prices}(s^{\mathcal{F}})[\{(p, T)\}] = 1$. Then the cheaper price for (p, l_3) in $s^{\mathcal{F}}$ does not matter, because the only purpose of having the package at l_3 is to load it into the truck. Indeed, the only outgoing transition of the leaf state (p, l_3) leads to (p, T).

We capture the relevant leaf states in a decoupled state $s^{\mathcal{F}}$ in terms of its *frontier*: those leaf states that are either themselves relevant (this applies only to the goal leaf state), or that can still contribute to achieving cheaper prices somewhere.

Definition 45 (Frontier)**.** *We define the* frontier *of a decoupled state* $s^{\mathcal{F}}$, $F(s^{\mathcal{F}}) \subseteq S^{\mathcal{L}}$ *as* $F(s^{\mathcal{F}}) := \{s^L_G\} \cup \{s^L \mid \exists s^L \xrightarrow{a} t^L : \mathsf{prices}(s^{\mathcal{F}})[s^L] + \mathsf{cost}(a) < \mathsf{prices}(s^{\mathcal{F}})[t^L]\}$.

From the set of frontier leaf states, we obtain a decoupled dominance relation by comparing prices only on the frontier of $s^{\mathcal{F}}$:

Definition 46 ($\preceq_F$ Dominance Relation)**.** $\preceq_F$ *is a relation over decoupled states where* $s^{\mathcal{F}} \preceq_F t^{\mathcal{F}}$ *iff* $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$ *and for all* $s^L \in F(s^{\mathcal{F}})$: $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{prices}(t^{\mathcal{F}})[s^L]$.

Theorem 31. $\preceq_F$ *is a decoupled dominance relation.*

Comparing the prices on the frontier is enough because, in any global plan for $s^{\mathcal{F}}$, if a compliant leaf path π^L decreases the price of the goal leaf state (e. g., from ∞ to some finite value), then π^L must pass through a frontier state s^L. Hence, in a global plan for $t^{\mathcal{F}}$, we can use the postfix behind s^L. This global plan can only be better than that for $s^{\mathcal{F}}$ because $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{prices}(t^{\mathcal{F}})[s^L]$.

It is easy to see that $\preceq_F$ is strictly better than $\preceq$:

Theorem 32. $\preceq_F$ *subsumes* $\preceq$ *and is exponentially separated from it.*

The first part of this claim is trivial as both relations are based on comparing prices, but $\preceq_F$ does so on a subset of leaf states. A task family demonstrating the second part of the claim is our logistics example. The only leaf action applicable in any leaf state (p, l_i) is load(T, p, l_i), leading to (p, T). However, for any reachable $s^{\mathcal{F}}$, we have $\mathsf{prices}(s^{\mathcal{F}})[\{(p, T)\}] = 1$ because this price is already achieved in the initial state, and prices can only decrease. So the only possible frontier state, apart from (p, T), is the goal (p, l_2). But only two different prices are reachable for (p, l_2), namely ∞ and 2.

13.3 Effective-Price Dominance

Our next method is based on replacing the prices in $t^{\mathcal{F}}$, i. e., the dominating state in the comparison $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$, with smaller *effective* prices, denoted $\mathsf{Eprices}(t^{\mathcal{F}})$. We then simply compare all such prices. First, we define how to compute the effective prices, then we define the dominance relation based on this.

Definition 47 (Effective Prices)**.** *The* effective pricing function $\mathsf{Eprices}(t^{\mathcal{F}}) := p$ *of a decoupled state* $t^{\mathcal{F}}$ *is defined by the point-wise minimum function* p *that satisfies:*

$$p[s^L] = \begin{cases} \mathsf{prices}(t^{\mathcal{F}})[s^L] & \textit{if } s^L = s^L_G \\ \min\{\mathsf{prices}(t^{\mathcal{F}})[s^L], \max\limits_{s^L \xrightarrow{a} t^L} \big(p[t^L] - \mathsf{cost}(a)\big)\} & \textit{otherwise} \end{cases}$$

Definition 48 ($\preceq_E$ Dominance Relation)**.** *$\preceq_E$ is a relation over decoupled states where $s^{\mathcal{F}} \preceq_E t^{\mathcal{F}}$ if and only if* $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$ *and, for all $s^L \in S^{\mathcal{L}}$,* $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{Eprices}(t^{\mathcal{F}})[s^L]$.

The modified relation based on effective prices is sound because these are designed to preserve $h^*_{\mathcal{F}}(t^{\mathcal{F}})$. In particular, *(*) for any center path π^C starting in $t^{\mathcal{F}}$, and for any leaf state s^L of leaf L, if π^L_s is a π^C-compliant leaf path from s^L to s^L_G, then there exists a path π^L from $s^L_{\mathcal{I}}$ to s^L_G that complies with $\pi^C(t^{\mathcal{F}}) \circ \pi^C$ such that* $\mathsf{cost}(\pi^L) \leq \mathsf{Eprices}(t^{\mathcal{F}})[s^L] + \mathsf{cost}(\pi^L_s)$. In other words, if $\mathsf{prices}(t^{\mathcal{F}})[s^L] > \mathsf{Eprices}(t^{\mathcal{F}})[s^L]$, then any global plan can be modified to use some other leaf state which does provide a total price of $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] + \mathsf{cost}(\pi^L_s)$ or less.

For each leaf, the effective prices can be computed by a simple backwards algorithm starting at the goal leaf state s^L_G. To illustrate the concept, consider any decoupled state $t^{\mathcal{F}}$ in our logistics example. The price of (p, T) is always 1, and its effective price is 1, too, because its successor leaf state $s^L_G = \{(p, l_2)\}$ always has an effective price ≥ 2. For any irrelevant location l_i, $i > 2$, however, due to the transition to (p, T) whose effective price is 1, we get $\mathsf{Eprices}(t^{\mathcal{F}})[\{(p, l_i)\}] = 0$ regardless of what the actual price of (p, l_i) in $t^{\mathcal{F}}$ is. Intuitively, the effective price of 0 is sound for these leaf states, because there is no global plan for $t^{\mathcal{F}}$ that uses an load(T, p, l_i) action. The cheapest compliant path will always be $\langle$load(T, p, l_1), unload$(T, p, l_2)\rangle$, since it is never required to unload the package somewhere other than the goal location, as it would need to be loaded again. So these locations have "no purpose".

Theorem 33. *$\preceq_E$ is a decoupled dominance relation.*

To prove Theorem 33, observe that, whenever $s^{\mathcal{F}} \preceq_E t^{\mathcal{F}}$, given a global plan for $s^{\mathcal{F}}$, we can construct an equally good global plan for $t^{\mathcal{F}}$ by using the same center path π^C, and, with (*) above, constructing equally good or cheaper compliant goal leaf paths. To see that (*) holds, consider any $t^{\mathcal{F}}$, center path π^C, leaf state s^L, and π^C-compliant goal leaf path π^L_s starting in s^L. In our example, e. g., say $t^{\mathcal{F}}$ is reached from $\mathcal{I}^{\mathcal{F}}$ by applying drive(T, l_1, l_3); that $\pi^C = \langle$drive$(T, l_3, l_2)\rangle$; that $s^L = \{(p, l_3)\}$; and that $\pi^L_s = \langle$load(T, p, l_3), unload$(T, p, l_2)\rangle$. Then, there exists a leaf path $\pi^L = \langle$load(T, p, l_1), unload$(T, p, l_2)\rangle$ that is compliant with $\pi^C(t^{\mathcal{F}}) \circ \pi^C$.

Formally, denote $\pi^L_s = \langle a_1, \dots, a_n\rangle$ and denote the leaf states it traverses by $s^L = s^L_0, \dots, s^L_n = s^L_G$. Observe that, as $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_n] = \mathsf{prices}(t^{\mathcal{F}})[s^L_n]$, π^L_s necessarily passes through a leaf state s^L_i whose effective and actual prices in $t^{\mathcal{F}}$ are identical. Let i be the smallest index for which that is so. Then, for all $j < i$, it holds that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_j] \neq \mathsf{prices}(t^{\mathcal{F}})[s^L_j]$, and thus by the definition of effective prices we have that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_j] \geq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_{j+1}] - \mathsf{cost}(a_{j+1})$. Accumulating these inequalities, we get that *(**)* $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_0] \geq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] - \sum_{j=1}^{i} \mathsf{cost}(a_j)$. Consider now the path π^L from $s^L_{\mathcal{I}}$ to s^L_G constructed as the concatenation of: a cheapest $\pi^C(t^{\mathcal{F}})$-compliant path to s^L_i (in our example, $\langle$load$(T, p, l_1)\rangle$); with the postfix of π^L_s behind s^L_i (in our example, $\langle \text{unload}(T, p, l_2)\rangle$). Then $\mathsf{cost}(\pi^L) = \mathsf{prices}(t^{\mathcal{F}})[s^L_i] + \sum_{j=i+1}^{n} \mathsf{cost}(a_j)$. As $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] = \mathsf{prices}(t^{\mathcal{F}})[s^L_i]$, we get $\mathsf{cost}(\pi^L) = \mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] + \sum_{j=i+1}^{n} \mathsf{cost}(a_j)$. With (**), we get the desired property that $\mathsf{cost}(\pi^L) \leq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_0] + \sum_{j=1}^{i} \mathsf{cost}(a_j) + \sum_{j=i+1}^{n} \mathsf{cost}(a_j) = \mathsf{Eprices}(t^{\mathcal{F}})[s^L] + \mathsf{cost}(\pi^L_s)$.

While this method appears orthogonal to frontier-based dominance at first sight, it turns out to subsume it:

Theorem 34. *$\preceq_E$ subsumes $\preceq_F$ and is exponentially separated from it.*

To prove the exponential separation, we extend our running example with a new $\mathrm{teleport}(l_i, l_j)$ action, for $i, j > 2$, that moves the package between irrelevant locations if the truck is at l_2. Then, as long as l_2 and at least one such l_i have not been visited yet, all leaf states (p, l_i) for $i > 2$ with finite price are in the frontier, and $\preceq_F$ suffers from the same blow-up as $\preceq$. The effective prices of (p, l_i), however, remain 0 as before.

To see that $\preceq_E$ subsumes $\preceq_F$, observe that the former can be viewed as a recursive version of the latter, when reformulating the frontier condition to "$\exists s^L \xrightarrow{a} t^L : p[s^L] < p[t^L] - \mathsf{cost}(a)$". Formally, one can show that, if $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$ holds for all frontier states $s^L \in F(s^{\mathcal{F}})$, then it also holds for all non-frontier states $s^L \notin F(s^{\mathcal{F}})$. This shows the claim as, for $s^{\mathcal{F}} \preceq_F t^{\mathcal{F}}$, we have $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{prices}(t^{\mathcal{F}})[s^L]$ on $s^L \in F(s^{\mathcal{F}})$, and thus $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{Eprices}(t^{\mathcal{F}})[s^L]$ on these states.

Note that, with the above, to evaluate $\preceq_E$ it suffices to compare the price of $s^{\mathcal{F}}$ vs. effective price of $t^{\mathcal{F}}$ on $F(s^{\mathcal{F}})$. This is equivalent to, but faster than, comparing all prices.

13.4 Simulation-Based Dominance

In this section, we devise a dominance relation that is based itself on a simulation relation over the leaf states of each leaf factor [Milner, 1971; Gentilini et al., 2003]. Thereby, similar to the last section, we can *propagate* leaf-state prices to states not actually reached, increasing the potential for dominance between decoupled states.

We use the concept of simulation relations on leaf state spaces in order to identify leaf states t^L which "can do everything that another leaf state s^L can do". In this situation, suppose that we are checking whether $s^{\mathcal{F}} \preceq t^{\mathcal{F}}$, and $\mathsf{prices}(t^{\mathcal{F}})[s^L] > \mathsf{prices}(s^{\mathcal{F}})[s^L]$, but $\mathsf{prices}(t^{\mathcal{F}})[t^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$. Then $t^{\mathcal{F}}$ can still dominate $s^{\mathcal{F}}$, because if a solution for $s^{\mathcal{F}}$ relies on s^L, then starting from $t^{\mathcal{F}}$ we can use t^L instead.[2]

[2]Note that this is quite different from the use of simulation relations on the state space for dominance pruning in explicit-state search [Torralba and Hoffmann, 2015; Torralba, 2017, 2018]. In that approach, the simulation *is* the dominance relation and the main difficulty is how to compute such a relation over the exponentially large state space. In our approach, the simulation relation is over small leaf state spaces, and it is a tool used *towards* defining a simulation relation over decoupled states.

Definition 49 (Leaf simulation)**.** *Let $L \in \mathcal{L}$ be a leaf factor. A binary relation $\preceq^L$ on L-states is a* leaf simulation *if: $s^L_G \not\preceq^L s^L$ for all $s^L \neq s^L_G$; and whenever $s^L_1 \preceq^L t^L_1$, for every transition $s^L_1 \xrightarrow{a} s^L_2$ either (i) $s^L_2 \preceq^L t^L_1$ or (ii) there exists a transition $t^L_1 \xrightarrow{a'} t^L_2$ s.t. $s^L_2 \preceq^L t^L_2$, $\mathsf{pre}(a')[C] \subseteq \mathsf{pre}(a)[C]$, and $\mathsf{cost}(a') \leq \mathsf{cost}(a)$.*

We call $\preceq^L$ the coarsest *leaf simulation if, for every leaf-simulation $\preceq'^L$, we have $\preceq'^L \subseteq \preceq^L$.*

This follows common notions, except for (i) which, intuitively, "allows t^L_1 to stay where it is", and except for allowing in (ii) different actions a' so long as they are at least as good in terms of center precondition and cost. The coarsest leaf simulation can be computed in time polynomial in the size of the leaf state space, as usual with simulation relations [Henzinger et al., 1995], and it has similar properties, such as being reflexive and transitive.

It is easy to see that, whenever $s^L \preceq^L t^L$, if a leaf path π^L_s starting in s^L complies with a center path π^C, then there exists a π^C-compliant leaf path π^L_t starting in t^L s.t. $\mathsf{cost}(\pi^L_t) \leq \mathsf{cost}(\pi^L_s)$.

Lemma 14. *Let $\preceq^L$ be a leaf simulation and π^C a center path. If $s^L \preceq^L t^L$ and there exists a path π^L_s from s^L to s^L_G compliant with π^C, then there exists a path π^L_t from t^L to s^L_G compliant with π^C such that $\mathsf{cost}(\pi^L_t) \leq \mathsf{cost}(\pi^L_s)$.*

Based on this, we define a new dominance relation that propagates the price of t^L to any leaf state s^L that is simulated by t^L:

Definition 50 ($\preceq_S$ Dominance Relation)**.** *The relation $\preceq_S$ over decoupled states is defined by $s^{\mathcal{F}} \preceq_S t^{\mathcal{F}}$ iff $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$ and for all $s^L \in S^{\mathcal{L}}$: $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L]$.*

Replacing the price of s^L by $\min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L]$ during the dominance pruning comparison is safe. If $s^L \preceq^L t^L$ and $\mathsf{prices}(s^{\mathcal{F}})[t^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$, then the price of s^L is irrelevant because, for any possible center path π^C, there exists an at least as good compliant path from t^L:

Theorem 35. *$\preceq_S$ is a decoupled dominance relation.*

It is easy to see that this is strictly better than $\preceq$:

Theorem 36. *$\preceq_S$ subsumes $\preceq$ and is exponentially separated from it.*

The first part of this claim holds simply because $\preceq^L$ is reflexive (and therefore $\min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L] \leq \mathsf{prices}(t^{\mathcal{F}})[s^L]$). For the second part, we use again our running example. Leaf simulation captures that $(p, l_i) \preceq^L (p, T)$ for all $i > 2$, since (p, T) is the only successor of any (p, l_i) and naturally $(p, T) \preceq^L (p, T)$. So, $\preceq_S$ reduces the price of such (p, l_i) to 1, avoiding the exponential blow-up.

$$\begin{tikzcd}
 & \boxed{\preceq_F} \arrow[rr] \arrow[ddr, leftrightarrow, "/" marking] & & \boxed{\preceq_E} \arrow[dr] \arrow[ddl, leftrightarrow, "/" marking] & \\
\boxed{\preceq} \arrow[ur] \arrow[drr] & & & & \boxed{\preceq_{ES}} \\
 & & \boxed{\preceq_S} \arrow[urr] & &
\end{tikzcd}$$

Figure 13.1: Summary of method interrelations. "$A \rightarrow B$": B subsumes A and is exponentially separated from it. "$A \not\leftrightarrow B$": A is exponentially separated from B and vice versa.

Inspired by Torralba and Kissmann [2015], we also employ leaf simulation to discover and remove irrelevant leaf states and leaf actions. A transition $s^L \xrightarrow{a} t^L$ is *irrelevant* if $t^L \preceq^L s^L$, or there exists another transition $s^L \xrightarrow{a'} u^L$ such that $t^L \preceq u^L$, $\mathsf{pre}(a')[C] \subseteq \mathsf{pre}(a)[C]$, and $\mathsf{cost}(a') \leq \mathsf{cost}(a)$. After removing such transitions, we run a reachability check on the leaf state space, removing unreachable leaf states. We subsequently remove leaf actions that do not induce any transition any longer. This reduces leaf state-space size, and may sometimes improve the heuristic function due to the removal of some actions.

13.5 Method Interrelations and Combination

We have already established the relation of our methods relative to $\preceq$, as well as the relation between $\preceq_E$ and $\preceq_F$. We next design a combination $\preceq_{ES}$ of $\preceq_E$ and $\preceq_S$, which combines their respective strengths, and we establish the remaining method interrelations. Figure 13.1 provides the overall picture.

The combined relation $\preceq_{ES}$ is obtained by modifying the effective prices underlying $\preceq_E$, enriching their definition with a leaf simulation, $\preceq^L$.

Definition 51 (Effective Simulation Prices)**.** *The* effective simulation pricing function $\mathsf{ESprices}(t^{\mathcal{F}}) := p$ *of a decoupled state* $t^{\mathcal{F}}$ *is defined by the point-wise minimum function* p *that satisfies:*

$$p[s^L] = \begin{cases} \mathsf{prices}(t^{\mathcal{F}})[s^L] & \textit{if } s^L = s^L_G \\ \min\{\min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L], \max\limits_{s^L \xrightarrow{a} t^L} \big(p[t^L] - \mathsf{cost}(a)\big)\} & \textit{otherwise} \end{cases}$$

We integrate the information from a leaf simulation into the effective prices by allowing s^L to take cheaper prices from simulating states t^L. This amounts to substituting $\mathsf{prices}(t^{\mathcal{F}})[s^L]$ with $\min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L]$ in the equation. We thus obtain, again, a decoupled dominance relation:

Definition 52 ($\preceq_{ES}$ Dominance Relation)**.** *$\preceq_{ES}$ is the relation over decoupled states where $s^{\mathcal{F}} \preceq_{ES} t^{\mathcal{F}}$ iff* $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$ *and for all $s^L \in S^{\mathcal{L}}$:* $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{ESprices}(t^{\mathcal{F}})[s^L]$.

Theorem 37. *$\preceq_{ES}$ is a decoupled dominance relation.*

Theorem 37 is shown by adapting the property (*) underlying the proof of Theorem 33. Say $\pi_s^L = \langle a_1, \dots, a_n \rangle$ is a π^C-compliant goal leaf path starting in s^L, traversing the leaf states $s^L = s_0^L, \dots, s_n^L = s_G^L$. Then, with the same arguments as before, there exists i such that (a) $\mathsf{ESprices}(t^{\mathcal{F}})[s_0^L] \geq \mathsf{ESprices}(t^{\mathcal{F}})[s_i^L] - \sum_{j=1}^{i} \mathsf{cost}(a_i)$, and (b) $\mathsf{ESprices}(t^{\mathcal{F}})[s_i^L] = \min_{s_i^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L]$. We construct our desired path π^L from $s_{\mathcal{I}}^L$ to s_G^L by a cheapest $\pi^C(t^{\mathcal{F}})$-compliant path to a leaf state t^L that minimizes the expression in (b), concatenated with a π^C-compliant goal leaf path π_t^L starting in t^L where $\mathsf{cost}(\pi_t^L) \leq \mathsf{cost}(\pi_s^L)$. Such π_t^L exists by the properties of leaf simulations, as in Theorem 35.

$\preceq_{ES}$ subsumes each of its components. The exponential separations therefore follow directly from the individual ones:

Theorem 38. *$\preceq_{ES}$ subsumes $\preceq_E$ and $\preceq_S$, and is exponentially separated from each of them.*

One can also construct cases where $\preceq_{ES}$ yields an exponentially stronger reduction than *both* $\preceq_E$ and $\preceq_S$, i. e., where $\preceq_{ES}$ is strictly more than the sum of its components. We complete our analysis by filling in the missing cases:

Theorem 39. *$\preceq_S$ is exponentially separated from $\preceq_E$, and therefore also from $\preceq_F$. $\preceq_F$, and therefore also $\preceq_E$, is exponentially separated from $\preceq_S$.*

13.6 Experimental Evaluation

We implemented the new dominance relations in our decoupled search planner based on Fast Downward [Helmert, 2006b]. The general settings, benchmarks, algorithms, and heuristics used in the evaluation are as described in Chapter 7.2. We use the Fork factoring strategy throughout. The source code and evaluation data are publicly available [Gnad, 2021a].

We focus our evaluation on optimal planning, where the novel dominance relations have the highest impact on search performance, showing results in satisficing planning and proving unsolvability on a small subset of the new relations only. We implemented all relations defined above, namely frontier pruning $\preceq_F$, effective-price pruning $\preceq_E$, simulation pruning $\preceq_S$, and the combination $\preceq_{ES}$. Additionally, we implemented a leaf-state-space pruning method that removes irrelevant leaf states and actions as described at the end of Chapter 13.4.

			Blind Search								A* with $h^{\text{LM-cut}}$							
Domain	#	$\#\mathcal{F}$	B	DS	aT	e	r	s	f	fser	B	DS	aT	e	r	s	f	fser
Driverlog	20	20	7	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
Logistics	63	63	12	26	26	**31**	28	28	26	**31**	26	**34**	33	**34**	33	**34**	32	**34**
Miconic	150	145	45	**46**	**46**	**46**	**46**	**46**	45	**46**	**136**	135	135	135	135	135	135	135
NoMystery	20	20	8	**20**	**20**	**20**	**20**	**20**	**20**	**20**	14	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Rovers	40	40	5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	7	**9**	**9**	**9**	**9**	**9**	**9**	**9**
Satellite	36	36	5	**6**	**6**	5	**6**	**6**	5	5	7	7	**8**	**8**	**8**	**8**	**8**	**8**
TPP	30	27	5	**23**	**23**	22	**23**	22	**23**	22	5	18	**23**	22	**23**	22	**23**	22
Woodworking	30	13	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	6	10	**11**	**11**	**11**	**11**	**11**	**11**
Zenotravel	20	20	8	**12**	**12**	**12**	**12**	**12**	11	**12**	**13**	**13**	12	**13**	12	**13**	12	**13**
Other	1221	33	7	7	7	7	7	7	7	7	8	8	8	8	8	8	8	8
Total	1630	417	106	162	162	**165**	164	163	159	**165**	235	267	272	**273**	272	**273**	271	**273**

Figure 13.2: Coverage data (number of solved tasks) in optimal planning, on instances where Fork does not abstain. $\#\mathcal{F}$ denotes the number of such instances per domain. Domains with the same coverage for all planners are summarized in "Other". We highlight the best coverage (separately for blind search and A* with $h^{\text{LM-cut}}$) in **bold face**.

In our comparison in optimal planning, we run all new dominance relations with the Anytime Decoupled A* algorithm defined in Chapter 4.2.1, since it gives the best results for fork factorings, as we have seen in our main evaluation in Chapter 7.6. As mentioned in Chapter 13.1, the new dominance relation can be employed together with augmented-cost dominance $\preceq_{\text{aug}}$, which we do throughout the optimal-planning evaluation. We compare the new pruning variants to explicit-state search (Base), decoupled A* (DS), and the anytime A* variant (aT), both with standard augmented-cost dominance $\preceq_{\text{aug}}$.

We use the following acronyms for decoupled search configurations with the new dominance relations:

- aT with $\preceq_F$ pruning: f
- aT with $\preceq_E$ pruning: e
- aT with $\preceq_S$ pruning: s
- aT with $\preceq_{\text{aug}}$ pruning, removing irrelevant states/actions in leaf state spaces: r
- r combined with f (only for non-optimal planning): fr
- aT with $\preceq_{ES}$ pruning combined with f and r: fser.

Figure 13.2 shows coverage data (number of solved tasks) in optimal planning for blind search and A* with $h^{\text{LM-cut}}$. In most domains, there are only minor changes in

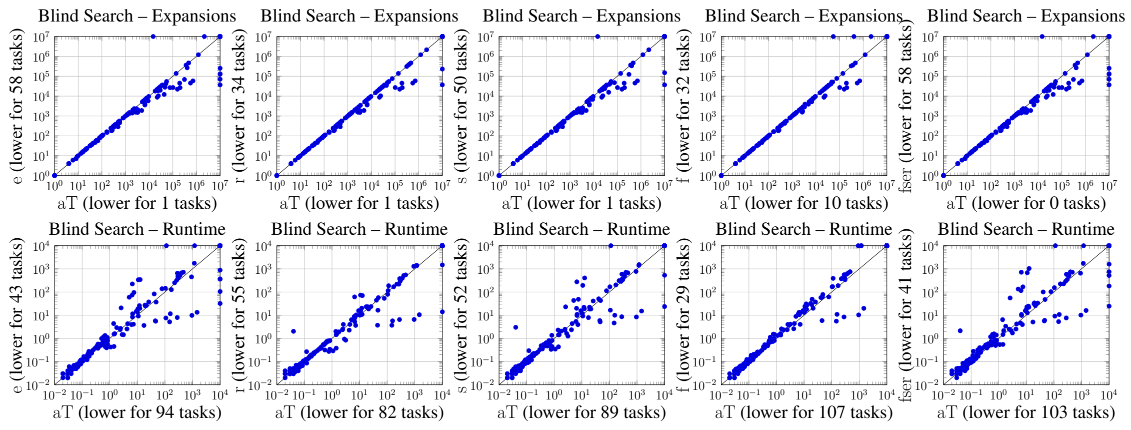


Figure 13.3: Scatter plots comparing the number of state expansions until the last f-layer in A^* (top), and runtime (bottom) of Anytime Decoupled A^* to (from left to right) pruning via effective-price dominance, leaf-state-space irrelevance pruning, simulation dominance, frontier dominance, and the combination of all methods. In all plots we run blind search.

coverage, if any. On the positive side, the only exception is Logistics, with up to 5 more tasks solved with blind search, depending on the dominance relation. With $h^{\text{LM-cut}}$, e, s, and their combination fser solve one more instance than the base version aT, and we see the same in Zenotravel. In some other domains, we observe that the more sophisticated dominance seems to have a certain overhead that does not always outweigh the additional pruning. In Miconic, Satellite, TPP, and Zenotravel, some blind-search configurations lose an instance in coverage, the same happens with $h^{\text{LM-cut}}$ also in Logistics.

In general, it looks like pruning via effective-price dominance e, simulation dominance (s), though only with $h^{\text{LM-cut}}$, and the combination of all methods fser give consistently the best results.

In the scatter plots in Figure 13.3, we see a quite similar behaviour for all methods. The search space size (number of expanded states until the last f-layer in A^*) gets reduced only by a small amount, but this can still lead to a runtime advantage of up to two orders of magnitude. While effective-price dominance and simulation dominance can cause a certain overhead, this is mostly not the case for irrelevance pruning and frontier dominance. The combination of all methods is not able to reduce the search space by more than the maximum of any subsumed method.

In satisficing planning and proving unsolvability, we only evaluate irrelevance pruning of the leaf state spaces (r), frontier dominance (f), and a combination of the two (fr). For satisficing planning, Figure 13.4, there is only a single instance in NoMystery that can additionally be solved by these methods compared to the baseline (DS).

The search space size (number evaluated states) and runtime plots in Figure 13.5

			GBFS with h^{FF}				
Domain	#	#$\mathcal{F}$	Base	DS	r	f	fr
Driverlog	20	20	18	**19**	**19**	**19**	**19**
Logistics	63	63	51	**63**	**63**	**63**	**63**
NoMystery	20	20	8	19	**20**	**20**	**20**
Pathways	30	29	10	**12**	**12**	**12**	**12**
Satellite	36	36	25	**26**	**26**	**26**	**26**
TPP	30	27	22	**23**	**23**	**23**	**23**
Woodworking	40	34	28	**30**	**30**	**30**	**30**
Other	1447	208	190	190	190	190	190
Total	1686	437	352	382	**383**	**383**	**383**

Figure 13.4: Same setup as in Figure 13.2 for satisficing planning with GBFS and h^{FF}. We highlight the best coverage in **bold face**.

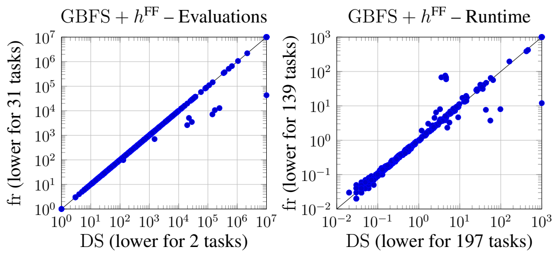


Figure 13.5: Scatter plots comparing the number of state evaluations (left), and runtime (right) of the baseline DS to fr, which combines frontier dominance with irrelevance pruning in the leaf state spaces. In all plots we run GBFS with h^{FF}.

confirm that there is no difference in most instances. Where there is an advantage in search space size (up to one order of magnitude), we also see a similar speed-up. The points above the diagonal in the runtime plot are all due to the overhead of computing and pruning irrelevant leaf states and actions using r on instances with large leaf state spaces.

For proving unsolvability, we see a similar picture. Coverage increases only in NoMystery (see Figure 13.6), although there is more margin for improvement here, so up to 8 additional instances are solved. Frontier dominance seems to have a large overhead compared to standard pruning, indicated by the fact that it solves 2 instances less than irrelevance pruning in both NoMystery and Rovers.

The search space size and runtime plots in Figure 13.7 show significant improvements across many instances with a speed-up of up to two orders of magnitude.

			A* with $h^{\max}$				
Domain	#	#$\mathcal{F}$	Base	DS	r	f	fr
NoMystery	23	23	2	12	**20**	18	18
Rovers	19	19	6	**8**	**8**	6	6
Other	276	0	0	0	0	0	0
$\sum$	318	42	8	20	**28**	24	24
Unsolvable Benchmarks from Hoffmann et al. [2014]							
NoMystery	25	25	0	**25**	**25**	**25**	**25**
Rovers	25	25	1	**2**	**2**	**2**	**2**
Other	66	0	0	0	0	0	0
$\sum$	116	50	1	**27**	**27**	**27**	**27**
Total	434	92	9	47	**55**	51	51

Figure 13.6: Same setup as in Figure 13.2 for proving unsolvability with A^* and $h^{\max}$. We highlight the best coverage in **bold face**.

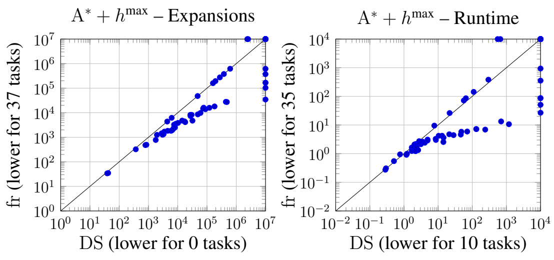


Figure 13.7: Scatter plots comparing the number of state expansions (left), and runtime (right) of the baseline DS to fr, which combines frontier dominance with irrelevance pruning in the leaf state spaces. In all plots we run A^* with $h^{\max}$.

Chapter 14

Summary

In Part III of this work, we combined decoupled state-space search with four orthogonal state-space reduction techniques, namely partial-order reduction via strong stubborn sets, symmetry breaking, symbolic state representations, and state-dominance pruning.

Strong stubborn sets pruning is a well-established technique that has originally been introduced in model checking and recently caught interest in the planning community. Both stubborn sets pruning and decoupled search have been proposed to tackle the state explosion problem inherent to many variants of explicit-state search. We combined the two methods in the context of classical planning to be able to get the best of both worlds. In fact, we proved that decoupled strong stubborn sets are exponentially separated from both component methods *on the same family* of planning tasks, i. e., the combination can be exponentially more than the sum of its parts. Our experiments illustrate that enabling strong stubborn sets pruning in decoupled search can be highly beneficial in optimal planning. By using a simple safety belt mechanism, we are able to switch off the pruning on instances where it is not useful, leading to a superior overall performance. For the future, we deem it an interesting question whether the recently introduced *fact-based* computation of stubborn sets can be adapted to the decoupled setting [Röger et al., 2020]. Extensions to *weak stubborn sets* [Sievers and Wehrle, 2021], that we discussed in Appendix A.4.1, are promising as well.

Symmetry breaking also aims at tackling the state explosion problem, being employed in many areas of computer science. Like decoupled search, symmetry breaking can exponentially reduce the search effort on its own. We extended the concept of structural symmetries to the decoupled state space and adapted the orbit-space search algorithm to decoupled search. This allows to prune symmetric parts of the decoupled search space, which can significantly improve performance. We proved that the combination of the two techniques is exponentially separated from its components, demonstrating in our evaluation that similar structures also appear in standard planning benchmarks. Interesting future work includes a more thorough investigation of the interplay between symmetry pruning and decoupled-state dominance. Applying permutations to decoupled states corresponds to moving small prices to leaf states that are placed at the front of an arbitrary order over the leaf states. It could be advantageous to permute leaf-state prices in a way that improves dominance pruning, further reducing the search space.

Our approach of symbolically representing the pricing function introduced a new hybrid of explicit and symbolic state-space search. We developed the operations required to perform decoupled search with such a representation, and connected the symbolic pricing function to standard planning heuristics. In particular, we derived a well-working adaptation of LM-cut. In our experimental evaluation, we observed that the new representation can indeed be beneficial across all algorithmic planning problems. It remains an open question if we can tailor our factoring strategies to the specifics of the symbolic representation. For instance, it could be interesting to move to larger leaf factors, since the representation of pricing functions can be a lot more compact when using, e. g., binary decision diagrams to represent the sets of reached leaf states.

Lastly, we have shown that state-dominance pruning methods can be useful for decoupled search, addressing the exponential blow-ups inherent in standard benchmarks similar to our logistics example. We provide a thorough analysis of different dominance-pruning methods for fork topologies. While these can solve the worst-case exponential blow-ups of decoupled search incurred by some leaf structures, it it an open question whether more effective techniques can be devised that more generally tackle these blow-ups without the use of hypercube pruning. For the future, we believe that further investigating decoupled-state dominance relations is an important topic. A first approach could be the extension of the relations so-far specialized to fork topologies to more general star factorings. Moreover, going from reasoning about dominance only within leaves to relations that also incorporate the center factor is very promising. In that regard, a combination with the recently introduced concept of *quantitative dominance* would be interesting [Torralba, 2017]. Also, adapting dominance variants for satisficing planning that do not preserve optimality can be beneficial for decoupled search [Torralba, 2018].

Overall, we presented the combination of several orthogonal techniques with decoupled search. Naturally, the question arises if we can combine *all* these methods in a single planning algorithm. For the combination of partial-order reduction and symmetry breaking, this has already been answered positively [Wehrle et al., 2015], and the same applies in decoupled search. Adding specialized state-dominance pruning for fork topologies to the mix is feasible, too, since it does not directly interact with the other methods. Matters get more complicated for symbolic leaf representations, since all other methods depend on the explicit representation and enumeration of leaf states. It is an open question whether, e. g., strong stubborn sets can be computed efficiently on the decision diagrams that represent the leaves without enumerating all leaf states.

Finally, so far we introduced these combinations only for classical planning. We believe that the same could be done in the context of model checking, where these methods have successfully been used to tackle the state explosion problem.

Part IV

Model Checking

Chapter 15

Introduction

In the area of formal verification, model checking addresses the question whether a desired property ϕ holds in a given system. Given a formal system description $\mathcal{M}$, the *model checking problem* is to decide whether $\mathcal{M}$ satisfies ϕ.

In particular, we look into model checking of safety and liveness properties. Safety properties express things that need to hold in any *finite* system execution, and are hence used to prove that certain "bad" things never happen. This is closely related to goal reachability that we focused on in the previous parts of this work: does there exist a state s reachable from the initial state of the system such that s violates ϕ, respectively satisfies the goal condition. This connection is well-known and has been exploited before to transfer techniques (e. g. Cimatti et al., 2003; Dräger et al., 2006; Kupferschmid et al., 2006, 2007, 2008). Herein, we adopt decoupled state-space search, which has originally been introduced in AI planning, to the context of model checking.

Liveness properties, in contrast, reason about *infinite* runs of a system, specifying desired properties that must be satisfied along any infinite execution. Liveness properties are usually considered to express "good" behaviours of the system that should occur repeatedly, i. e., infinite runs in which something good happens infinitely often.

State-space search is a common approach to solve the model checking problem for both safety and liveness properties [West, 1978; Rudin, 1987; Liu, 1989; Courcoubetis et al., 1992; Valmari, 1992; McMillan, 1993; Godefroid, 1996; Holzmann et al., 1996; Holzmann, 2004]. The approach is to explore the state space induced by the model description $\mathcal{M}$ and check if the property holds on the reachable sub-space. This ensures that the system will always satisfy the property. To formalize systems, we will consider models that are composed of several non-deterministic automata. These automata synchronize on a set of shared global labels, and can move individually using internal labels. For safety properties, we consider non-deterministic finite automata (NFA). In this case, a witness of an unsatisfied property ϕ is a finite trace, i. e., a sequence of transitions, that ends in a state that violates ϕ. Algorithmically, this corresponds to proving unsolvability in the planning context, where the state space is exhausted to prove the absence of a reachable goal state, or a counterexample, i. e., a plan, is returned.

For model checking of liveness properties, we consider non-deterministic Büchi automata (NBA). Here, a counterexample takes the form of a *lasso*, i. e., a finite system run $\rho_p(\rho_c)^\omega$ with a prefix ρ_p and a cycle ρ_c that never visits a state in which the desired property holds, so "nothing good ever happens". This captures standard liveness verification problems related to ω-regular properties. An archetypal example is automata-based checking of properties expressed in linear temporal logic (LTL) [Pnueli, 1977], where system components are represented as NBAs and are composed with a property monitor, represented as a Büchi automaton (often the negation of an LTL property). In this case, an accepting run—a lasso—witnesses a violation of a linear-time property. The predominant approach to address the verification of liveness properties using explicit state-space search is nested depth-first search (NDFS) [Courcoubetis et al., 1992; Holzmann et al., 1996; Schwoon and Esparza, 2005]. NDFS performs on-the-fly checking of liveness properties while composing the automata of $\mathcal{M}$.

We start by providing the required background on non-deterministic automata and their composition in Chapter 16. In Chapter 17, we show how automata can be composed using decoupled search, where component states reachable via internal transitions will be enumerated for each automaton, and the main search only branches over global transitions, which affect at least two components. We then look more closely into how the decoupled composition can be used to verify safety (Chapter 18) and liveness properties (Chapter 19). For safety properties, we show how decoupled search can be employed in the established SPIN model checker [Holzmann, 2004]. We illustrate in our empirical evaluation that it can significantly outperform the explicit-state search with partial-order reduction of SPIN on standard benchmarks defined in the Promela language, the input format of SPIN [PromelaManual, 2020]. For liveness checking, we adapt the nested depth-first search algorithm to the decoupled state representation, and compare the performance of a prototype implementation on a set of randomly generated models, and small showcase examples similar to the dining philosophers problem. In Chapter 20, we look into related techniques, in particular partial-order reduction via ample sets, and Petri-net unfolding. We prove that decoupled search is exponentially separated from both, also on our formulation on composed automata. We summarize this part of the work in Chapter 21.

The following chapters are based on Gnad et al. [2018a] and Gnad et al. [2021c]. Alberto Lluch-Lafuente contributed with his expertise on the SPIN model checker, Promela modeling, and the intricacies of the automata-based formulation of decoupled search. Patrick Dubbert implemented a basic version of decoupled search in SPIN for safety checking. A version of that implementation, extended by the author of this work, is used in the evaluation in Chapter 18.2. Jan Eisenhut implemented the decoupled NDFS algorithm from Chapter 19.1.3 in a new tool and conducted the experiments reported in Chapter 19.3. All other contributions are due to the author of this work.

Chapter 16

Background

This section recalls some basic notions of non-deterministic automata, their composition, the verification problems we consider in this work for such composition, and the standard algorithmic resolution based on state-space search.

16.1 Non-Deterministic Automata

We will consider non-deterministic automata (NFA) for safety checking and non-deterministic Büchi automata (NBA) for liveness checking. Since these two kinds of automata only differ in the semantics of accepting runs, we give a general formal definition of non-deterministic automata, and make the distinction in how accepting runs are defined.

Definition 53 (Non-Deterministic Automaton)**.** *A* non-deterministic automaton $\mathcal{A}$ *is a tuple* $\langle S, \rightarrow, L, s_0, A\rangle$*, where* S *is a finite set of* states*,* L *is a finite set of* transition labels*,* $\rightarrow \subseteq S \times L \times S$ *is a* transition relation*,* $s_0 \in S$ *is an* initial state*, and* $A : S \rightarrow \mathbb{B}$ *is an* acceptance function*.*

Throughout the remainder of this work, we will use the following notation convention: *non-deterministic finite automata* are denoted $\mathcal{A}_f$, *non-deterministic Büchi automata* are denoted $\mathcal{A}_B$. We will often denote transitions $(s, l, s') \in \rightarrow$, by $s \xrightarrow{l} s'$.

We next specify runs and accepting runs for both types of automata.

Definition 54 (NFA Acceptance)**.** *Let* $\mathcal{A}_f = \langle S, \rightarrow, L, s_0, A\rangle$ *be an NFA. A* run ρ *of* $\mathcal{A}_f$ *is a finite sequence of states* $(s_0, s_1, s_2, \ldots, s_n) \in S^n$ *that starts in the initial state* s_0*, such that* $0 \leq i < n : \exists l_i \in L : s_i \xrightarrow{l_i} s_{i+1} \in \rightarrow$*. A run is* accepting *if it ends in an accepting state, i. e.,* $A(s_n) = \top$*.*

For NFA, an accepting run is a finite sequence of states that starts in the initial state s_0 and ends in an accepting state. For model checking of safety properties ϕ, exactly those states $s \in S$ are accepting that *violate* ϕ, so an accepting run is a witness that ϕ is not satisfied in a reachable state.

Definition 55 (NBA Acceptance)**.** *Let $\mathcal{A}_B = \langle S, \rightarrow, L, s_0, A\rangle$ be an NBA. A* run *ρ of $\mathcal{A}_B$ is a infinite sequence of states $(s_0, s_1, s_2, \dots) \in S^\omega$ that starts in the initial state s_0, such that for all $i \geq 0 : \exists l_i \in L : s_i \xrightarrow{l_i} s_{i+1} \in \rightarrow$. A run ρ is* accepting *if it traverses accepting states infinitely often, formally $\overset{\infty}{\exists} i : A(s_i) = \top$.*

A run ρ of an NBA is an infinite sequence of states $s_0, s_1, s_2, \dots \in S^\omega$ starting from the initial state. Such a run ρ is accepting if it traverses accepting states infinitely often. Here, accepting states typically result from the negation of the property we want to verify. Thus, an accepting run is a witness for an infinite system execution during which the desired property does not hold.

We define a *trace* π of an infinite run $\rho = s_0, s_1, s_2, \dots \in S^\omega$ as a sequence of labels $\pi = l_0, l_1, \dots \in L^\omega$ such that $\forall i \in \mathbb{N} : \langle s_i, l_i, s_{i+1}\rangle \in \rightarrow$. Analogously, we consider finite traces $\pi \in L^n$ of finite runs $\rho \in S^n$.

As hinted in Chapter 15, the existence of accepting runs is interesting for several theoretical and practical reasons. On the theoretical side, the language of, e. g., an NBA, is the set of all traces σ in L^ω for which an accepting run exists. On the practical side, model checking ω-regular properties, including LTL properties, can be reduced to checking the existence of accepting runs.

16.1.1 Composition of Automata

From now on we assume that the set of labels L of an automaton is partitioned into a set L_I of *internal labels* and a set L_G of *global labels*. The notion of composition we use is based on (maximal) synchronisation on global labels, in words: in every transition involving a global label, each component having the global label in its set of labels must perform a local transition, while transitions with internal labels can be performed independently. When composing automata we assume w.l.o.g. that they do not share any internal label. Furthermore, we assume that every global label is shared by at least two component automata. Otherwise, such labels can be made internal. For a set $\mathcal{A}^1, \dots, \mathcal{A}^n$ of automata, we will use superscripting to denote the components of each $\mathcal{A}^i$, i. e., we assume $\mathcal{A}^i = \langle S^i, \rightarrow^i, L^i = L^i_I \cup L^i_G, s^i_0, A^i\rangle$.

Definition 56 (Composition of Automata)**.** *The* composition *of n automata $\mathcal{A}^1, \dots, \mathcal{A}^n$, denoted by $\mathcal{A}^1 \parallel \dots \parallel \mathcal{A}^n$, is the automaton $\langle S, \rightarrow, L, \vec{s}_0, A\rangle$, where $S = S^1 \times \dots \times S^n$, $L = \bigcup_{i\in\{1,\dots,n\}} L^i$, $\vec{s}_0 = (s^1_0, \dots, s^n_0)$, $A = \{(s_1, \dots, s_n) \mapsto \bigwedge_{i=1,\dots,n} A^i(s_i)\}$ and $\rightarrow$ is the smallest set of transitions closed under the following rules for interleaving of local transitions (1) and maximal synchronization on global labels (2):*

$$(1)\ \frac{s_i \xrightarrow{l_I} s_i' \qquad l_I \in L_I^i}{(s_1,\ldots,s_i,\ldots,s_n) \xrightarrow{l_I} (s_1,\ldots,s_i',\ldots,s_n)}$$

$$(2)\ \frac{\exists \mathcal{A}^i : l_G \in L_G^i \quad \forall j \in \{1{\le}i{\le}n \mid l_G \in L_G^i\} : s_j \xrightarrow{l_G} s_j' \quad \forall j \in \{1{\le}i{\le}n \mid l_G \notin L_G^i\} : s_j'{=}s_j}{(s_1,\ldots,s_n) \xrightarrow{l_G} (s_1',\ldots,s_n')}$$

As notation convention, we will denote component states simply by small case letters, e. g., s, and composed states $(s_1,\ldots,s_n) \in S$ by $\vec{s}$, i. e., as a vector, and similarly for local runs ρ (resp. traces π) and composed runs $\vec{\rho}$ (composed traces $\vec{\pi}$).

In Figure 16.1 we illustrate a small example of a composition of two automata $\mathcal{A}^1, \mathcal{A}^2$. In the left of the figure, we show the local state space of the two components ($\mathcal{A}^1$ top, $\mathcal{A}^2$ bottom), where the component states are $S^1 = \{1,2,3\}$, $S^2 = \{A,B\}$, and the labels are defined as $L_G^1 = L_G^2 = \{l_G^1, l_G^2\}$, $L_I^1 = \{l_I^1\}$, $L_I^2 = \{l_I^2\}$. A local state is accepting for $\mathcal{A}^1$, so $A^1(s) = \top$, iff $s = 2$, and similarly $A^2(s) = \top$ iff $s = B$. The initial states are $s_0^1 = 1$ and $s_0^2 = A$. The transitions are as shown. In the right, we depict the part of the state space of the composition $\mathcal{A}^1 \parallel \mathcal{A}^2$ reachable from $\vec{s}_0 = (1,A)$ as it would be generated by, for example, a standard depth-first search. Here, transitions via global labels synchronize the components, internal transitions (dashed) are executed independently. The states crossed out would be pruned by duplicate checking, the underlined state is accepting.

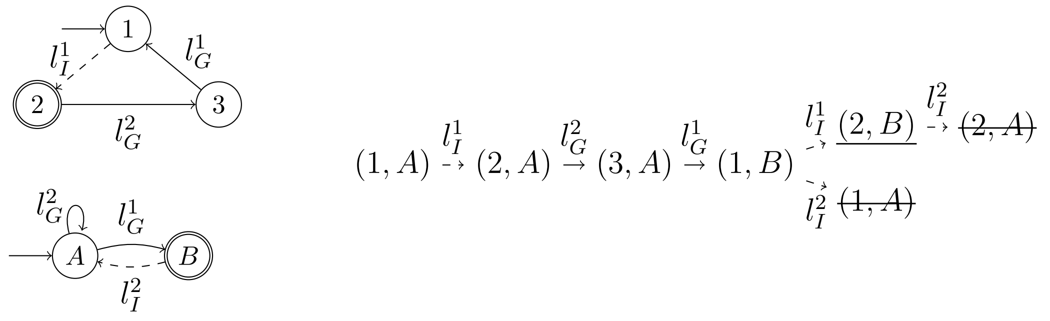

1
2
3
l_I^1
l_G^1
l_G^2
l_G^2
l_G^1
A
B
l_I^2
$(1,A) \xrightarrow{l_I^1} (2,A) \xrightarrow{l_G^2} (3,A) \xrightarrow{l_G^1} (1,B)$
l_I^1 $\underline{(2,B)}$ l_I^2 ~~(2, A)~~
l_I^2 ~~(1, A)~~

Figure 16.1: Example of two non-deterministic automata, $\mathcal{A}^1$ (top left) and $\mathcal{A}^2$ (bottom left), and the state space of their composition $\mathcal{A}^1 \parallel \mathcal{A}^2$ (right).

```
1  **CheckSafety**($\mathcal{A}_f^1 \parallel \ldots \parallel \mathcal{A}_f^n$):
2      Stack $\leftarrow \langle \vec{s}_0 \rangle$
3      $V \leftarrow \emptyset$
4      DFS($\vec{s}_0$)
5      **return** empty

6  **DFS**($\vec{s}$):
7      **if** $A(\vec{s})$ **then return** run
8      $V \leftarrow V \cup \{\vec{s}\}$
9      **foreach** $\vec{t}$ s.t. $\vec{s} \rightarrow \vec{t}$ **do**
10         **if** $\vec{t} \in V$ **then continue**
11         push(Stack, $\vec{t}$)
12         DFS($\vec{t}$)
13         pop(Stack)
14     **end**
```

Figure 16.2: A standard DFS algorithm for reachability analysis in composed NFAs.

16.2 The Model Checking Problem

In this work, we consider two types of properties that we want to verify, safety properties and liveness properties.

Safety properties ϕ are conceptually easier to verify, since this only requires to prove the absence of a reachable accepting state s, where s violates ϕ. Hence, any complete search algorithm can be employed on the composition of NFA $\mathcal{A}_f^1 \parallel \ldots \parallel \mathcal{A}_f^n$ that forms the system model. The absence of a reachable accepting state then shows that all reachable states, and thus the entire system, satisfy ϕ. In contrast, if an accepting state, so an accepting run, is found, this is a counterexample showing that ϕ does not hold.

In practice, the most-common approach to verify safety properties is to build the composed state space on-the-fly using a depth-first search (DFS) procedure, which is preferred due to its memory efficiency. Often such an approach is combined with a pruning technique that ignores certain states or transitions if it can prove that this still guarantees completeness. The details of an algorithm for safety checking based on DFS are specified in Figure 16.2. The state space of the composition $\mathcal{A}_f^1 \parallel \ldots \parallel \mathcal{A}_f^n$ is explored starting in the initial state $\vec{s}_0$. A set V is used to record already visited states and prune duplicates (line 10), and recursion enforces the depth-first exploration order of the state space. Moreover, a stack $Stack$ keeps track of the states on the current trace being explored. If an accepting state is generated (line 7), the corresponding accepting run is reported. In the absence of an accepting run, **CheckSafety** returns "empty".

$$\begin{tikzcd}(1,A) \arrow[r, dashed, "l_I^1"] & (2,A) \arrow[r, "l_G^2"] & (3,A) \arrow[r, "l_G^1"] & (1,B) \arrow[r, dashed, "l_I^1"] & \underline{(2,B)}\end{tikzcd}$$

Figure 16.3: Example run of a depth-first search.

In Figure 16.3, we show the search space generated by DFS on our example from

```
CheckEmptiness($\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$):
    Stack ← $\langle \vec{s}_0 \rangle$
    $V \leftarrow \emptyset$
    $V' \leftarrow \emptyset$
    DFS($\vec{s}_0$)
    return empty

NestedDFS($\vec{s}$):
    foreach $\vec{t}$ s.t. $\vec{s} \rightarrow \vec{t}$ do
        if $\vec{t} \in V'$ then continue
        if $\vec{t} \in Stack$ then return cycle
        $V' \leftarrow V' \cup \{\vec{t}\}$
        NestedDFS($\vec{t}$)
    end

DFS($\vec{s}$):
    $V \leftarrow V \cup \{\vec{s}\}$
    foreach $\vec{t}$ s.t. $\vec{s} \rightarrow \vec{t}$ do
        if $\vec{t} \in V$ then continue
        push(Stack, $\vec{t}$)
        DFS($\vec{t}$)
        pop(Stack)
    end
    if $A(\vec{s})$ then
        NestedDFS($\vec{s}$)
        $V' \leftarrow V' \cup \{\vec{s}\}$
    end
```

Figure 16.4: A standard NDFS algorithm for lasso search in composed NBAs.

Figure 16.1. As an accepting state, namely $(2, B)$, is reached, we found a witness showing that an undesired state is reachable, so the system does not satisfy the given safety property.

The verification problem for liveness properties corresponds to the existence of accepting runs in the composed NBA $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$. In words, we look for runs in $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$ that infinitely often traverse states in which *all* component NBAs are in an accepting state. We discuss alternative acceptance conditions in Chapter 21.

Determining the existence of accepting runs in an NBA can be boiled down to the existence of so-called *lassos*, i. e., finite sequences of states in the NBA of the form $\vec{\rho}_p \vec{\rho}_c$ where $\vec{\rho}_p$ is the prefix of the lasso and $\vec{\rho}_c$ is the cycle of the lasso, which contains at least one accepting state and closes the cycle (i. e., assuming $\vec{\rho}_p = \vec{s}_1, \ldots, \vec{s}_n$ and $\vec{\rho}_c = \vec{s}_{n+1}, \ldots, \vec{s}_m$, then $\vec{s}_m = \vec{s}_n$, and there exists a state $\vec{s}_i$ with $n < i \leq m$ where $A(\vec{s}_i) = \top$). Such a finite sequence of states represents an accepting run $\vec{\rho}_p(\vec{\rho}_c)^\omega$.

Several algorithms can be used to check the existence of lassos. The predominant family of algorithms are the variants of nested depth-first search (NDFS), originally introduced in [Courcoubetis et al., 1992]. Figure 16.4 shows the pseudo-code for one such variant, based on NDFS as presented in Clarke et al. [2001]. The algorithm is based on an ordinary depth-first search algorithm that works as explained before. The main difference with respect to ordinary DFS is that a second, nested, depth-first search algorithm (**NestedDFS**) is invoked from accepting states on backtracking, i. e., after the recursive call to **DFS**. The idea is that, if this second depth-first search finds a state that is on $Stack$, then it is guaranteed that a cycle has been found, which contains at least one accepting state. That is, one finds the (un)desired lasso. The algorithm is also complete:

$$\begin{tikzcd}
(1,A) \arrow[r, dashed, "l_I^1"] & (2,A) \arrow[r, "l_G^2"] & (3,A) \arrow[r, "l_G^1"] & (1,B) \arrow[r, dashed, "l_I^1"] & \underline{(2,B)} \arrow[r, dashed, "l_I^2"] \arrow[d, squiggly] & \cancel{(2,A)} \\
 & & & & \underline{(2,B)} \arrow[r, dashed, "l_I^2"] & (2,A) \arrow[ullll, dotted, bend left]
\end{tikzcd}$$

Figure 16.5: Example run of **CheckEmptiness**. The wavy arrow indicates the invocation of **NestedDFS**$((2, B))$; the dotted arrow indicates how the cycle is closed.

no accepting cycle is missed.

In Figure 16.5, we illustrate an example run of the **CheckEmptiness** algorithm on our example. When **DFS** backtracks from $(2, B)$, **NestedDFS** is invoked, illustrated by the wavy arrow. **NestedDFS** generates the successor $(2, A)$, which is on *Stack*, so a cycle is reported. We can construct an accepting run $\vec{\rho}_p(\vec{\rho}_c)^\omega$ with prefix $\vec{\rho}_p$ induced by the trace l_I^1 and cycle $\vec{\rho}_c$ induced by the trace $l_G^2, l_G^1, l_I^1, l_I^2$.

Chapter 17

Decoupled Composition of Automata

In the context of composed automata, decoupled state-space search can be instantiated by searching over global transitions, enumerating all local states reachable via only internal labels for each component automaton. This relaxes the concept of the *star topology* enforced in the AI planning setting, since there is no need for a center component, anymore. Instead, if a transition involves more than one automaton, it needs to be considered in the global search. An advantage over the formulation in planning is the simplicity of this decomposition. There is no need to partition the state variables to obtain a star factoring, nor to analyze potentially complex cross-factor dependencies.

In contrast to the explicit construction of the state space, where all reachable states are generated by searching over all traces of enabled transitions, decoupled search then only searches over traces of global transitions, the ones that synchronize the component automata. A *decoupled state* $s^{\mathcal{D}}$ compactly represents an exponential set of composed states closed under internal steps, which results from the cross-product of local states reached for each component.

We next introduce decoupled search for automata models, defining the decoupled state space for composed automata. In Chapter 17.2, we prove that the decoupled composition captures reachability of component states exactly.

17.1 Decoupled Composition

We formally introduce the *decoupled composition of automata*, which adapts the composition operation provided in Definition 56 to decoupled state-space search:

Definition 57 (Decoupled composition of Automata)**.** *The* decoupled composition *of n automata $\mathcal{A}^1, \dots, \mathcal{A}^n$, denoted by $\mathcal{A}^1 \parallel_{\mathcal{D}} \dots \parallel_{\mathcal{D}} \mathcal{A}^n$, is the automaton $\langle S^{\mathcal{D}}, \rightarrow_{\mathcal{D}}, L_G, s_0^{\mathcal{D}}, A^{\mathcal{D}} \rangle$ defined as follows:*

- $S^{\mathcal{D}} = \mathcal{P}^+(S^1) \times \dots \times \mathcal{P}^+(S^n)$*, with* $\mathcal{P}^+(S) := 2^S \setminus \emptyset$.

- *$s_0^{\mathcal{D}} = \langle iclose(s_0^1), \ldots, iclose(s_0^n)\rangle$, with $iclose(s)$ being the set of states s' that are reachable from s in $\mathcal{A}^i$ using only $\mathcal{A}^i$'s internal transitions L_I^i:*

  $iclose(s) = \{s' \mid s \xrightarrow{l_I \in L_I^i}{}^* s'\}$ and $iclose(S) = \bigcup_{s \in S} iclose(s)$.

- *$A^{\mathcal{D}}(s^{\mathcal{D}}) = \top \Leftrightarrow \forall \mathcal{A}^i : \exists s^i \in S_i : A^i(s^i){=}\top$, where $s^{\mathcal{D}}{=}\langle S_1, \ldots, S_n\rangle$.*

- *$\rightarrow_{\mathcal{D}}$ is the smallest set of transitions closed under the following rule:*

$$\frac{l_G \in L_G \quad \forall 1 \leq i \leq n : S'_i = \{s'_i \mid \vec{s} \in s^{\mathcal{D}} : \vec{s} \xrightarrow{l_G} (s'_1, \ldots, s'_i, \ldots, s'_n)\} \quad S'_i{\neq}\emptyset}{s^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} \langle iclose(S'_1), \ldots, iclose(S'_n)\rangle}$$

where, abusing notation, we write $\vec{s}{\in}s^{\mathcal{D}}$ if $s^{\mathcal{D}}{=}\langle S_1, \ldots, S_n\rangle$ and $\vec{s} \in S_1 \times \ldots \times S_n$.

In the decoupled composition $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$, a decoupled state $s^{\mathcal{D}}$ is defined by a tuple $\langle s^{\mathcal{D}}[\mathcal{A}^1], \ldots, s^{\mathcal{D}}[\mathcal{A}^n]\rangle$, consisting of a non-empty set of component states $s^{\mathcal{D}}[\mathcal{A}^i] \subseteq S^i$ for each $\mathcal{A}^i$. A decoupled state represents exponentially many *member states*, namely all composed states $\vec{s} = (s_1, \ldots, s_n)$ such that $\vec{s} \in s^{\mathcal{D}}[\mathcal{A}^1] \times \cdots \times s^{\mathcal{D}}[\mathcal{A}^n]$. As a notation convention, we will use a superscript $\mathcal{D}$ to denote decoupled states $s^{\mathcal{D}}$.

We overload the subset operation $\subseteq$ for decoupled states $s^{\mathcal{D}}$ by doing it componentwise on the sets of reached local states, namely $s^{\mathcal{D}} \subseteq t^{\mathcal{D}} \Leftrightarrow \forall \mathcal{A}^i : s^{\mathcal{D}}[\mathcal{A}^i] \subseteq t^{\mathcal{D}}[\mathcal{A}^i]$.

During a search in the decoupled composition we define the *global trace* of a decoupled state $s^{\mathcal{D}}$, denoted $\pi^G(s^{\mathcal{D}})$, as the sequence of global transitions on which $s^{\mathcal{D}}$ was reached from $s_0^{\mathcal{D}}$. For DFS, as considered in this work, this is well-defined.

In explicit state search, states that have been visited before—duplicates—are pruned to avoid repeating the search effort unnecessarily. The corresponding operation in decoupled search is *dominance pruning*, as previously introduced in the context of planning (cf. Chapter 3.4). A newly generated decoupled state $t^{\mathcal{D}}$ is pruned if there exists a previously seen decoupled state $s^{\mathcal{D}}$ that *dominates* $t^{\mathcal{D}}$, i. e., where $t^{\mathcal{D}} \subseteq s^{\mathcal{D}}$. With the correctness result given below, this is safe.

The initial decoupled state is obtained by closing each local state with internal steps (iclose), and decoupled transitions generate decoupled states whose local states are also closed under internal steps. This maximally preserves the decomposition afforded by the decoupled representation. Namely, as we will prove in what follows, a decoupled state $s^{\mathcal{D}}$ compactly represents all explicit states that are reachable via traces that extend the global trace $\pi^G(s^{\mathcal{D}}) = l_G^1, l_G^2, \ldots, l_G^k$ with local transition labels. That is, for every component $\mathcal{A}^i$, $s^{\mathcal{D}}$ contains the non-empty subset of its local states $s^{\mathcal{D}}[\mathcal{A}^i] \subseteq S^i$ that can be reached with traces $\pi_i = l_1, l_2, \ldots, l_n$ such that there exist indices $j_1 < j_2 < \cdots < j_k$ where l_{j_t} is the j_t-th element of $\pi^G(s^{\mathcal{D}})$ for all $1 \leq t \leq k$. In words, after every global label on $\pi^G(s^{\mathcal{D}})$, arbitrary enabled sequences of internal transitions are allowed.

Given a global trace $\pi^G = l_G^1, \ldots, l_G^k$, we can efficiently reconstruct a composed trace $\vec{\pi}$ by filling in the required internal labels between every pair of global labels, doing so separately for each component. This can be done in time polynomial in the size of the component and linear in the length of π^G.

We remark that the decoupled composition of a set of automata is always deterministic. For every pair of decoupled state $s^{\mathcal{D}}$ and global label l_G, there is a unique successor $t^{\mathcal{D}}$. This is easy to see, since, if there is a composed state $\vec{s}$ contained in $s^{\mathcal{D}}$ that has multiple outgoing transitions labelled with l_G, all of the composed successor states are contained in $t^{\mathcal{D}}$. This increases the possible state-space reduction compared to standard search, which needs to consider all these successors separately. Note that this is different from the determinization of automata, which comes with a blow-up [Roggenbach, 2001]. The determinism is a consequence of the compact representation where all possible outcome states of a non-deterministic transition are contained in the decoupled successor state.

17.2 Correctness

In this section we show that decoupled search, as presented here, is sound and complete with respect to reachability properties, adapting the corresponding result from AI planning as presented in Chapter 3.3.

We require some additional notation. For a trace $\vec{\pi}$, by $\pi^G(\vec{\pi})$ we denote the subsequence of $\vec{\pi}$ that is obtained by projecting onto the global labels L_G.

The decoupled state space captures reachability of the composed system exactly:

Theorem 40. *A state $\vec{t}$ of a composition of automata $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ is reachable from a state $\vec{s}$ via a trace $\vec{\pi}$, iff there exist decoupled states $s^{\mathcal{D}}$, $t^{\mathcal{D}}$ in the decoupled composition $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$, such that $\vec{s} \in s^{\mathcal{D}}$, $\vec{t} \in t^{\mathcal{D}}$, and $t^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$ via $\pi^G(\vec{\pi})$.*

Proof. Let $\pi^G(\vec{\pi}) = l_G^1, \ldots, l_G^k$, and $s_i^{\mathcal{D}} \xrightarrow{l_G^{i+1}}_{\mathcal{D}} s_{i+1}^{\mathcal{D}}$ for all $1 \leq i < k$. We prove the claim by induction over the length of $\pi^G(\vec{\pi})$. For the base case $|\pi^G(\vec{\pi})| = 0$, the claim trivially holds, since, by the definition of iclose(), $s^{\mathcal{D}}$ contains exactly the composed states $\vec{t}$ that are reachable from any $\vec{s} \in s^{\mathcal{D}}$ via only internal transitions.

Assume a decoupled state $s_i^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$ via $l_G^1, \ldots, l_G^i$. Then, by the definition of decoupled transitions and iclose(), the state $s_{i+1}^{\mathcal{D}}$ contains all composed states $\vec{s}_{i+1}$ that are reachable from any state $\vec{s}_i \in s_i^{\mathcal{D}}$ via a trace $\pi^{i \rightarrow i+1}$ that consists of only internal transitions and l_G^{i+1}. By hypothesis, we can extend the traces reaching every such $\vec{s}_i$ from a $\vec{s} \in s^{\mathcal{D}}$ by $\pi^{i \rightarrow i+1}$ and obtain a trace reaching $\vec{s}_{i+1}$ from $\vec{s}$ with global sub-trace $l_G^1, \ldots, l_G^i, l_G^{i+1}$.

For the other direction, if a composed state $\vec{s}_i$ is reached in a decoupled state $s_i^{\mathcal{D}}$ and can reach a state $\vec{s}_{i+1}$ via a trace $\pi^{i \rightarrow i+1}$ that consists of internal labels and l_G^{i+1}, then there exists a decoupled transition $s_i^{\mathcal{D}} \xrightarrow{l_G^{i+1}}_{\mathcal{D}} s_{i+1}^{\mathcal{D}}$ and, again by the definition of decoupled transitions and iclose(), $s_{i+1}^{\mathcal{D}}$ contains $\vec{s}_{i+1}$. By hypothesis $s_i^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$, where $\vec{s}_i$ is reachable from $\vec{s} \in s^{\mathcal{D}}$. Thus, $s_{i+1}^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$ via $l_G^1, \dots, l_G^i, l_G^{i+1}$. □

The missing piece is to show that dominance pruning preserves completeness:

Proposition 13. *Let $\mathcal{A}^1 \parallel_{\mathcal{D}} \dots \parallel_{\mathcal{D}} \mathcal{A}^n$ be the decoupled composition of n automata, and $s^{\mathcal{D}}$ and $t^{\mathcal{D}}$ two decoupled states where $s^{\mathcal{D}} \subseteq t^{\mathcal{D}}$. Then, for every transition $s^{\mathcal{D}} \xrightarrow{l_G} s_i^{\mathcal{D}}$ in the decoupled composition, $t^{\mathcal{D}} \xrightarrow{l_G} t_i^{\mathcal{D}}$ also is a transition, and $s_i^{\mathcal{D}} \subseteq t_i^{\mathcal{D}}$.*

Proof. Observe that $s^{\mathcal{D}} \subseteq t^{\mathcal{D}}$ implies that for all components $\mathcal{A}_i$, $s^{\mathcal{D}}[\mathcal{A}_i] \subseteq t^{\mathcal{D}}[\mathcal{A}_i]$. So, trivially, l_G is enabled in $s^{\mathcal{D}}$, and $t^{\mathcal{D}} \xrightarrow{l_G} t_i^{\mathcal{D}}$ is a transition in the decoupled composition.

To see that $t_i^{\mathcal{D}}$ dominates $s_i^{\mathcal{D}}$, note that the set of local states reached in $t^{\mathcal{D}}$ that have an l_G-transition can only be larger than in $s^{\mathcal{D}}$. Thus, the set of local successor states in $t_i^{\mathcal{D}}$ is a superset of that set in $s_i^{\mathcal{D}}$, and this superset relation is preserved by the iclose() operation. □

From Theorem 40 we know that a composed state $\vec{s}$ is reachable from the initial state $\vec{s}_0$ iff there exists a decoupled state $s^{\mathcal{D}}$ that contains $\vec{s}$ and that is reachable from $s_0^{\mathcal{D}}$. With Proposition 13, which shows that dominance pruning preserves state reachability, running any complete search algorithm with dominance pruning on the decoupled composition ensures completeness of the overall approach.

Chapter 18

Decoupled Search for Safety Checking

In this chapter, we introduce decoupled search for verifying safety properties in the SPIN model checker [Holzmann, 2004]. Decoupled search is not the first planning technique adopted in SPIN, heuristic search methods, for example, have been adapted to the system before [Edelkamp et al., 2001, 2004b]. Moreover, compilations from the Promela language, the input format of SPIN, to planning languages have been designed [Edelkamp, 2003a]. This shows that there is a close relation between goal reachability in the planning context and safety checking as employed in SPIN.

Consequently, with the correctness result from the previous chapter (Theorem 40), which extends the analogous result from planning, we can directly employ decoupled search with any complete search algorithm to verify safety properties. We start by illustrating how Promela models can be decomposed and discussing some specifics of our implementation in Chapter 18.1. An empirical evaluation of our decoupled search extension of SPIN is provided in Chapter 18.2.

18.1 Implementation in SPIN

In its formulation in AI planning, the input planning task needs to be decomposed into factors identified by a partition of the state variables. Two factors *interact* if there is an action reading or updating state variables from both of them. Decoupled search requires the component interactions to take the form of a star topology, where there is a *center* component to which all interactions are incident. All other components are then referred to as *leaves*. Given such a topology, the leaves depend only indirectly on each other, via the center. In the automata-based formulation presented in the previous section, there is no more center component. Instead, transitions are *global* if the corresponding label is shared by at least two automata, all other transitions are *internal*.

Promela models are defined by a set of *processes*, each with its own internal structure consisting of control-flow statements and local variables. The processes work on a set of shared global variables and channels, that can be read and written by each process. Here, each of the processes forms a non-deterministic automaton, and global variables and channels, which have a finite domain of possible values, can be interpreted as automata, too. These global structures can be seen as automata without internal labels, so more or less correspond to the center component that we have in planning. Then each process is a “leaf” component, but everything that is affected by more than a single process is grouped into the “center” component. A star-topology decomposition arises directly from the formulation as processes interacting with global data-structures.

Concretely, each of the statements in a process corresponds to either an internal transition, affecting only local variables or advancing the process location; or a global transition, namely a channel operation, a statement that affects a global variable, or a run command invoking a new process. The annotation of statements to become internal or global transitions is done fully automatic. Global and local states are defined as assignments to the respective parts of the model.

We implemented decoupled search in version 6.4.7 of SPIN, focusing on safety properties only. Our implementation is preliminary in that it does not handle the full Promela language accepted by SPIN itself. We specify the handled fragment below.

Our implementation of decoupled search is minimally intrusive. We keep SPIN’s current state pointer to store the global state s_G. Alongside that pointer, we maintain a data structure storing the current associated set $S^L(s_G)$ of reached local states. Decoupled search is then adopted as follows: the primary search only branches over global transitions, i. e., transitions that affect global data-structures. We loop over $S^L(s_G)$ to determine the global transitions enabled by the reached local states. A global transition with a label l_G applied to a decoupled state $\langle s_G, S^L(s_G)\rangle$ can allow updates in other processes. We compute the set $S^L(s_G)|_{l_G}$ of compliant local states, and apply the local updates of l_G to the states in that set. Afterwards, $S^L(s_G)|_{l_G}$ is augmented by all reachable local states to obtain the successor decoupled state $\langle r_G, S^L(r_G)\rangle$. We perform duplicate checking over decoupled states, testing the global states first to save runtime.

The remaining issue with our implementation is SPIN’s parsing process. Due to the generation of model-specific code, the distinction between internal and global transitions cannot be identified anymore within the verifier itself, but must be identified at Promela level. SPIN’s parsing process must be extended to identify the internal-vs-global information, and to communicate that to the verifier. Currently, our implementation supports this for *assignments*, *conditions*, basic control constructs (`do`...`od`, `if`...`fi`), all unary and binary *operators*, channel operations (*send*/*receive*, both synchronous and buffered), and `run` commands. We do not support `timeout`, `unless`, and channel *polling* statements (`empty`/`full`/...), nor the `priority` and `provided` process constraints, nor more complex constructs like `c-code` and `inline`. For `atomic` and `d_step` sequences, we handle basic compounds of statements, series of conditions, assignments, and channel operations, but not more complex control flows.

18.2 Experimental Evaluation

We performed experiments on several case studies, selected to suit the Promela fragment we can currently handle, and selected to showcase the potential of decoupled search. The experiments are preliminary and they serve mostly to illustrate the possible reduction power of decoupled search. Our implementation and all models used in the below are publicly available [Gnad et al., 2021a].

We run the scalable variant of Peterson's Mutex algorithm from Lynch [1996], an elevator control model developed by Armin Biere and used as benchmark in several papers (e. g. Edelkamp et al., 2001, 2004b), the X.509 protocol from Jøsang [1995], and a client-server communication protocol. The latter is a toy example we created for the purpose of this study, as a simple pattern to highlight the kind of structure relevant to decoupled search. The model consists of a server process handling requests from a scalable number of client processes; communication is via two channels. In decoupled search, all processes become leaf components, and the technique is beneficial if there are local transitions within each client. To show this, we experiment with two variants, *EmptyC* where the clients do nothing other than communicating with the server, and *NonEmptyC* where each client increments a local variable from 0 to 1. For illustrative purposes, we also include a logistics planning example very similar to the one used in the previous parts of this work. The model has a single truck which can move between two locations, and a scalable number of packages. Modeling this in Promela is straightforward. We scale the number of packages from 1 to 50.

We compare our decoupled-search SPIN implementation to the standard SPIN model checker in version 6.4.7 with default settings, providing no additional command line options (SPIN), and to a configuration disabling statement merging (-M) and partial-order reduction (-POR). All configurations exhaust the entire state space, using the verifier options *-A* and *-E*. Restricting ourselves to safety properties, we removed any *never claims* from the models. We use a runtime limit of 60 min and allow for a maximum of 32 GiB of memory. The evaluation was performed on a cluster of Intel E5-2660 machines running at 2.20 GHz.

Figure 18.1 shows the results, scaling each case study until all configurations run out of memory (indicated by a "-"). Memory is always the bottleneck, no configuration runs out of time. Decoupled search works very well in Peterson, significantly reducing memory consumption and runtime. The number of states is reduced by more than two orders of magnitude in the large instances. From the strong gains of SPIN over its -M -POR variant, we conclude that Peterson in general benefits partial-order reduction techniques. To a lesser extent, decoupled search also has advantages in Elevator and X.509. In Elevator, although the number of states is again reduced by up to two orders of magnitude, the reduction in runtime and memory is less pronounced than in the Peterson model. Comparing the SPIN variants, we also see a somewhat smaller advantage of the POR and statement merging techniques.

		SPIN -M -POR				SPIN				Decoupled Search			
Model		Time	Mem	#States	Depth	Time	Mem	#States	Depth	Time	Mem	#States	Depth
	2	0	0.13	216	145	0	0.13	56	42	0	0.13	16	12
Peterson	3	0.04	0.13	33434	6924	**0**	0.13	2999	615	**0**	0.13	274	120
	4	19	1.14	8886434	1703147	0.53	0.21	533083	165342	**0.1**	**0.13**	6698	1615
	5	-	-	-	-	124	10.08	76620358	25309679	**4.16**	**0.27**	153548	27392
	6	-	-	-	-	-	-	-	-	**157**	**4.79**	3503908	473228
Elevator	3	0.23	0.14	99057	4609	0.12	**0.13**	78284	4950	**0.06**	**0.13**	7081	590
	4	3.15	0.19	685169	29487	1.49	0.17	498676	30239	**0.42**	**0.16**	37095	1643
	5	15.5	0.44	3620470	28638	5.02	0.33	2354211	27634	**2.38**	**0.26**	115077	1630
	6	95.7	1.86	18813600	30818	26.7	1.13	10868993	29712	**7.35**	**0.65**	359163	1728
	7	676	10.31	97574250	32998	153	5.56	49636481	31790	**25.5**	**2.08**	1119285	1826
	8	-	-	-	-	782	26.62	224704000	33868	**95.5**	**7.51**	3483243	1924
	9	-	-	-	-	-	-	-	-	**360**	**27**	10825893	2022
X.509		1.58	0.18	403311	91	**0**	**0.13**	3054	57	**0**	**0.13**	1090	35
	5	0.1	0.13	25600	3963	**0.02**	0.13	7731	2001	0.03	0.13	3245	324
Client-	6	0.72	0.15	141312	16235	**0.08**	**0.13**	32296	6830	0.15	0.14	13128	755
Server-	7	4.84	0.21	745472	63236	**0.42**	**0.15**	143741	27442	0.74	0.19	51037	1607
EmptyC	8	31.3	0.59	3801088	263835	**2.05**	**0.24**	507967	104759	3.3	0.42	192464	3297
	9	199	2.83	18874368	1062399	**8.53**	**0.44**	2206702	370926	14.6	1.36	708597	6274
	10	1160	12.38	91750400	4252067	**35**	**1.66**	8140911	1277049	63.2	5.34	2558800	13414
	11	-	-	-	-	**149**	**4.45**	29856762	4335070	256	21.11	9093557	28150
	12	-	-	-	-	**609**	**21.06**	109424300	14937082	-	-	-	-
	4	0.13	0.13	39936	7497	**0.01**	0.13	4037	1021	**0.01**	0.13	760	142
Client-	5	2.69	0.18	450560	65354	0.05	**0.13**	23614	6209	**0.04**	**0.13**	3245	324
Server-	6	30.2	0.73	4816896	518833	0.33	**0.15**	132210	32645	**0.21**	**0.15**	13128	755
NonEmptyC	7	416	7.6	49545216	4256969	2.12	0.27	708019	172048	**1.04**	**0.21**	51037	1607
	8	-	-	-	-	14	0.74	3813278	882008	**4.86**	**0.53**	192464	3297
	9	-	-	-	-	83.8	3.79	19384754	4254923	**20.4**	**1.87**	708597	6274
	10	-	-	-	-	480	22.14	95568530	19967819	**87.1**	**7.57**	2558800	13414
	11	-	-	-	-	-	-	-	-	**369**	**30.39**	9093557	28150
Transport-	4	0.22	0.13	112735	329	**0**	0.13	31018	311	**0**	0.13	18	8
Planning	5	3.85	0.19	1240092	978	0.36	0.14	237249	892	**0**	**0.13**	21	9
	6	47.8	0.95	13641019	2923	3.14	0.24	1815310	2698	**0**	**0.13**	24	10
	7	728	12.8	150051220	8756	29	1.07	13954478	6404	**0**	**0.13**	27	11
	8	-	-	-	-	269	8.03	107967020	19740	**0**	**0.13**	30	12
	50	-	-	-	-	-	-	-	-	**0.01**	**0.13**	156	54

Figure 18.1: Performance of SPIN with default options (SPIN), disabling statement merging (-M) and partial-order reduction (-POR), and with decoupled search. We show runtime (in seconds) and memory consumption (in GiB), as well as the number of stored states (#States), and the maximum search depth (Depth) reported by SPIN. Best runtime/memory is highlighted in **bold face**.

In Client-Server, as expected, decoupled search is more beneficial with more internal transitions in the clients. It performs worse than SPIN on the EmptyC variant, in spite exploring more than an order of magnitude less states. In both variants, the default SPIN configuration has tremendous advantages over the restricted configuration. Remarkably, the two model variants result in the exact same number of states for decoupled search. This nicely illustrates that internal transitions that do not result in different global behaviour can be completely compressed into a decoupled state. There is, however, an overhead in runtime and memory of the larger local state spaces. In the logistics case study adopted from planning, decoupled search excels. This is not a relevant observation in model checking per se, but points to the advantage decoupled state-space search may in principle have over previous search methods in SPIN.

In general, the number of decoupled states is consistently smaller than the number of states in SPIN. Where the reduction is relatively small, it is outweighed by the overhead of handling decoupled states. Regarding the search depth, keep in mind that the maximum depth of decoupled search is that of the *global transitions* only. The depth bound can, thus, in general be kept significantly smaller for decoupled search, leading to a reduced memory consumption for the search stack.

Chapter 19

Decoupled Search for Liveness Checking

In this chapter, we show how decoupled search can be applied to the verification of liveness properties for composed Büchi automata. We adapt, and show correct, a standard nested depth-first search algorithm for detecting lassos, i. e., infinite accepting runs.

Nested depth-first search (NDFS), like all state-space search methods, suffers from the state explosion problem. Various methods, such as partial-order reduction [Valmari, 1992; McMillan, 1992; Godefroid, 1996; Esparza et al., 2002; Rodríguez and Schwoon, 2013], symbolic representations [Bryant, 1986; McMillan, 1993], symmetry reduction [Emerson and Sistla, 1996; Ip and Dill, 1996], or Petri-net unfolding [Esparza and Heljanko, 2000, 2001] have been proposed to alleviate the state explosion problem. Here, we add decoupled state-space search as a new method for model checking liveness properties, complementary to the existing approaches.

In Chapter 19.1, we first discuss some issues that would arise in a naïve attempt to (incorrectly) adapt NDFS, and describe the (correct) adapted NDFS algorithm. To guarantee completeness, we introduce a novel decoupled-state *splitting* mechanism, that keeps track of the root state of the invocation of each call to the inner DFS. Chapter 19.2 provides the correctness proof of our decoupled NDFS variant.

In Chapter 19.3 we show our experimental evaluation, whose code and models are publicly available [Gnad et al., 2021b]. We evaluate our approach on two showcase examples similar to the dining philosophers problem, and a set of randomly generated models using a prototype implementation. We compare to established tools, namely the SPIN model checker [Holzmann, 2004], and Petri-net unfolding with Cunf [Rodríguez and Schwoon, 2013]. The results show that, like for safety properties, decoupled search can yield exponential advantages over state-of-the-art methods. In particular, its advantage grows with the degree to which components act independently of others, via internal transitions that do not affect other components.

19.1 NDFS for Decoupled Search

We now adapt NDFS to decoupled search. We start by discussing the deficiencies of a naïve adaptation. We will then introduce the key concepts in our fixed algorithm in Chapter 19.1.2, and present the algorithm itself in Chapter 19.1.3.

19.1.1 Issues with a Naïve Adaptation of NDFS

In a naïve adaptation of NDFS to decoupled search, the only thing that changes is the treatment of decoupled states, which represent *sets of composed states*, compared to single states in the standard variant. This leads to three mostly minor changes: (1) instead of duplicate checking we perform dominance pruning; (2) checking if a decoupled state is accepting boils down to checking if it contains an accepting member state; and (3) to see if a state $\vec{t}$ contained in a decoupled state $t^{\mathcal{D}}$ generated in **NestedDFS** is on the stack, we need to check if $t^{\mathcal{D}}$ has a non-empty intersection with a state on $Stack$.

As we will show next, it turns out that this naïve adaptation can *miss* cycles due to pruning. Moreover, revisiting a composed state in **NestedDFS** does actually not imply a cycle, because reaching $t^{\mathcal{D}}$ from $s^{\mathcal{D}}$ entails only that every member state of $t^{\mathcal{D}}$ can be reached from *at least one* member state of $s^{\mathcal{D}}$, not from all of them. Concretely, say we started **NestedDFS** from $s^{\mathcal{D}}_A$, and reached $t^{\mathcal{D}}$ with $\vec{t} \in t^{\mathcal{D}} \cap r^{\mathcal{D}}$ for $r^{\mathcal{D}}$ on $Stack$. Then we know that $\vec{t}$ can be reached from at least one $\vec{s} \in s^{\mathcal{D}}_A$; and we know that $\vec{s}$ can be reached from at least one $\vec{t}' \in r^{\mathcal{D}}$. It is not necessarily the case that $\vec{t} = \vec{t}'$. The critical point is that pruning does not take into account *from where* states are reachable.

Consider the example in Figure 19.1. The left part of the figure shows the local state space of component NBA $\mathcal{A}^1_B$. For simplicity, we only show a single component, which is sufficient to illustrate the issue. Here, $\mathcal{A}^1_B$ is defined as follows: $S^1 = \{1, 2, 3, 4\}$, $L^1_G = \{l^1_G, l^2_G\}$, $L^1_I = \{l^1_I, l^2_I, l^3_I\}$, $A^1(s) = \top$ iff $s \in \{2, 4\}$, and $s^1_0 = 1$. The transitions are as shown in the left of the figure. The decoupled search space generated using NDFS is depicted in the right of the figure. Pruned states are crossed out.

NestedDFS is launched (indicated by the wavy arrow) on the accepting initial state $s^{\mathcal{D}}_0$. Before explaining the main issue, we remark that, to ensure that a cycle through an *accepting* member state of $s^{\mathcal{D}}_0$ is found, not a cycle through a non-accepting one, we need to restrict the set of reached local states to those that are accepting, and the states internally reachable from those via iclose(). Thus, **NestedDFS** starts in what we call the acceptance-restriction $s^{\mathcal{D}}_{0,A}$ of $s^{\mathcal{D}}_0$, where $s^{\mathcal{D}}_{0,A}[\mathcal{A}^1_B] = \{2, 4\}$. Now, the issue results from the fact that $s^{\mathcal{D}}_{0,A}$ contains two accepting member states, only one of which, namely state 2, is on a cycle. Assuming that the decoupled states are generated in order of increasing subscripts, so $s^{\mathcal{D}}_1$ before $s^{\mathcal{D}}_2$ and so on, state 2 is first reached in **NestedDFS** as a member state of $s^{\mathcal{D}}_{2,A}$, but via the transition labelled with l^2_G from state 3, so the cycle cannot be closed. When generating the l^1_G successor $s^{\mathcal{D}}_{4,A}$ of $s^{\mathcal{D}}_{0,A}$, its only member state 3 has already been reached in $s^{\mathcal{D}}_{1,A}$, so $s^{\mathcal{D}}_{4,A}$ is pruned and the cycle of state 2 via

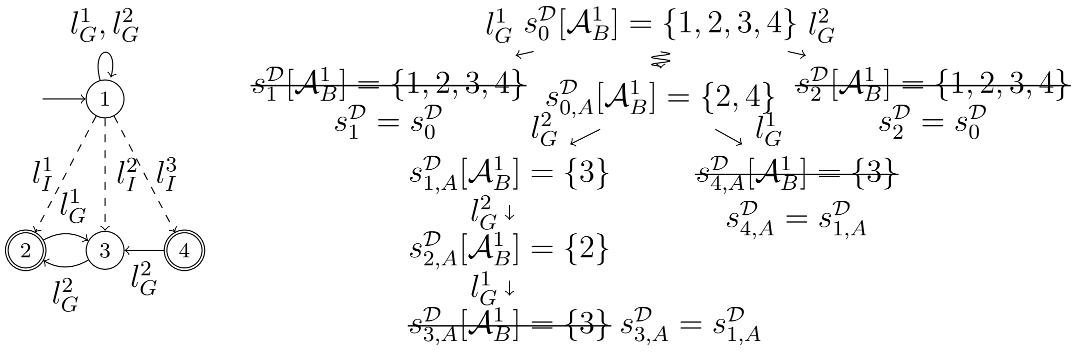

l_G^1, l_G^2
1
2
3
4
l_I^1
l_I^2
l_I^3
l_G^1
l_G^2
l_G^2
l_G^1 $s_0^{\mathcal{D}}[\mathcal{A}_B^1] = \{1,2,3,4\}$ l_G^2
~~$s_1^{\mathcal{D}}[\mathcal{A}_B^1] = \{1,2,3,4\}$~~ $s_1^{\mathcal{D}} = s_0^{\mathcal{D}}$
$s_{0,A}^{\mathcal{D}}[\mathcal{A}_B^1] = \{2,4\}$
~~$s_2^{\mathcal{D}}[\mathcal{A}_B^1] = \{1,2,3,4\}$~~ $s_2^{\mathcal{D}} = s_0^{\mathcal{D}}$
l_G^2
l_G^1
$s_{1,A}^{\mathcal{D}}[\mathcal{A}_B^1] = \{3\}$
~~$s_{4,A}^{\mathcal{D}}[\mathcal{A}_B^1] = \{3\}$~~ $s_{4,A}^{\mathcal{D}} = s_{1,A}^{\mathcal{D}}$
l_G^2
$s_{2,A}^{\mathcal{D}}[\mathcal{A}_B^1] = \{2\}$
l_G^1
~~$s_{3,A}^{\mathcal{D}}[\mathcal{A}_B^1] = \{3\}$~~ $s_{3,A}^{\mathcal{D}} = s_{1,A}^{\mathcal{D}}$

Figure 19.1: Counterexample showing that a naïve adaptation of the NDFS algorithm is incomplete. The (only) component NBA $\mathcal{A}_B^1$ is depicted on the left. The search tree on the right shows the entire reachable decoupled state space, where pruned states are crossed out; the wavy arrow depicts the invocation of **NestedDFS** on the acceptance restriction $s_{0,A}^{\mathcal{D}}$ of $s_0^{\mathcal{D}}$.

l_G^1, l_G^2 is missed. In the next section we show how to fix this, through an extended state representation that keeps track of reachability from a set of reference states.

Another minor issue are lassos $\vec{\rho}_p(\vec{\rho}_c)^\omega$ whose cycle $\vec{\rho}_c$ is induced by internal labels only. These will not be detected, because **NestedDFS** only considers traces via global labels. We fix this by checking for L_I-cycles in every accepting decoupled state generated during **DFS**, to see if there exists a component that can reach such a state.

19.1.2 Reference-State Splits

The problem underlying the issue described in the previous section is that pruning is done regardless of the accepting states in the root node of **NestedDFS**. We now introduce an operation on decoupled states splitting them with respect to the set of reached local accepting states for each component. In our algorithm, this will serve to distinguish the different accepting states, and thus force dominance pruning to distinguish reachability from these. Formally, we define the restriction to accepting local states as a new transition with a global label l_G^A that is a self-loop for all accepting states:

Definition 58 (Acceptance-Split Transition)**.** *Let* $\langle S^{\mathcal{D}}, \rightarrow_{\mathcal{D}}, L, s_0^{\mathcal{D}}, A^{\mathcal{D}}\rangle$ *be the decoupled composition of* $\mathcal{A}_B^1, \ldots, \mathcal{A}_B^n$*. Let* $s^{\mathcal{D}}$ *be an accepting decoupled state, and for* $1 \leq i \leq n$ *let* $\langle s_1^i, \ldots, s_{c_i}^i\rangle \subseteq s^{\mathcal{D}}[\mathcal{A}_B^i]$ *be the list of reached accepting states of* $\mathcal{A}_B^i$*, where for all* $1 \leq j \leq c_i : A^i(s_j^i) = \top$*. Then the* acceptance-split transition l_G^A *in* $s^{\mathcal{D}}$ *is defined as follows:*

$$\frac{A^{\mathcal{D}}(s^{\mathcal{D}}) = \top \quad \forall i \in \{1, \ldots n\}, j \in \{1, \ldots, c_i\} : s^i_j \in s^{\mathcal{D}}[\mathcal{A}^i_B] \wedge A^i(s^i_j) = \top}{s^{\mathcal{D}} \xrightarrow{l^A_G}_{\mathcal{D}} \langle\langle \mathit{iclose}(s^1_1), \ldots, \mathit{iclose}(s^1_{c_1})\rangle, \ldots, \langle \mathit{iclose}(s^n_1), \ldots, \mathit{iclose}(s^n_{c_n})\rangle\rangle}$$

The outcome state $s^{\mathcal{D}}_A$ of an acceptance-split transition is a split decoupled state. *The set of* reference states *of $s^{\mathcal{D}}_A$ is $R(s^{\mathcal{D}}_A) := \{s \mid \exists \mathcal{A}^i_B : s \in s^{\mathcal{D}}[\mathcal{A}^i_B] \wedge A^i(s) = \top\}$.*

In words, the operation splits up the single set of reached component states $s^{\mathcal{D}}[\mathcal{A}^i_B]$ of $\mathcal{A}^i_B$ into a list of state sets, where each such set $s^{\mathcal{D}}_A[\mathcal{A}^i_B]_s$ contains the states that can be reached via internal transitions from the respective accepting state $s \in s^{\mathcal{D}}[\mathcal{A}^i_B]$.

Our search algorithm will use the acceptance-split transition to generate the root node $s^{\mathcal{D}}_A$ of **NestedDFS** from an accepting state $s^{\mathcal{D}}$ backtracked from in **DFS**. Hence **NestedDFS** will search in the space of split decoupled states. The transitions over these behind an $s^{\mathcal{D}}_A$ are defined as follows:

Definition 59 (Split Transitions). *Let $\langle S^{\mathcal{D}}, \rightarrow_{\mathcal{D}}, L, s^{\mathcal{D}}_0, A^{\mathcal{D}}\rangle$ be the decoupled composition of $\mathcal{A}^1_B, \ldots, \mathcal{A}^n_B$. Let $s^{\mathcal{D}}$ and $t^{\mathcal{D}}$ be decoupled states, with a transition $s^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} t^{\mathcal{D}}$. Let $\langle s^i_1, \ldots, s^i_{c_i}\rangle \subseteq S^i$ be reference states for $\mathcal{A}^i_B$. Then the* split transition $s^{\mathcal{D}}_R \xrightarrow{l_G}_{\mathcal{D}} t^{\mathcal{D}}_R$ *is defined such that for every $\mathcal{A}^i_B$ and every $1 \leq j \leq c_i$ we have:*

$$t^{\mathcal{D}}_R[\mathcal{A}^i_B]_{s^i_j} = \begin{cases} \mathit{iclose}(\{s' \in t^{\mathcal{D}}[\mathcal{A}^i_B] \mid \exists s \in s^{\mathcal{D}}_R[\mathcal{A}^i_B]_{s^i_j} : s \xrightarrow{l_G}{}^i s'\}) & l_G \in L^i_G \\ s^{\mathcal{D}}_R[\mathcal{A}^i_B]_{s^i_j} & l_G \notin L^i_G \end{cases}$$

The list of reference states for an $\mathcal{A}^i_B$ does not change along a trace of split transitions. Let $s^{\mathcal{D}}_A$ be a decoupled state generated by an acceptance-split transition $s^{\mathcal{D}} \xrightarrow{l^A_G}_{\mathcal{D}} s^{\mathcal{D}}_A$, then for all successor states $t^{\mathcal{D}}$ of $s^{\mathcal{D}}_A$, the set of reference states is $R(t^{\mathcal{D}}) = R(s^{\mathcal{D}}_A)$.

We extend set operations to the split representation as follows. A split decoupled state $s^{\mathcal{D}}_R$ *dominates* a split decoupled state $t^{\mathcal{D}}_R$, denoted $t^{\mathcal{D}}_R \subseteq_R s^{\mathcal{D}}_R$, if $R(t^{\mathcal{D}}_R) \subseteq R(s^{\mathcal{D}}_R)$ and for all components $\mathcal{A}^i_B$ and reference states $s \in R(t^{\mathcal{D}}_R) \cap S^i$ we have $t^{\mathcal{D}}_R[\mathcal{A}^i_B]_s \subseteq s^{\mathcal{D}}_R[\mathcal{A}^i_B]_s$. In contrast, state membership is defined in a global manner, across reference states. Namely, the set of local states of an $\mathcal{A}^i_B$ reached in a split decoupled state $s^{\mathcal{D}}_R$ is defined as $s^{\mathcal{D}}_R[\mathcal{A}^i_B] := \bigcup_{s \in R(s^{\mathcal{D}}_R) \cap S^i} s^{\mathcal{D}}_R[\mathcal{A}^i_B]_s$. Composed state membership is defined relative to these $s^{\mathcal{D}}_R[\mathcal{A}^i_B]$ as before.

An important property of the splitting is that it preserves reachability of member states. Concretely, for a split-transition $s^{\mathcal{D}}_R \xrightarrow{l_G}_{\mathcal{D}} t^{\mathcal{D}}_R$ induced by a transition $s^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} t^{\mathcal{D}}$ for all $\mathcal{A}^i_B$ it holds that if $s^{\mathcal{D}}_R[\mathcal{A}^i_B] = s^{\mathcal{D}}[\mathcal{A}^i_B]$, then $t^{\mathcal{D}}_R[\mathcal{A}^i_B] = t^{\mathcal{D}}[\mathcal{A}^i_B]$.

As a notation convention, we will always denote split states $s^{\mathcal{D}}_R$ by a subscript R, and the direct outcome of an acceptance-split transition by $s^{\mathcal{D}}_A$, with a subscript A.

$$\begin{tikzcd}
 & s^{\mathcal{D}}_{1,R}[\mathcal{A}^1_B] = \langle\{\}_2,\{3\}_4\rangle \arrow[r, "l^2_G"] & s^{\mathcal{D}}_{2,R}[\mathcal{A}^1_B] = \langle\{\}_2,\{2\}_4\rangle \arrow[r, "l^1_G"] & \cancel{s^{\mathcal{D}}_{3,R}[\mathcal{A}^1_B] = \langle\{\}_2,\{3\}_4\rangle} \\
s^{\mathcal{D}}_{1,A}[\mathcal{A}^1_B] = \langle\{2\}_2,\{4\}_4\rangle \arrow[ur, "l^2_G"] \arrow[dr, "l^1_G"'] & & & s^{\mathcal{D}}_{3,R} \subseteq_R s^{\mathcal{D}}_{1,R} \\
 & s^{\mathcal{D}}_{4,R}[\mathcal{A}^1_B] = \langle\{3\}_2,\{\}_4\rangle \arrow[r, "l^2_G"] & s^{\mathcal{D}}_{5,R}[\mathcal{A}^1_B] = \langle\{2\}_2,\{\}_4\rangle &
\end{tikzcd}$$

Figure 19.2: With acceptance-splitting, **NestedDFS** invoked on the l^A_G-successor $s^{\mathcal{D}}_{1,A}$ of $s^{\mathcal{D}}_0$ of the example in Figure 19.1 correctly detects the cycle of state 2 induced by the trace l^1_G, l^2_G.

Considering our example again, Figure 19.2 illustrates how, on split decoupled states, the cycle $2 \xrightarrow{l^1_G} 3 \xrightarrow{l^2_G} 2$ is not pruned. The state $s^{\mathcal{D}}_{3,R}$ is still pruned, as it contains only component states reached from state 4. In $s^{\mathcal{D}}_{4,R}$ and $s^{\mathcal{D}}_{5,R}$, the decoupled state keeps track of the traces from the origin state 2, so none of the two is pruned, since they are not dominated by any state $s^{\mathcal{D}}_{i,R}$ (the root node $s^{\mathcal{D}}_{1,A}$ of **NestedDFS** is not yet visited).

As indicated before, in our emptiness checking algorithm we will use split decoupled states only within **NestedDFS**. The seed state $s^{\mathcal{D}}_A$ of **NestedDFS** will always be the l^A_G-successor of an accepting state $s^{\mathcal{D}}$ backtracked from in **DFS**. Every member state of $s^{\mathcal{D}}_A$ is accepting, or can be reached with L_I-transitions from an accepting state.

19.1.3 Putting Things Together: Decoupled NDFS

We are now ready to describe our adaptation of the standard NDFS algorithm to decoupled compositions. The pseudo-code is shown in Figure 19.3. The differences w.r.t the standard algorithm (Figure 16.4) are highlighted in blue. The basic structure of the algorithm is preserved. It starts by putting the decoupled initial state $s^{\mathcal{D}}_0$ onto the Stack in **CheckEmptiness**, and launches the main **DFS** from it.

In **DFS**, the control flow does not change, decoupled states are generated in depth-first order by recursion, updating the stack accordingly. There are however three differences to the standard variant:

1. Before generating the successors, we call **CheckLocalAccept** on each accepting decoupled state $s^{\mathcal{D}}$. This detects cycles resulting from L_I-transitions, i. e., cycles that occur "within" a decoupled state. To this end, we check whether there exists a component $\mathcal{A}^i_B$ for which an accepting local state s^i_a is reached that can reach itself using only internal labels L^i_I (the set of such local states can be precomputed, so that the check becomes a lookup operation). We can then construct an accepting run for the composed system by appending the L^i_I-cycle to the sequence of states

```
CheckEmptiness($\mathcal{A}^1_B \parallel_{\mathcal{D}} \dots \parallel_{\mathcal{D}} \mathcal{A}^n_B$):
  Stack ← $\langle s^{\mathcal{D}}_0 \rangle$
  $V \leftarrow \emptyset$
  $V' \leftarrow \emptyset$
  DFS($s^{\mathcal{D}}_0$)
  return empty

NestedDFS($s^{\mathcal{D}}_R$):
  foreach $t^{\mathcal{D}}_R$ s.t. $s^{\mathcal{D}}_R \rightarrow_{\mathcal{D}} t^{\mathcal{D}}_R$ do
    if $\exists r^{\mathcal{D}}_R \in V'$ s.t. $t^{\mathcal{D}}_R \subseteq_R r^{\mathcal{D}}_R$ then
      continue
    if $\forall \mathcal{A}^i_B : \exists s : s \in t^{\mathcal{D}}_R[\mathcal{A}^i_B]_s$ then
      return cycle
    if $\exists r^{\mathcal{D}} \in Stack$ s.t. $r^{\mathcal{D}} \subseteq t^{\mathcal{D}}_R$ then
      return cycle
    $V' \leftarrow V' \cup \{t^{\mathcal{D}}_R\}$
    NestedDFS($t^{\mathcal{D}}_R$)
  end

DFS($s^{\mathcal{D}}$):
  $V \leftarrow V \cup \{s^{\mathcal{D}}\}$
  if $A^{\mathcal{D}}(s^{\mathcal{D}})$ then
    CheckLocalAccept($s^{\mathcal{D}}$)
  foreach $t^{\mathcal{D}}$ s.t. $s^{\mathcal{D}} \rightarrow_{\mathcal{D}} t^{\mathcal{D}}$ do
    if $\exists r^{\mathcal{D}} \in V$ s.t. $t^{\mathcal{D}} \subseteq r^{\mathcal{D}}$ then
      continue
    push(Stack, $t^{\mathcal{D}}$)
    DFS($t^{\mathcal{D}}$)
    pop(Stack)
  end
  if $A^{\mathcal{D}}(s^{\mathcal{D}})$ then
    Let $s^{\mathcal{D}}_A$ s.t. $s^{\mathcal{D}} \xrightarrow{l^A_G}_{\mathcal{D}} s^{\mathcal{D}}_A$.
    NestedDFS($s^{\mathcal{D}}_A$)
    $V' \leftarrow V' \cup \{s^{\mathcal{D}}_A\}$
  end

CheckLocalAccept($s^{\mathcal{D}}$):
  if $\exists \mathcal{A}^i_B, s \in s^{\mathcal{D}}[\mathcal{A}^i_B] : A^i(s) \wedge s \xrightarrow{l_I \in L^i_I}{}^{+} s$
   then return cycle
```

Figure 19.3: Adaptation of NDFS for lasso search in decoupled compositions of NBA.

that reaches s^i_a in $s^{\mathcal{D}}$ for $\mathcal{A}^i_B$. Note that it suffices if a single component moves and all other components remain in a reached accepting state.

2. Instead of doing duplicate checking, the algorithm performs dominance pruning, pruning a new decoupled state $t^{\mathcal{D}}$ if all its member states have been reached in an already visited decoupled state $r^{\mathcal{D}}$.

3. As discussed in Section 19.1.2, when we launch **NestedDFS** at a decoupled state $s^{\mathcal{D}}$, we do so on the acceptance-split l^A_G-successor $s^{\mathcal{D}}_A$ of $s^{\mathcal{D}}$.

NestedDFS now starts in the acceptance-split $s^{\mathcal{D}}_A$, and traverses split transitions as per Definition 59. On generation of a new state $t^{\mathcal{D}}_R$, we perform dominance pruning against the decoupled states visited during all prior calls to **NestedDFS**. If in an $t^{\mathcal{D}}_R$ for every component $\mathcal{A}^i_B$ there exists a reference state $s \in S^i$ that is reachable from itself, so $s \in t^{\mathcal{D}}[\mathcal{A}^i_B]_s$, then we can construct a cycle. As we will show in Theorem 41, this test is guaranteed to find all cycles that start from an accepting state $\vec{s}_A \in s^{\mathcal{D}}_A$.

Note that we cannot check for a non-empty intersection with states $r^{\mathcal{D}}$ on $Stack$, since these are not split relative to the reference states of $s^{\mathcal{D}}_A$. Thus, since we do not know from which local state in $r^{\mathcal{D}}$ the state in the intersection was reached, such a non-empty intersection would *not* imply a cycle. What we can do, however, is check for dominance instead, as an algorithm optimization inspired by [Holzmann et al., 1996]. The pseudo-code in Figure 19.3 does so by checking whether $t_R^{\mathcal{D}} \supseteq r^{\mathcal{D}}$, where the $\supseteq$ relation between a split vs. non-split state is simply evaluated based on the overall sets $t_R^{\mathcal{D}}[\mathcal{A}_B^i]$ vs. $r^{\mathcal{D}}[\mathcal{A}_B^i]$ of reached components states. If this domination relation holds true, then the reachability issue mentioned in the previous section is resolved because *all* $\vec{t} \in r^{\mathcal{D}}$ are then reachable from $s_A^{\mathcal{D}}$—including those $\vec{t}$ from which an accepting state $\vec{s} \in s_A^{\mathcal{D}}$ is reachable. Lemma 15 in the next section will spell out this argument as part of our correctness proof.

Observe that splitting a decoupled state incurs an increase in the size of the state representation, as the same local state may be reached from several reference states. More importantly, as dominance pruning is weaker on split states (which after all is the purpose of the split operation) the size of the search space may increase. As shown by the example in Figure 19.1, though, there is no easy way around the splitting, since the algorithm has to be able to know *from which* component state the successors states are reached. Assuming a component has M accepting states, then in the worst case all local successor states that are shared between these accepting states can be visited M times across all **NestedDFS** invocations. Unless some of the decoupled states revisiting the same member state are pruned by dominance pruning, it can actually happen that the revisits multiply across the components, so the size of the decoupled state space in **NestedDFS** can potentially be exponentially larger than the standard state space. As we shall see in our experimental evaluation, typically such blow-ups do not seem to occur.

In case we want to construct a lasso, we need to store a pointer to the predecessor of each decoupled state and the label of the generating transition. With this, we can, for each component $\mathcal{A}_B^i$ separately, reconstruct a trace $\vec{\pi}$ of a state $\vec{t} \in t^{\mathcal{D}}$ reached from a state $\vec{s} \in s^{\mathcal{D}}$ where $\pi^G(s^{\mathcal{D}}, t^{\mathcal{D}}) = \pi^G(\vec{\pi})$. Here, for a decoupled state $t^{\mathcal{D}}$ that was reached from another decoupled state $s^{\mathcal{D}}$, by $\pi^G(s^{\mathcal{D}}, t^{\mathcal{D}})$ we denote the global trace via which $t^{\mathcal{D}}$ was reached from $s^{\mathcal{D}}$. Reconstructing $\vec{\pi}$ can be done in time polynomial in the size of the component and linear in the length of $\pi^G(s^{\mathcal{D}}, t^{\mathcal{D}})$. Since the traces for all components are synchronized via $\pi^G(\vec{\pi})$, we add the required internal labels for each component in between every pair of global labels. We remark that, to decide if a lasso exists, we do not need to store any predecessor or generating label pointers.

We next show on an example how our algorithm works.

Example 12. *The model has two component NBAs $\mathcal{A}_B^1, \mathcal{A}_B^2$ illustrated in Figure 19.4. It is a variant of an example from Laarman et al. [2013]. We remark that all global transitions $l_G^1, \dots l_G^7$ induce a self loop in the only state 1 of $\mathcal{A}_B^2$.*

***CheckEmptiness** starts by putting $s_0^{\mathcal{D}}$ onto Stack and enters **DFS**$(s_0^{\mathcal{D}})$. Let $s_1^{\mathcal{D}} = \langle\{B, D\}, \{1\}\rangle$, $s_2^{\mathcal{D}} = \langle\{E, F\}, \{1\}\rangle$, and $s_3^{\mathcal{D}} = \langle\{D\}, \{1\}\rangle$ be the successors generated along the trace l_G^1, l_G^2, l_G^3 in **DFS**. Since $s_3^{\mathcal{D}} \subseteq s_1^{\mathcal{D}} \in V$, $s_3^{\mathcal{D}}$ is pruned and the search*

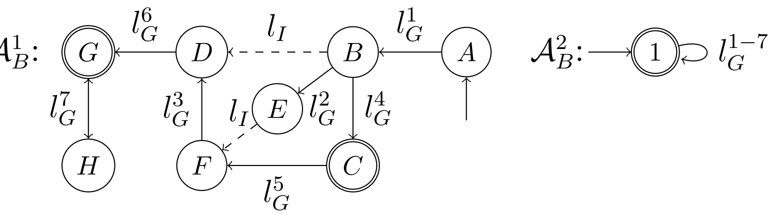


Figure 19.4: Illustration of the component NBAs used in Example 12.

*backtracks to $s_1^{\mathcal{D}}$. Say **DFS** selects the transition via l_G^4 next, generating the state $s_4^{\mathcal{D}} = \langle\{C\},\{1\}\rangle$ and its l_G^5-successor $s_5^{\mathcal{D}} = \langle\{F\},\{1\}\rangle$. Then $s_5^{\mathcal{D}}$ is pruned because it is dominated by $s_2^{\mathcal{D}} \in V$, and the search backtracks from $s_4^{\mathcal{D}}$, which is accepting.*

*Thus, **NestedDFS**$(s_{5,A}^{\mathcal{D}})$ is invoked, where $s_{5,A}^{\mathcal{D}} = \langle\langle\{C\}_C\rangle,\langle\{1\}_1\rangle\rangle$, because C and 1 are accepting local states that become the reference states of $s_{5,A}^{\mathcal{D}}$. **NestedDFS** will follow the trace $l_G^5, l_G^3, l_G^6, l_G^7, l_G^7$, which among others generates the state $s_{6,R}^{\mathcal{D}} = \langle\langle\{G\}_C\rangle,\langle\{1\}_1\rangle\rangle$ by l_G^6, and ends in $s_{7,R}^{\mathcal{D}} = \langle\langle\{G\}_C\rangle,\langle\{1\}_1\rangle\rangle$. The latter is pruned, because it is dominated by $s_{6,R}^{\mathcal{D}}$, which is contained in V'. No cycle is reported. This is correct, because the only member state $(C,1)$ of $s_{5,A}^{\mathcal{D}}$ does not occur on a cycle.*

***DFS** then backtracks to $s_1^{\mathcal{D}} = \langle\{B,D\},\{1\}\rangle$ and generates its remaining successor $s_8^{\mathcal{D}} = \langle\{G\},\{1\}\rangle$ via l_G^6. **DFS** further generates the l_G^7-successors of $s_8^{\mathcal{D}}$ and eventually backtracks from $s_8^{\mathcal{D}}$, invoking **NestedDFS**$(s_{8,A}^{\mathcal{D}})$, where $s_{8,A}^{\mathcal{D}} = \langle\langle\{G\}_G\rangle,\langle\{1\}_1\rangle\rangle$.*

After two transitions via l_G^7 the resulting state $s_{9,R}^{\mathcal{D}} = \langle\langle\{G\}_G\rangle,\langle\{1\}_1\rangle\rangle$ satisfies the condition that for all components $\mathcal{A}_B^i : s \in s_{9,R}^{\mathcal{D}}[\mathcal{A}_B^i]_s$, namely G and 1. Thus, a cycle is reported. It is induced by the trace $l_G^1, l_I, l_G^6, l_G^7, l_G^7$.

*Note that no decoupled state in the second **NestedDFS** is pruned, since none of them is dominated by a state in V' of the first **NestedDFS** invocation. In particular, $s_{8,A}^{\mathcal{D}} = \langle\{G\}_G,\{1\}_1\rangle$ is not dominated by $s_{6,R}^{\mathcal{D}} = \langle\{G\}_C,\{1\}_1\rangle$, because the reference states differ—G and 1 for $s_{8,R}^{\mathcal{D}}$ and C and 1 for $s_{6,R}^{\mathcal{D}}$.*

19.2 Decoupled NDFS Correctness

We now show the correctness of our approach. In Lemmas 15, 16, 17, we show that if our algorithm reports a cycle, then there exists an accepting run for $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$. In Theorem 41, we then show that decoupled NDFS does not miss an accepting run.

We first show that the optimization of checking dominance of states in **NestedDFS** against states on the stack is sound, i. e., that an accepting run exists if it fires.

Lemma 15. *Let $r^{\mathcal{D}}$ be a decoupled state on the current **DFS** $Stack$, and let $t_R^{\mathcal{D}}$ be a decoupled state generated by **NestedDFS**. If $t_R^{\mathcal{D}} \supseteq r^{\mathcal{D}}$, then there exists an accepting run for $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$.*

Proof. Let $s^{\mathcal{D}}$ be the accepting state that is backtracked from in **DFS**, i. e., the current **NestedDFS** was invoked on its l^A_G-successor $s^{\mathcal{D}}_A$.

From Theorem 40 we know that if $s^{\mathcal{D}}_2$ is reachable from $s^{\mathcal{D}}_1$, then for every state $\vec{s}_2 \in s^{\mathcal{D}}_2$ there exists a state $\vec{s}_1 \in s^{\mathcal{D}}_1$ such that $\vec{s}_1 \xrightarrow{\vec{\pi}} \vec{s}_2$, where $\pi^G(\vec{\pi}) = \pi^G(s^{\mathcal{D}}_1, s^{\mathcal{D}}_2)$.

This result also holds for decoupled states reached in **NestedDFS** from states in **DFS**. This is because the acceptance-split transition l^A_G only restricts the set of reached member states of $s^{\mathcal{D}}$ in $s^{\mathcal{D}}_A$, so in particular $s^{\mathcal{D}}_A \subseteq s^{\mathcal{D}}$. Furthermore, split transitions generating states behind $s^{\mathcal{D}}_A$ do not affect reachability of member states of these split-decoupled states compared to their non-split counterparts.

In particular, (1) for every state $\vec{s}_2 \in s^{\mathcal{D}}$ there exists a state $\vec{s}_1 \in s^{\mathcal{D}}_0$ that reaches $\vec{s}_2$ on a trace $\vec{\pi}$ where $\pi^G(\vec{\pi}) = \pi^G(s^{\mathcal{D}}_0, s^{\mathcal{D}})$, which, with $s^{\mathcal{D}}_A \subseteq s^{\mathcal{D}}$ also holds for all $\vec{s}_2 \in s^{\mathcal{D}}_A$; and (2) for every state $\vec{t} \in t^{\mathcal{D}}_R$ there exists an accepting state $\vec{s}_A \in s^{\mathcal{D}}_A$ that reaches $\vec{t}$ on a trace $\vec{\pi}$ where $\pi^G(\vec{\pi}) = \pi^G(s^{\mathcal{D}}_A, t^{\mathcal{D}}_R)$.

Since $r^{\mathcal{D}}$ is on *Stack*, it holds that every $\vec{s} \in s^{\mathcal{D}}_A$ is reachable from a $\vec{r} \in r^{\mathcal{D}}$, and, with $t^{\mathcal{D}}_R \supseteq r^{\mathcal{D}}$, that every $\vec{r} \in r^{\mathcal{D}}$ is reachable from an accepting state $\vec{s}_A \in s^{\mathcal{D}}_A$.

Let $pred(s^{\mathcal{D}}_1, s^{\mathcal{D}}_2, \vec{s}_2)$ be a function that, if $s^{\mathcal{D}}_2$ is reachable from $s^{\mathcal{D}}_1$ and $\vec{s}_2 \in s^{\mathcal{D}}_2$, outputs a state $\vec{s}_1 \in s^{\mathcal{D}}_1$ that reaches $\vec{s}_2$ via a trace $\vec{\pi}$ with $\pi^G(s^{\mathcal{D}}_1, s^{\mathcal{D}}_2) = \pi^G(\vec{\pi})$.

Let $\vec{s}_0$ be a state reached in both $t^{\mathcal{D}}_R$ and $r^{\mathcal{D}}$, and let $\vec{s}_1 = pred(r^{\mathcal{D}}, t^{\mathcal{D}}_R, \vec{s}_0)$ be its predecessor in $r^{\mathcal{D}}$. If $\vec{s}_1 = \vec{s}_0$, then we are done, because there exists a lasso $\vec{s}_0, \dots, \vec{s}_0, \dots, \vec{s}_A, \dots, \vec{s}_0, \dots, \vec{s}_A$, where $\vec{s}_A$ is an accepting state traversed in $s^{\mathcal{D}}_A$. Such an accepting state exists because all member states of a decoupled state in **NestedDFS** are reachable from an accepting state in $s^{\mathcal{D}}_A$.

If $\vec{s}_1 \neq \vec{s}_0$, then we iterate and set $\vec{s}_i = pred(r^{\mathcal{D}}, t^{\mathcal{D}}_R, \vec{s}_{i-1})$, where such $\vec{s}_i$ exist because $r^{\mathcal{D}} \subseteq t^{\mathcal{D}}_R$. Because there are only finitely many states in $r^{\mathcal{D}}$, eventually we get $\vec{s}_i = \vec{s}_j$ (where $j < i$) and there exists a lasso as follows:

First, there exists a cycle $\vec{s}_i, \dots, \vec{s}_{i-1}, \dots, \vec{s}_j = \vec{s}_i$, where between every pair of states $\vec{s}_k, \vec{s}_{k-1}$ an accepting state $\vec{s}_{k,A}$ in $s^{\mathcal{D}}_A$ is traversed, for the same reason as before. We can obviously shift and truncate the cycle to start right after and end in $\vec{s}_{i,A}$. The prefix of the lasso is $\vec{s}_0, \dots, \vec{s}_{i,A}$. □

Lemmas 16 and 17 show the soundness of our main termination criterion, and of **CheckLocalAccept**.

Lemma 16. *Let $t^{\mathcal{D}}_R$ be a split decoupled state generated in **NestedDFS**. If for every component $\mathcal{A}^i_B$ there exists a component state s^i such that $s^i \in t^{\mathcal{D}}[\mathcal{A}^i_B]_{s^i}$, then there exists an accepting run for $\mathcal{A}^1_B \parallel \dots \parallel \mathcal{A}^n_B$.*

Proof. Let $s^{\mathcal{D}}_A$ be the acceptance-split decoupled state from which **NestedDFS** was started. If for every component $\mathcal{A}^i_B$ such an s^i exists, then the state $\vec{s} = (s^1, \dots, s^n)$ is reachable in both $s^{\mathcal{D}}_A$ and $t^{\mathcal{D}}_R$. By the construction of the reached state sets $t^{\mathcal{D}}_R[\mathcal{A}^i_B]_{s^i}$, $\vec{s}$ is reachable from itself and is accepting. Hence, there exists a lasso $\vec{s}_0, \dots, \vec{s}, \dots, \vec{s}$. □

Lemma 17. *Let $t^{\mathcal{D}}$ be an accepting decoupled state generated in* ***DFS*** *such that a cycle is reported by* ***CheckLocalAccept***$(t^{\mathcal{D}})$*, then an accepting run for $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$ exists.*

Proof. By prerequisite, there exists an accepting member state $\vec{s}$ of $t^{\mathcal{D}}$. If **CheckLocalAccept**$(t^{\mathcal{D}})$ reports a cycle, then there exists a component $\mathcal{A}_B^i$, where an accepting state $s^i \in t^{\mathcal{D}}[\mathcal{A}_B^i]$ is reached that lies on an cycle induced by transitions labelled with L_I^i. Thus, we can set the local state of $\mathcal{A}_B^i$ in $\vec{s}$ to s^i, and the lasso looks as follows: $\vec{s}_0, \ldots, \vec{s}, \ldots, \vec{s}$, where on the cycle only $\mathcal{A}_B^i$ moves. □

We are now ready to prove the correctness of our decoupled NDFS algorithm.

Theorem 41. *Let $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$ be the composition of n NBA and let $\mathcal{A}_B^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}_B^n$ be its decoupled composition. Then* ***CheckEmptiness***$(\mathcal{A}_B^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}_B^n)$ *reports a cycle if and only if an accepting run for $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$ exists.*

Proof. If **CheckEmptiness** reports a cycle, then by Lemmas 15, 16, and 17, which cover exactly the cases where a cycle is reported, an accepting run for $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$ exists.

For the other direction, assume that $\vec{\rho}$ is an accepting run for $\mathcal{A}_B^1 \parallel \ldots \parallel \mathcal{A}_B^n$. Let $\vec{s}_a$, with $0 \leq a < k$, be the accepting state that starts the cycle of the lasso $\vec{\rho}_p = \vec{s}_0, \ldots, \vec{s}_a$, $\vec{\rho}_c = \vec{s}_{a+1}, \ldots, \vec{s}_k$, where $\vec{s}_a = \vec{s}_k$. Let $\vec{\pi} = l_1, \ldots, l_k$ be the trace on which $\vec{s}_k$ is reached, i. e., for all $1 \leq i < k : \langle \vec{s}_i, l_{i+1}, \vec{s}_{i+1} \rangle \in \rightarrow$.

By Theorem 40, there exists a decoupled state $s^{\mathcal{D}}$ reached in **DFS** that contains $\vec{s}_a$.

If $\vec{\pi}$ is such that for all $a < i \leq k : l_i \in L_I$, i. e., the cycle $\vec{\rho}_c$ is induced only by internal labels, we next proof that **CheckLocalAccept**$(s^{\mathcal{D}})$ reports a cycle: As $\vec{s}_a$ is accepting, $s^{\mathcal{D}}$ is accepting, too, so unless a cycle is reported before, eventually **CheckLocalAccept**$(s^{\mathcal{D}})$ is called. If $\vec{\rho}_c$ is induced by only internal labels, then, because there cannot be any component interaction via L_I-transitions, there must exist a component $\mathcal{A}_B^i$ for which the local state s^i in $\vec{s}_a$ reaches itself with only L_I^i-transitions. We can pick any such $\mathcal{A}_B^i$ and ignore transitions from $\vec{\rho}_c$ that are labelled by an element of $L_I \setminus L_I^i$, since these are not required for an accepting cycle. Consequently, **CheckLocalAccept**$(s^{\mathcal{D}})$ reports a cycle.

We next show that, if $\vec{\pi}$ contains a global label on the cycle, i. e., there exists an $i \in \{a+1, \ldots, k\}$ such that $l_i \in L_G$, then, unless a cycle is reported before, **NestedDFS**$(s_A^{\mathcal{D}})$ reports a cycle, where $s_A^{\mathcal{D}}$ is the l_G^A-successor of $s^{\mathcal{D}}$.

Assume for contradiction that this is not the case, i. e., no cycle has been reported before, and **NestedDFS**$(s_A^{\mathcal{D}})$ does not report a cycle. Let **NestedDFS**$(s_A^{\mathcal{D}})$ be the first call to **NestedDFS** that misses a cycle, although an $\vec{s}_a \in s_A^{\mathcal{D}}$ that is on a cycle exists.

If $\vec{s}_a$ is on a cycle, then by Theorem 40 there exists a decoupled state $t^{\mathcal{D}}$ reachable from $s_A^{\mathcal{D}}$ that also contains $\vec{s}_a$. The result of Theorem 40 holds in this case because, by the definition of split transitions, the splitting does not affect reachability of member states. So there exists $t_R^{\mathcal{D}}$ reachable from $s_A^{\mathcal{D}}$ that contains $\vec{s}_a$.

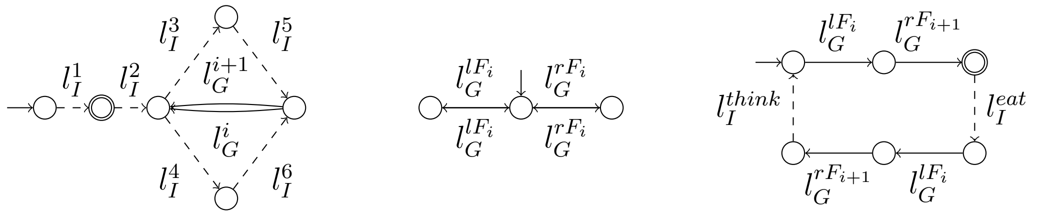


Figure 19.5: Illustrations of the ring model (left) and the fork (middle) and philosopher (right) NBAs of the philosophers model. Initial states are marked by an incoming arrow, accepting states by a double circle.

Denote by $\vec{\pi}_c = l_{a+1}, \dots l_k$ the cycle part of $\vec{\pi}$. Because $\vec{s}_a$ is an accepting member state of $s^{\mathcal{D}}$, all its component states s^i_A become reference states in $s^{\mathcal{D}}_A$. Therefore, assuming that $\pi^G(s^{\mathcal{D}}_A, t^{\mathcal{D}}_R) = \pi^G(\vec{\pi}_c)$, for all components we have $s^i_A \in t^{\mathcal{D}}_R[\mathcal{A}^i_B]_{s^i_A}$ and a cycle is reported. If this is not the case, then either (1) $s^{\mathcal{D}}$ was not reached in **DFS**, or (2) $t^{\mathcal{D}}_R$ was not reached in **NestedDFS**$(s^{\mathcal{D}}_A)$.

In case (1), there must exist a state $s^{\mathcal{D}}_P \supseteq s^{\mathcal{D}}$ that prunes $s^{\mathcal{D}}$. But then, $s^{\mathcal{D}}_P$ contains $\vec{s}_a$, too, and **NestedDFS** was called on its l^A_G-successor $s^{\mathcal{D}}_{P,A}$ and the cycle of $\vec{s}_a$ was missed before, in contradiction.

For (2), either (a) there exists a state $t^{\mathcal{D}}_{P,R} \supseteq_R t^{\mathcal{D}}_R$ that was reached in a prior invocation of **NestedDFS** on an accepting state $s^{\mathcal{D}}_{P,A}$, or (b) a state $t^{\mathcal{D}}_{P,R} \supseteq t^{\mathcal{D}}_R$ was reached in **NestedDFS**$(s^{\mathcal{D}}_A)$ before $t^{\mathcal{D}}_R$. In both cases, $t^{\mathcal{D}}_R$ is pruned and the cycle through $\vec{s}_a$ is missed. Case (a) can only happen if $s^{\mathcal{D}}_{P,A}$ contains $\vec{s}_a$, too, because the reference states of $s^{\mathcal{D}}_A$ need to be a subset of the ones of $s^{\mathcal{D}}_{P,A}$. But then, the cycle of $\vec{s}_a$ was missed before, in contradiction. For (b), if $t^{\mathcal{D}}_R \subseteq_R t^{\mathcal{D}}_{P,R}$, then for all $\mathcal{A}^i_B$ we have $s^i_A \in t^{\mathcal{D}}_{P,R}[\mathcal{A}^i_B]_{s^i_A}$, so the cycle would have been reported before, in contradiction.

The reachability argument in (1,2a,2b) applies recursively to all predecessors of $s^{\mathcal{D}}$ in **DFS**, and of $t^{\mathcal{D}}_R$ in **NestedDFS**$(s^{\mathcal{D}}_A)$, so, unless a cycle is reported before, eventually a state $s^{\mathcal{D}}$ is reached in **DFS** that contains $\vec{s}_a$, and a state $t^{\mathcal{D}}_R$ with $s^i_A \in t^{\mathcal{D}}_R[\mathcal{A}^i_B]_{s^i_A}$ in **NestedDFS**$(s^{\mathcal{D}}_A)$. □

19.3 Experimental Evaluation

We implemented a prototype of the decoupled NDFS algorithm from Figure 19.3. The input is specified in the Hanoi Omega-Automata format [Babiak et al., 2015], describing a set of NBAs synchronized via global labels as in Definition 56. Although, as mentioned before, it is possible to represent decoupled states symbolically (see Chapter 12), we maintain decoupled states as sets of explicit local states in our implementation. We compare our prototype to the SPIN model checker [Holzmann, 2004] (v6.5.1), and to the Cunf Petri-net unfolding tool [Rodríguez and Schwoon, 2013] (v1.6.1). We also

Dining Philosophers										Ring Topology									
	SPIN			Cunf			DecNDFS				SPIN			Cunf			DecNDFS		
$\#\mathcal{A}_B$	Time	#States	M	Time	#E	M	Time	#States	M	$\#\mathcal{A}_B$	Time	#States	M	Time	#E	M	Time	#S	M
2	0.0	18	129	0.0	23	**6**	0.0	13	8	5	0.01	12.2K	129	**0.0**	225	6	**0.0**	7	8
3	0.0	76	129	0.0	75	**6**	0.0	36	8	6	0.10	81.5K	133	**0.0**	342	6	**0.0**	8	8
4	0.0	348	129	0.0	162	**6**	0.0	97	8	7	0.95	560K	157	**0.0**	484	7	**0.0**	9	8
5	0.0	2000	129	0.0	293	**6**	0.0	272	8	8	8.35	3.7M	303	0.01	651	7	**0.0**	10	8
6	**0.0**	9416	131	0.01	482	**7**	0.01	783	8	9	73.6	24.6M	1367	0.01	843	**8**	**0.0**	11	**8**
7	0.2	45132	139	**0.01**	735	**8**	0.06	2290	**8**	10	-	-	-	0.01	1060	9	**0.0**	12	**8**
8	1.3	212K	175	**0.02**	1066	9	0.60	6761	**8**	15	-	-	-	0.03	2525	17	**0.0**	17	**8**
9	7.9	992K	333	**0.02**	1481	11	5.49	20.1K	**9**	20	-	-	-	0.10	4570	37	**0.0**	22	**8**
10	46.8	4.6M	993	**0.04**	1994	15	56.7	59.9K	**14**	25	-	-	-	0.22	7240	74	**0.01**	27	**8**
11	278.0	21.6M	3965	**0.04**	2386	**18**	558	179K	44	50	-	-	-	3.80	30K	917	**0.06**	52	**8**
12	-	-	-	**0.06**	2874	**23**	-	-	-	75	-	-	-	-	-	-	**0.26**	77	**8**

Figure 19.6: Statistics on the two scaling models, where $\#\mathcal{A}_B$ is the number of philosophers, resp. the number of NBAs, Time is runtime in seconds, #States (#S) and #E are the number of visited states, resp. generated events, and M is the memory usage in MiB. We highlight best runtime and memory performance in **bold face**.

experimented with the symbolic model checkers NuSMV and PRISM [Cimatti et al., 2002; Kwiatkowska et al., 2011], but both are significantly outperformed by the other methods. We conjecture that this is because both systems are not specifically designed for LTL checking on an asynchronous execution of processes. For SPIN, we translate each NBA to a process where NBA states are represented by state labels, internal transitions by `goto` statements, and global transitions by rendezvous channel operations. For the latter, SPIN only supports synchronization of two processes at a time, so we restrict the models to global transitions with exactly two components. We model acceptance for SPIN explicitly using a monitor process that gets into an accepting state if all processes are in a local accepting state. The translation for Cunf encodes NBA states as net places and transitions as net transitions into a single Petri net, ignoring the individual components. In our prototype and in SPIN, when a lasso is reported or the algorithm proved that no lasso exists within the cut-off limits, we say that the instance was *solved*. For Cunf, we attempt to construct a complete unfolding prefix. We consider an instance solved if the construction terminates, i. e., we do not actually check the liveness property. The experiments were performed on a cluster of Intel E5-2660 machines running at 2.20 GHz, with a time cut-off of 15 min and a memory limit of 4 GiB. Our code and models are publicly available [Gnad et al., 2021b].

We compare SPIN with standard options, i. e., with partial-order reduction enabled, Cunf with the cut-off rule of Esparza et al. [2002], and decoupled search (DecNDFS), using two kinds of benchmarks: (1) two scaling examples to showcase the behaviour on well-known models. One is an encoding of the dining philosophers problem, the other is a ring-shaped synchronisation topology. Both are illustrated in Figure 19.5. The philosophers model has $2N$ NBAs, N philosophers and N forks, synchronized by global transitions $l_G^{lF_i}$ and $l_G^{rF_i}$. After synchronizing with its left and right fork, a philosopher

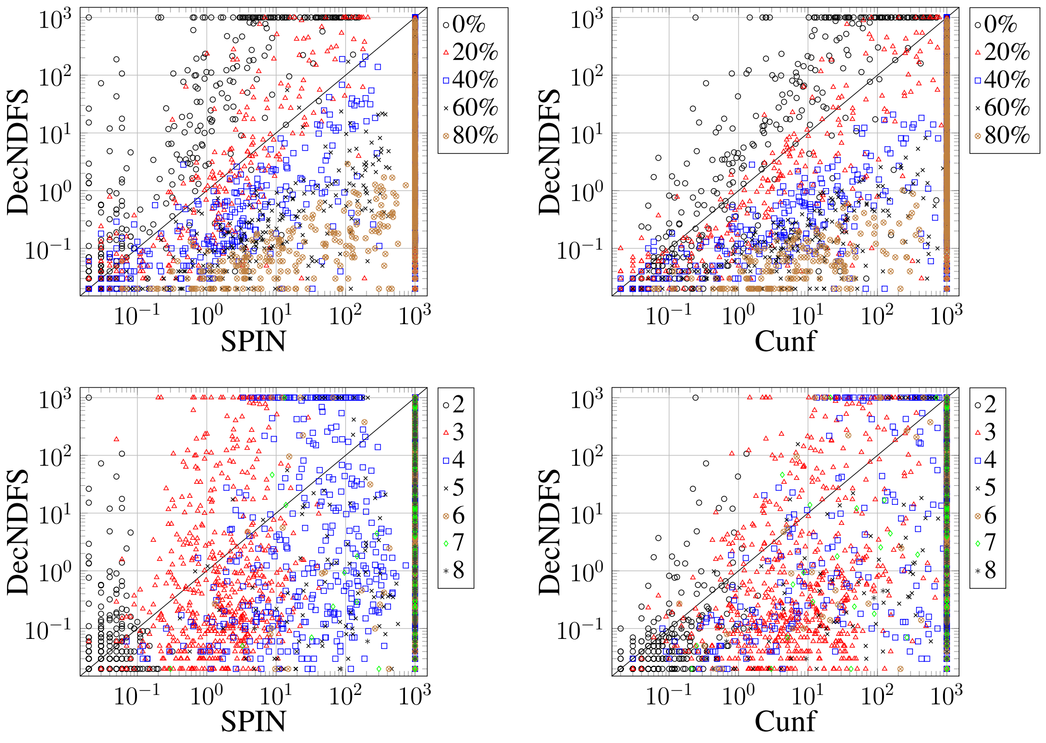


Figure 19.7: Scatterplots with the runtime of DecNDFS on the y-axis and the runtime of SPIN (left column) and Cunf (right column) on the x-axis, on randomly generated models. Each point represents one instance. In the top row, we highlight different ratios of local labels with different colors/shapes, in the bottom row we highlight different numbers of components.

can perform an internal *eat* transition; after releasing the forks it can perform an internal *think* transition. In the ring-topology model, each component can enter a diamond-shaped region via internal transitions, followed by a synchronization with its left or right neighbor via l_G^i or l_G^{i+1}. No accepting run exists for either model. Moreover, (2) we use a set of random automata, where for each combination of a ratio of internal transitions in $\{0\%, 20\%, \ldots, 80\%\}$, i. e., the number of transitions labelled with L_I divided by the total number of transitions, and a number of components in $\{2, \ldots, 8\}$, we generated sets of 150 random graphs. Each component has 15 to 100 local states, out of which up to 3% are accepting (at least one). We ensure that none of the instances has an internal accepting cycle to focus on more interesting cases. One could easily implement a lookup similar to **CheckLocalAccept**, which is necessary for DecNDFS, for the other methods, too, which then essentially simplifies the problem to basic reachability.

In Figure 19.6, we show detailed statistics for the scaling models, with increasing number of components $\#\mathcal{A}_B$ (Time in seconds, #States is the sum of states visited

Ratio	#	SPIN	Cunf	DecNDFS	$\#\mathcal{A}_B$	#	SPIN	Cunf	DecNDFS
					2	750	721	**750**	749
					3	750	696	**745**	712
0%	1050	**373**	369	319	4	750	411	243	**541**
20%	1050	385	384	**462**	5	750	130	114	**372**
40%	1050	397	384	**573**	6	750	49	53	**266**
60%	1050	422	394	**723**	7	750	24	34	**180**
80%	1050	468	431	**888**	8	750	14	23	**145**
$\sum$	5250	2045	1962	**2965**	$\sum$	5250	2045	1962	**2965**

Figure 19.8: Number of solved instances on the random models as a function of the ratio of internal transitions (left) and the number of components $\#\mathcal{A}_B$ (right).

in both DFSs, #E is the number of events in the prefix, Memory in MiB). In dining philosophers, SPIN and DecNDFS show similar results. SPIN has a runtime advantage in the larger instances of roughly a factor of 2, but DecNDFS uses only a fraction of the memory. Cunf clearly outperforms both. This model is not very well suited to decoupled search. Only half of the NBAs have internal transitions, and only two each, and there are no non-deterministic transitions that DecNDFS could represent compactly. On the ring-topology model, SPIN manages to exhaust the search space for up to 9 components. Cunf and DecNDFS scale significantly higher, the number of decoupled states grows only linearly in the number of components. Cunf on the other hand does show a blow-up and runs out of memory between 50 and 75 components. This showcase example only serves to illustrate a near-to-optimal case for decoupled search reductions, which likely does not carry over in this extent to real-world models.

In Figure 19.7, we show detailed runtime behaviour in terms of scatter plots with a per-instance comparison on the random models. Each point corresponds to one instance, where the x-value is the runtime of SPIN, resp. Cunf, and the y-value is the runtime of DecNDFS, so points below the diagonal indicate an advantage of DecNDFS. Different ratios of internal labels (top row) and numbers of components (bottom row) are depicted in different colors/shapes. We observe that, as expected, with a higher ratio of internal transitions, the advantage of DecNDFS increases significantly. For all ratios, DecNDFS clearly improves with a higher number of components.

In Figure 19.8, for the same benchmark set we show the number of solved instances as a function of the ratio (left) and of the number of components (right). Observe that, from around 20% internal transitions, DecNDFS consistently beats both SPIN and Cunf. SPIN and Cunf also benefit from the decrease in synchronizing statements, although not as much as DecNDFS. On the right, we see that starting with 4 component NBAs ($\#\mathcal{A}_B$), DecNDFS consistently beats SPIN and Cunf. While SPIN and Cunf show a significant decline with more components, this effect is less pronounced for DecNDFS.

Chapter 20

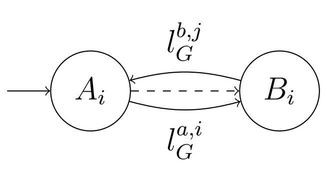
Related Work – Exponential Separations

As we observed in the context of AI planning (see Chapter 6), decoupled state-space search relates to several existing state-space reduction techniques. In the setting of verification of safety and liveness properties on composed non-deterministic automata, the most closely related techniques are partial-order reduction and Petri-net unfolding.

Decoupled search relates to partial-order reduction (POR) techniques (e. g. Valmari, 1992; McMillan, 1992; Godefroid, 1996; Esparza et al., 2002; Rodríguez and Schwoon, 2013), in that it avoids interleavings of internal traces. It can be viewed as a variant of unfolding, exploiting component independence by organizing the unfolding in terms of global transition paths, which ensures by design that there are no cross-component conflicts. We show that decoupled search can have exponential advantages over both methods, extending our findings from AI planning to the model checking field.

The connections to symbolic state representations [McMillan, 1993], and symmetry breaking [Clarke et al., 1993; Emerson and Sistla, 1993; Sistla et al., 1997] are weak, analogously to our observations in the planning context; the reduction achieved by both methods is in general incomparable. For safety checking, examples similar to those for planning show that decoupled search can have advantages over both techniques, and vice versa. Since for liveness checking the state space also needs to be exhausted in the first place, we expect simple adaptations of such examples to show exponential separations also for that setting. This is indeed the case for the separating examples that we use in our comparison to POR and unfolding. In this chapter, we hence focus on ample-sets pruning, as a representative of partial-order reduction typically employed with depth-first search, and Petri-net unfolding.

For liveness checking, there exists an alternative approach to nested depth-first search (NDFS) that is based on state-space search, namely algorithms that compute the strongly-connected components of the state space [Lichtenstein and Pnueli, 1985; Clarke et al., 1986; Couvreur, 1999; Geldenhuys and Valmari, 2004]. In this work, we did not con-



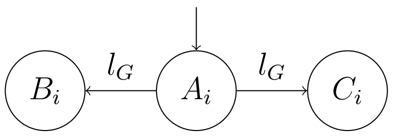


Figure 20.1: Illustration of the exponential separations to ample sets (left) and unfolding (right) for liveness checking.

sider this alternative and focused on NDFS. We believe that decoupled search could be adapted to this approach, too, but leave its exploration to future work.

20.1 Exponential Separations

In this section, we show that decoupled search can be exponentially more efficient than alternative methods. We capture this formally in terms of *exponential separations*, which we define analogously to Chapter 2.4 for models of composed automata.

Definition 60 (Exponential Separation)**.** *Let $\{\mathcal{M}^n{=}\mathcal{A}^1, \dots, \mathcal{A}^m \mid n \in \mathbb{N}^+\}$ be a family of models of size (number of states and labels) polynomially related to n. Then search method X is* exponentially separated *from search method Y if (i) the representation size of the reachable state space under X is bounded by a polynomial in n, while (ii) the representation size of the reachable state space under Y is exponential in n.*

First, we look into exponential separations for safety checking, then we extend these findings to liveness verification.

20.1.1 Safety Checking

We have investigated the relation of decoupled search to other state-space reduction methods in the context of AI planning in Part II of this work, in particular to partial-order reduction via strong stubborn sets [Valmari, 1992], and Petri-net unfolding [Esparza and Heljanko, 2000, 2001]. For both techniques, there exist families of scaling examples where decoupled search is exponentially more efficient.

We next describe two scaling models showing that the ample sets variant implemented in SPIN [Holzmann and Peled, 1994; Peled, 1996], as a representative for partial-order reduction in explicit-state search, and Petri-net unfolding are exponentially separated from decoupled search employed on sets of composed automata. We give formal proofs in the following:

Theorem 42. *Decoupled search is exponentially separated from ample-sets pruning.*

Proof. Consider the model depicted in the left of Figure 20.1 (the dashed transition is internal). There are n such NFA, each with the same state space: two local states A_i, B_i, initial state A_i, two global labels $l_G^{a,i}$, $l_G^{b,j}$, one internal label. The global transitions synchronize components pairwise; our argument holds for every possible such synchronization.

Under ample set pruning, no reduction is achieved (no state is pruned) because there is a global transition enabled in every state. Thus, there exists no state where only *safe* (i.e. internal) transitions are enabled, and the search always branches over all enabled transitions of all components. The decoupled state space, in contrast, only has a single decoupled state, where both local states are reached in each component. All decoupled successor states are dominated by the initial decoupled state and will be pruned. □

The ample sets from Holzmann and Peled [1994] only consider internal transitions as safe if there is no global transition enabled in the same state. Thus, no reduction can be achieved in the presented model. Compared to that, decoupled search represents the entire state space in only a single decoupled state, even though there is only one internal transition per component.

Similar to decoupled search, Petri-net unfolding exploits component independence by a special representation. Instead of searching over composed states and pruning transitions, the states of individual components are maintained separately.[1]

Theorem 43. *Decoupled search is exponentially separated from Petri-net unfolding.*

Proof. Consider the model illustrated in the right of Figure 20.1. There are n such components, each with the same state space with three local states A_i, B_i, C_i, a global label l_G, and transitions as shown in the figure. In a Petri net, this model is encoded with $3n$ places and 2^n transitions, one for every combination of one output place in each of the components. In the unfolding, this results in an event (the equivalent of a state) for every net transition. The decoupled state space has only two decoupled states: the initial state where $\{A_i\}$ is reached for all components, and its l_G-successor where $\{B_i, C_i\}$ is reached in every component. □

The advantage of decoupled search here results from the compact representation of the outcome states of non-deterministic transitions. Both local states that are reached via the l_G-transition of every component are contained in the decoupled successor state, which hence represents 2^n composed states.

In the context of SPIN, decoupled search is exponentially separated from *statement merging*, too, which is a weak form of partial-order reduction. It can only reduce the number of states local to a process, merging statements that only touch local variables.

[1]Recall that a general difference between the two methods is that checking reachability of a conjunctive property is linear in the number of reached decoupled states, but **NP**-complete for an unfolding prefix [McMillan, 1992].

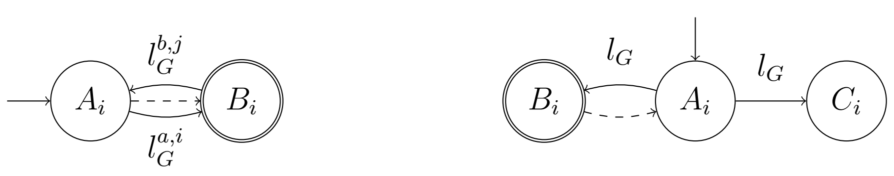


Figure 20.2: Illustration of the exponential separations to ample sets (left) and unfolding (right) for liveness checking.

It cannot merge statements that have conditions on global variables, and thus cannot tackle the exponential search-space size in, e. g., the logistics example described in Chapter 18.2. Similar arguments apply to reduction methods based on *τ-confluence* (e. g. Groote and Sellink, 1995; Groote and van de Pol, 2000). Moreover, note that internal transitions, while local to a process, may be relevant to the property being checked (e. g., be part of a conjunctive reachability property as in planning). All local states can still be observed when using decoupled search, while this is not necessarily the case for statement merging and τ-confluence.

20.1.2 Liveness Checking

We close our comparison to related techniques by showing that the exponential separations to ample-sets pruning and Petri-net unfolding from the previous chapter carry over to liveness checking by simple modifications of the models, illustrated in Figure 20.2.

Theorem 44. ***CheckEmptiness*** *with decoupled search is exponentially separated from* ***CheckEmptiness*** *with explicit-state search and ample sets pruning.*

Proof. The argument from the proof of Theorem 42 remains valid. With the states B_i accepting (see Figure 20.2, left), explicit-state search with ample sets pruning in the worst case has to exhaust the entire state space. It invokes **NestedDFS** on the accepting state $(B_1, \dots, B_n)$ and, worst-case, needs to exhaust the state space again to detect the cycle. Decoupled search invokes **NestedDFS** on the initial state restricted to the component states B_i. Every successor of that decoupled state closes the cycle via an arbitrary l_G^b transition. So there are only three decoupled states overall (including the acceptance-restricted initial state). $\square$

For unfolding, we only compare decoupled NDFS to the construction of a finite prefix for the model, which is required to perform the verification of the property.[2]

Theorem 45. ***CheckEmptiness*** *with decoupled search is exponentially separated from constructing a complete unfolding prefix.*

[2]We remark that, for liveness checking, a weaker cut-off rule is required, that can only increase the prefix size [Esparza and Heljanko, 2000].

Proof. Compared to the model from the proof of Theorem 43, the component states B_i are made accepting and internal transitions $B_i \to A_i$ are added (see the illustration in the right of Figure 20.2). Unfolding constructs a complete prefix as described in the proof of Theorem 43, and adds one more event for each new internal transition. Decoupled search generates the l_G-successor $s_1^{\mathcal{D}}$ of the initial decoupled state, which has $\{A_i, B_i, C_i\}$ reached for all components. The successor of $s_1^{\mathcal{D}}$ via l_G is pruned. **NestedDFS** is invoked on the restriction of $s_1^{\mathcal{D}}$ to B_i, in which all A_i get reached via the new internal transitions. The l_G-successor of that state closes the cycle, so there are only four decoupled states. □

The scaling examples that show the exponential separation for liveness checking are simple adaptation of those used in the case of safety checking. Decoupled search can lead to exponential advantages over established methods even on very small, almost trivial, models. We conclude that decoupled search has the potential to outperform these methods in practice whenever a given model shows characteristics that benefit the decoupled state representation.

Chapter 21

Summary

In this part of the thesis, we adapted decoupled state-space search to the setting of model checking. In particular, we addressed the verification of safety and liveness properties in systems modeled by several automata that synchronize on a set of shared global labels. Our approach is formulated for systems composed of a set of non-deterministic finite automata (NFA) for safety checking, respectively non-deterministic Büchi automata (NBA) for liveness checking. We formally defined the *decoupled composition* of such automata and proved that it captures reachability of the underlying composed system states exactly, i. e., the approach is sound and complete.

With this correctness result, it is straightforward to use the decoupled composition for safety checking, where properties ϕ are given as state features that need to hold in every reachable system state. Thus, the absence of a reachable state that violates ϕ is a proof that the property is satisfied by the system. We implemented decoupled search as an extension of the well-known SPIN model checker. The Promela models that SPIN takes as input can naturally be decomposed for decoupled search. Every *process* in the model becomes a separate component, and these interact via a set of shared global variables and channels. Such a decomposition then instantiates a star topology as in the AI planning formulation, where processes interact only via global (center) structures.

Our implementation in SPIN is still preliminary in that it does not support the full Promela language. Nevertheless, on models expressible in the supported language fragment, we have observed a significant state-space reduction. On a set of existing benchmarks, where the number of processes can be scaled linearly, decoupled search strongly outperforms the default configuration of SPIN that uses statement merging and ample-sets pruning as partial-order reduction techniques. On two models that we designed ourselves, a client-server architecture and a variant of our logistics example, we showcased the potential pruning power of decoupled search. In the client-server model, for instance, when there are internal transitions in the clients, decoupled search yields huge savings compared to the default SPIN configuration.

For liveness checking, we adapted a standard on-the-fly algorithm for checking ω-regular properties, nested depth-first search (NDFS), and proved its correctness. The necessary adaptations essentially pertain to the conditions that identify the existence of accepting runs, which must be handled differently given the different properties of decoupled states. A naïve adaptation of NDFS results in an incomplete and unsound algorithm, so accepting runs could be missed, and spurious cycles could be reported. We addressed this by introducing a decoupled-state *splitting* technique, that, in the inner depth-first search, keeps track of the accepting member states from which the nested search was invoked. Moreover, we need a special handling of cyclic runs induced only by internal transitions, which can be done efficiently by precomputing accepting member states that lie on such cycles. With this, and slightly adapted termination conditions to account for the decoupled-state representation, our new NDFS algorithm is correct.

Our approach is sufficiently general to cover significant cases like automata-based LTL model checking. This extends the scope of decoupled search from safety properties, which is conceptually close to AI planning, to liveness properties. Our experimental evaluation has shown that decoupled search can yield significant reductions in search effort across random models that consist of a set of synchronized NBAs, and simple scaling showcase examples. We compared decoupled search, implemented in our own new model checker, against SPIN and the Cunf Petri-net unfolding tool. We show results on two scaling examples, one where decoupled search does not perform well because of lack of internal transitions, and another where decoupled search excels. On the former, decoupled NDFS still outperforms SPIN in terms of memory usage (because of smaller state-space size), but the runtime overhead leads to a slowdown of a factor of two in large instances. Cunf performs at lot better than both on this model. On the second example, decoupled NDFS significantly beats both other approaches. On the randomly generated models, we observe that, as expected, the advantage of decoupled search increases with (1) a higher number of system components, and (2) larger percentage of internal over global transition labels. For high values of both, we see an advantage of up to four orders of magnitude over both competitors.

Finally, we analyzed the theoretical relation of decoupled search to ample-sets pruning, a variant of partial-order reduction used in SPIN, and Petri-net unfolding. Similar to the planning setting, we proved that decoupled search is exponentially separated from both methods by using small scaling example models where decoupled search results in a compact state-space representation, while the other two methods need to explore an exponential search space. This emphasizes our findings in the AI planning context, showing that decoupled state-space search is a novel approach that tackles the state explosion problem in a unique way.

Part V

Conclusion

Chapter 22

Conclusion

Decoupled state-space search is a novel technique designed to tackle the state explosion problem. We formally introduced decoupled search with a focus on classical AI planning, and extended its scope to model checking of safety and liveness properties.

Decoupled search exploits the structure of planning tasks, in particular the conditional independence between the leaf components in a star-topology decomposition of a task. Given this independence, it can compactly represent large sets of states, exponentially reducing the search effort. We developed the concept of the decoupled state space, which captures the reachability of the underlying planning task exactly. This allowed us to connect decoupled search to arbitrary search algorithms, and, important for planning, in principle use every existing heuristic function. We formally analyzed the properties of the decoupled state space, and conceived dominance pruning techniques under which its size is guaranteed to be bounded by the size of the standard state space. For an automatic task decomposition, we devised several strategies that can identify star topologies, respectively more restricted fork or inverted-fork factorings. These are detected on the majority of standard planning benchmarks. While the relevance of these benchmarks for practical purposes can be questioned, we argue that star topologies naturally arise in many human-designed systems, such as distributed client-server architectures, multi-agent systems, or modern multi-core processors working on shared memory.

We showed that the reduction achieved by decoupled search can lead to significant advantages over existing state-space reduction methods. In particular, we proved that decoupled search is exponentially separated from well-known related techniques, namely partial-order reduction via stubborn sets, Petri-net unfolding, symmetry breaking, and symbolic state representation with binary decision diagrams. This confirms that decoupled search is indeed a novel method that uniquely exploits the structure of planning tasks. Our decoupled search implementation extends the established Fast Downward framework. It is competitive with state-of-the-art planners in all common algorithmic planning problems—satisficing and optimal planning, and proving unsolvability—obtaining superior performance in presence of pronounced star topologies.

Given the observation that decoupled search is orthogonal to other reduction techniques, we developed combinations with strong stubborn sets, symmetry breaking, symbolic state representations, and state-dominance pruning. We showed the correctness of the combined algorithms and provided a theoretical and empirical evaluation of their pruning power. We proved that decoupled strong stubborn sets, decoupled symmetry breaking, and enhanced dominance pruning for fork topologies can lead to exponential advantages over the respective base methods. Our experimental evaluation confirmed these findings by discovering that there are standard planning benchmarks where the combined algorithm can lead to huge savings.

In model checking, we introduced decoupled search for systems of composed automata that synchronize on a set of shared labels. Similar to planning, we developed the decoupled automata composition and proved its correctness. For models using non-deterministic finite automata, employed for safety checking, we evaluated our decoupled-search extension of the established SPIN model checker and showcased its benefits on a set of existing Promela benchmarks. Here, decoupled search outperforms ample-sets pruning, the partial-order reduction variant implemented in SPIN. On the theoretical side, we show that, in its automata-based formulation for safety checking, decoupled search is exponentially separated from Petri-net unfolding and partial-order reduction via ample sets. For checking liveness properties, on models composed of non-deterministic Büchi automata, we have derived a variant of the nested depth-first search algorithm. This required a non-trivial adaptation of the decoupled state space in the inner search to ensure that no accepting runs are missed. We proved our adapted algorithm correct, and showed that it is exponentially separated from ample-sets pruning and Petri-net unfolding in the context of liveness checking. Empirically, our prototype implementation consistently outperforms SPIN on models with several components or a small percentage of internal (i. e., leaf) transitions.

For the future, one of the most interesting challenges is to adopt decoupled state space search to alternative formalisms in AI planning, model checking, and potentially beyond. In the context of planning, we believe that it is highly promising to move into richer frameworks, such as numeric planning, non-deterministic or probabilistic models, or planning with conditional effects and axioms. Similar formalisms are also common in the field of formal verification, in particular timed automata and probabilistic models look very interesting. Extending the scope of liveness properties that can be verified, CTL (computation tree logic) properties [Kupferman et al., 2000], as well as generalized Büchi acceptance [Tauriainen, 2006], so language intersection of the involved automata, seem to be viable topics for future research. On the application side, some of the aforementioned multi-core processors open up a highly relevant area, namely the verification of weak-memory models, where decoupled search can potentially lead to strong state-space reductions [Jonsson, 2008; Linden and Wolper, 2013; Travkin et al., 2013; Alrahman et al., 2014; Podkopaev et al., 2019].

Moreover, despite the thorough development and analysis of decoupled search in this work, there are still several questions that can be addressed in the future. In planning, we expect a generalization of star factorings to more flexible topologies without a center factor, similar to our presentation using automata, to further extend the applicability of decoupled search. Also, as we have seen in the experimental evaluation on planning benchmarks, it is still not clear how to determine upfront if a given factoring will result in good search performance. This is likely dependent on the algorithmic problem that is tackled, as we have observed that, e. g., inverted-fork topologies beat state-of-the-art planners by a large margin in satisficing planning, but are not competitive in optimal planning. What characterizes a good factoring for a given purpose?

In model checking, an obvious next step is the integration of orthogonal methods into decoupled search, as already done in planning. Partial-order reduction, symmetry breaking, and symbolic representations are commonly employed in that area. Thus, we believe it is highly promising to develop these combined algorithms also in that context, and obtain a system that has the strengths of multiple state-space reduction techniques.

Overall, decoupled state-space search is a very generic concept that can, in principle, be applied to many types of models that are based on factored state representations, where an overall system is described as a set of synchronized components. Many systems formalized in this way can probably be decomposed using decoupled search. This opens a wide space for applications of decoupled search, not only in the fields of planning and model checking, but whenever large state spaces that arise from the product of multiple components need to be explored.

Appendices

Appendix A

Full Proofs of Part II

A.1 Correctness of the Decoupled State Space

To capture decoupled-state reachability, we introduce an intermediate concept, *embedded states*, exhibiting the link between member states and decoupled states. Instead of an explicit leaf-state assignment, embedded states contain a link to the respective compliant-path graph vertex:

Definition 61 (Embedded State)**.** *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π with center C and leaves $\mathcal{L}$. For a decoupled state $s^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{F}}$, an* embedded state *in $s^{\mathcal{F}}$ is a function $\widehat{s}$ on $\mathcal{F}$, mapping C to $\mathsf{center}(s^{\mathcal{F}})$, and mapping each $L \in \mathcal{L}$ to a vertex s_n^L in $\mathrm{CPG}_{\Pi}(\pi^C(s^{\mathcal{F}}), L)$ with $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$, where $n := |\pi^C(s^{\mathcal{F}})|$. The set of all embedded states of $s^{\mathcal{F}}$ is the* embedded hypercube *of $s^{\mathcal{F}}$, denoted $\widehat{[s^{\mathcal{F}}]}$.*

The initial embedded state*, $\widehat{I}$, maps C to $\mathsf{center}(\mathcal{I}^{\mathcal{F}}) = I[C]$ and, for each $L \in \mathcal{L}$, maps L to the vertex $I[L]_0$ in $\mathrm{CPG}_{\Pi}(\langle\rangle, L)$.*

Given a decoupled state $s^{\mathcal{F}}$, as $\mathrm{CPG}_{\Pi}(\pi^C(s^{\mathcal{F}}), L)$ contains exactly one vertex s_n^L for every $s^L \in S^L$, the member states and embedded states of $s^{\mathcal{F}}$ are in one-to-one correspondence. Given a decoupled state $s^{\mathcal{F}}$ and member state $s \in [s^{\mathcal{F}}]$, we denote the unique corresponding embedded state by $\widehat{s}$, and vice versa.

Intuitively, embedded states $\widehat{s}$ serve to "track the progress of the corresponding states s through the decoupled state space". We formalize this through a notion of *embedded transitions*:

Definition 62 (Embedded Transitions)**.** *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π with center C and leaves $\mathcal{L}$. Let $s^{\mathcal{F}}$ be a decoupled state in $\Theta_{\Pi}^{\mathcal{F}}$, and $\widehat{s} \in \widehat{[s^{\mathcal{F}}]}$ an embedded state. Then $\widehat{s} \xrightarrow{a} \widehat{t}$ is an* embedded transition

(i) on L in $\Theta_{\Pi}^{\mathcal{F}}$, if $a \in \mathcal{A}_{\not{C}}^{L}$, $\mathrm{CPG}_{\Pi}(\pi^C(s^{\mathcal{F}}), L)$ contains an arc $\widehat{s}(L) \xrightarrow{a} \widehat{t}(L)$, and for all $L \neq F \in \mathcal{F}$ we have $\widehat{s}(F) = \widehat{t}(F)$;

(ii) on C in $\Theta_\Pi^\mathcal{F}$, if $a \in \mathcal{A}^C$, $s^\mathcal{F} \xrightarrow{a} t^\mathcal{F}$ is a transition in $\Theta_\Pi^\mathcal{F}$, and for all $L \in \mathcal{L}$ we have that $\mathrm{CPG}_\Pi(\pi^C(t^\mathcal{F}), L)$ contains an arc $\widehat{s}(L) \xrightarrow{0} \widehat{t}(L)$.

We refer to paths of embedded transitions as embedded paths. *The* cost *of an embedded path $\widehat{\pi}$, denoted* $\mathsf{cost}(\widehat{\pi})$*, is the summed-up cost of its transition labels.*

Note here that, necessarily by the construction of $\widehat{t}$ and compliant path graphs, $\widehat{t} \in [\widehat{s^\mathcal{F}}]$ in (i), and $\widehat{t} \in [\widehat{t^\mathcal{F}}]$ in (ii). Decoupled-state reachability is captured in the following form:

Lemma 18. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Let $\mathcal{S}^\mathcal{F}$ be the decoupled states in $\Theta_\Pi^\mathcal{F}$. For any $s^\mathcal{F} \in \mathcal{S}^\mathcal{F}$, $\widehat{s} \in [\widehat{s^\mathcal{F}}]$, and $\widehat{t}$ reachable from $\widehat{s}$ in $\Theta_\Pi^\mathcal{F}$, there exists $t^\mathcal{F} \in \mathcal{S}^\mathcal{F}$ such that $\widehat{t} \in [\widehat{t^\mathcal{F}}]$ and $t^\mathcal{F}$ is reachable from $s^\mathcal{F}$ in $\Theta_\Pi^\mathcal{F}$. Vice versa, for any $s^\mathcal{F}, t^\mathcal{F} \in \mathcal{S}^\mathcal{F}$ where $t^\mathcal{F}$ is reachable from $s^\mathcal{F}$ in $\Theta_\Pi^\mathcal{F}$, and for any $\widehat{t} \in [\widehat{t^\mathcal{F}}]$, there exists $\widehat{s} \in [\widehat{s^\mathcal{F}}]$ such that $\widehat{t}$ is reachable from $\widehat{s}$ in $\Theta_\Pi^\mathcal{F}$.*

Proof. For the first part of the claim, $t^\mathcal{F}$ and $\widehat{t}$ as specified must exist simply because the individual transitions on an embedded path from $\widehat{s}$ to $\widehat{t}$ all follow decoupled transitions (C) respectively compliant-path graph arcs ($\mathcal{L}$) present in $\Theta_\Pi^\mathcal{F}$. For the second part of the claim, if $\pi^\mathcal{F}$ is a decoupled path from $s^\mathcal{F}$ to $t^\mathcal{F}$, then we can backchain from $\widehat{t}$ through the compliant-path graphs along $\pi^\mathcal{F}$ to obtain the desired embedded state $\widehat{s}$ in $s^\mathcal{F}$. □

Having clarified the basics of embedded states and how they capture reachability, let us get back to the link with member states. The core observation is that, like the member states and embedded states themselves, their transitions also are in one-to-one correspondence:

Lemma 19. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π with center C and leaves $\mathcal{L}$. Let $s^\mathcal{F}$ be a decoupled state in $\Theta_\Pi^\mathcal{F}$. Then, for any member state $s \in [s^\mathcal{F}]$ and action a, $s \xrightarrow{a} t$ is a transition in Π if and only if $\widehat{s} \xrightarrow{a} \widehat{t}$ is an embedded transition in $\Theta_\Pi^\mathcal{F}$.*

Proof. From left to right, say a is applicable to s and $t = s[\![a]\!]$. We distinguish two cases. First, a is a non-center action, $a \notin \mathcal{A}^C$. Then, as $\mathcal{F}$ is a star factoring, a affects a single leaf factor L, $a^L \in \mathcal{A}^L_{\not{C}}$. As a is applicable to s, we have $\mathsf{center}(s^\mathcal{F}) \models \mathsf{pre}(a)[C]$ and $s[L] \models \mathsf{pre}(a)[L]$. Therefore, by Definition 8 (i), the compliant path graph $\mathrm{CPG}_\Pi(\pi^C(s^\mathcal{F}), L)$ layer at time n corresponding to $s^\mathcal{F}$ contains the arc $s[L]_n \xrightarrow{a} t[L]_n$, which establishes the desired embedded transition.

Second, say a is a center action, $a \in \mathcal{A}^C$. As a is applicable to s, for every $L \in \mathcal{L}$ where $\mathsf{pre}(a)[L] \neq \emptyset$, there exists a finite-price leaf state in $s^\mathcal{F}$, namely $s[L]$. Hence there exists a transition $s^\mathcal{F} \xrightarrow{a} t^\mathcal{F}$ in $\Theta_\Pi^\mathcal{F}$. Furthermore, for each L, by Definition 8 (ii) the compliant path graph $\mathrm{CPG}_\Pi(\pi^C(t^\mathcal{F}), L)$ contains the arc $s[L]_n \xrightarrow{0} t[L]_{n+1}$. Together, these establish the desired embedded transition.

From right to left, say $\widehat{s} \xrightarrow{a} \widehat{t}$ is an embedded transition in $\Theta_{\Pi}^{\mathcal{F}}$. Distinguishing the same two cases, if $a^L \in \mathcal{A}_{\not C}^L$ then $\widehat{s} \xrightarrow{a} \widehat{t}$ is an L-transition. By Definition 62 (i), $\widehat{s}$ and $\widehat{t}$ differ only in terms of taking a single arc on L, so s and t differ only on L. By Definition 8 (i), $\mathsf{center}(s^{\mathcal{F}}) \models \mathsf{pre}(a)[C]$, $s[L] \models \mathsf{pre}(a)[L]$, and $s[L][\![a]\!] = t[L]$. But this immediately implies that a is applicable to s and $t = s[\![a]\!]$, as desired.

Say finally that $a \in \mathcal{A}^C$ so $\widehat{s} \xrightarrow{a} \widehat{t}$ is an C transition. By Definition 62 (ii), $\widehat{t}$ results from $\widehat{s}$ by updating the center state according to a, and taking the arc $\widehat{s}(L) \xrightarrow{0} \widehat{t}(L)$ for every L. By Definition 8 (ii) for every L we have $s[L] \models \mathsf{pre}(a)[L]$ and $s[L][\![a]\!] = t[L]$. But then, again a is applicable to s and $t = s[\![a]\!]$, concluding the proof. □

We are now ready to prove the two core lemmas:

Lemma 1. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Let $s^{\mathcal{F}}$ be a reachable decoupled state in $\Theta_{\Pi}^{\mathcal{F}}$. Then:*

(i) $[s^{\mathcal{F}}]$ is exactly the set of states s for which there exists a path π, from $\mathcal{I}$ to s in Θ_{Π}, where $\pi^C(\pi) = \pi^C(s^{\mathcal{F}})$.

(ii) For every $s \in [s^{\mathcal{F}}]$, the cost of a cheapest such path π is $\mathsf{cost}(\pi^C(s^{\mathcal{F}}))+\mathsf{price}(s^{\mathcal{F}}, s)$.

Proof. To prove (i), say first that $s \in [s^{\mathcal{F}}]$. Consider the corresponding embedded state $\widehat{s}$. As all compliant path graph vertices in $\widehat{s}$ are reached at $s^{\mathcal{F}}$, for every L there must exist a path $\pi[L]$ from $I[L]_0$ to $\widehat{s}(L)$ in $\mathrm{CPG}_{\Pi}(\pi^C(s^{\mathcal{F}}), L)$. From the collection of these paths, along with $\pi^C(s^{\mathcal{F}})$, we obtain an embedded path $\widehat{\pi}$ from $\widehat{I}$ to $\widehat{s}$: simply interleave the embedded C transitions induced by $\pi^C(s^{\mathcal{F}})$ with the embedded L transitions induced by $\pi[L]$. With Lemma 19, from $\widehat{\pi}$ we obtain a path π as desired.

Say now that π is a path in Π from I to some s, where the center action subsequence of π is $\pi^C(s^{\mathcal{F}})$. With Lemma 19, we obtain an embedded path $\widehat{\pi}$ from $\widehat{I}$ to $\widehat{s}$. Collecting the L transitions from $\widehat{\pi}$ for each L, clearly we get paths $\pi[L]$ from $I[L]_0$ to $\widehat{s}(L)$ in $\mathrm{CPG}_{\Pi}(\pi^C(s^{\mathcal{F}}), L)$. Therefore, $s \in [s^{\mathcal{F}}]$ as desired.

Claim (ii) now follows directly, because the above shows that, for every $s \in [s^{\mathcal{F}}]$, the paths π as specified are in one-to-one correspondence with $\pi^C(s^{\mathcal{F}})$ augmented with the possible selections of $\mathrm{CPG}_{\Pi}(\pi^C(s^{\mathcal{F}}), L)$ paths from $I[L]_0$ to $s[L]_n$, where $n := |\pi^C(s^{\mathcal{F}})|$. Thus, by the definition of pricing functions, the cheapest such π has exactly the specified cost. □

Lemma 2. *Let Π be a planning task, and $\mathcal{F}$ a factoring for Π. Let s be a reachable state in Π, and let π be a path reaching s. Then there exists a reachable decoupled state $s^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{F}}$ so that $s \in [s^{\mathcal{F}}]$, and $\pi^C(\pi) = \pi^C(s^{\mathcal{F}})$.*

Proof. With Lemma 19, π corresponds to an embedded path $\widehat{\pi}$ from $\widehat{I}$ to $\widehat{s}$. By Lemma 18, there exists a decoupled state $s^{\mathcal{F}}$ such that $\widehat{s} \in \widehat{[s^{\mathcal{F}}]}$, and $s^{\mathcal{F}}$ is reachable from $\mathcal{I}^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{F}}$. Clearly, the decoupled transitions taken in reaching $s^{\mathcal{F}}$, according to the proof of Lemma 18, correspond exactly to the center action subsequence of π. □

A.2 Decoupled State-Space Size

In what follows, we consider N-vectors $v \in R^N$ over some subset $R \subseteq \mathbb{R}^{0+}$ of non-negative reals. For the special case of $N = 1$, we identify v with $v[1]$. We consider (possibly infinite) sequences $\vec{v} = v_0, v_1, v_2, \dots$ of vectors. We define the relation $\succ$ over vectors by saying that $v \succ v'$ iff there exists a vector position $1 \leq k \leq N$ so that $v[k] > v'[k]$. We say that a vector sequence $\vec{v}$ is *descending* if, whenever v precedes v' in the sequence, $v \succ v'$. We say that R *has an infinite descending* N*-sequence* if there exists an infinite descending sequence of N-vectors.

Theorem 3 (Finiteness under Dominance Pruning). *Let* Π *be a planning task, and* $\mathcal{F}$ *a factoring for* Π. *Under ancestor dominance pruning,* $\Theta_{\Pi}^{\mathcal{RF}}$ *is finite.*

Proof. Consider the non-pruned paths $\pi^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{RF}}$. Observe that such paths necessarily are descending: some prices along $\pi^{\mathcal{F}}$ must descend each time we encounter the same center state, as otherwise the new state would be dominated by some previous state. We prove that there is no infinite descending path, i. e., every $\pi^{\mathcal{F}}$ has finite length. As every decoupled state $s^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{RF}}$ must be the endpoint of such a path, and as the branching factor is finite, this proves the claim.

Assume to the contrary that there is an infinite descending path $\pi^{\mathcal{F}}$. As the number of different center states is finite, there must exist a center state s^C visited infinitely often on $\pi^{\mathcal{F}}$. Denote by $\vec{s^{\mathcal{F}}} = s_0^{\mathcal{F}}, s_1^{\mathcal{F}}, \dots$ the sub-sequence of decoupled states along $\pi^{\mathcal{F}}$ where $\mathsf{center}(s_i^{\mathcal{F}}) = s^C$.

Collect, from each $s_i^{\mathcal{F}}$, the vector p_i of leaf state prices (using some arbitrary order of leaf states to fix the ordering of vector positions). Denoting by N the number of leaf states, $\vec{p} := p_0, p_1, \dots$ is a sequence of N-vectors over $\mathbb{R}^{0+} \cup \{\infty\}$. More precisely, $\vec{p}$ is a sequence of N-vectors over possible plan cost values, i. e., over $R \cup \{\infty\}$ where R contains $\sum_{i=1}^{n} \mathsf{cost}(a_i)$ for any finite sequence $\langle a_1, \dots, a_n \rangle$ of actions in Π.

Proposition 14 below shows that R has no infinite descending 1-sequence. Lemma 20 below shows that, given this, $R \cup \{\infty\}$ has no infinite descending N-sequence for any N. However, by construction, whenever p precedes p' on $\vec{p}$ then there exists a vector position k so that $p[k] > p'[k]$. That is, $\vec{p}$ is descending, in contradiction, showing the claim. □

Proposition 14. *Let* Π *be a planning task. Let* $R \subseteq \mathbb{R}^{0+}$ *be the set of numbers that contains* $\sum_{i=1}^{n} \mathsf{cost}(a_i)$ *for any finite sequence* $\langle a_1, \dots, a_n \rangle$ *of actions in* Π. *Then* R *does not have an infinite descending* 1*-sequence.*

Proof. Consider any sequence $\vec{v} = v_0, v_1, v_2, \dots$ of 1-vectors over R. Then, in particular, $v_i < v_0$ for all $i > 0$. But, for any $C \in \mathbb{R}^{0+}$, there is only a finite number of values $\sum_{i=1}^{n} \mathsf{cost}(a_i) < C$. This is because any non-0 cost action a can occur at most $\lfloor C/\mathsf{cost}(a) \rfloor$ times on $\langle a_1, \dots, a_n \rangle$. □

Lemma 20. *Let $R \subseteq \mathbb{R}$ be a set of numbers that has no infinite descending 1-sequence. Let $\vec{v}$ be a descending sequence of N-vectors over $R \cup \{\infty\}$. Then $\vec{v}$ is finite.*

Proof. Let $K \subseteq \{1, \dots, N\}$ be an arbitrary subset of vector positions. Denote by $\vec{v}[K, \infty]$ the subsequence of vectors v on $\vec{v}$ where $v[k] \neq \infty$ iff $k \in K$. In words, consider the subsequence of vectors that fits the "∞-profile" given by K. We show that $\vec{v}[K, \infty]$ is finite, which proves the claim as there is only a finite number of profiles.

Construct the vector sequence $\vec{v}[K, \infty]|_K$ by projecting each element of $\vec{v}[K, \infty]$ onto the position subset K. Then $\vec{v}[K, \infty]|_K$ is a sequence of vectors over $R \cup \{\infty\}$. By construction, as $\vec{v}[K, \infty]$ is a descending sequence, and because the elements of $\vec{v}[K, \infty]$ all agree on the positions outside K, we have that $\vec{v}[K, \infty]|_K$ is a descending sequence. It thus remains to show that every descending sequence of finite vectors over R is finite. We prove this by induction over N.

The induction base case, $N = 1$, holds by prerequisite as R has no infinite descending 1-sequence.

For the inductive case, assume that there is no infinite descending sequence of N-position vectors over R. We show that there is no infinite descending sequence of $N+1$-position vectors over R.

Let $\vec{w} = w_0, w_1, w_2, \dots$ be any descending sequence of $N+1$-position vectors over R. Denote $w_0 = (c^1, \dots, c^{N+1})$, where each c^j is a constant, i. e., $c^j \in R$.

Let $j \in \{1, \dots, N+1\}$ be arbitrary. Denote $C^j := \{c' \in R \mid c' < c^j\}$. Then C^j is finite because otherwise we could sequence its elements into an infinite descending 1-sequence over R. Let $c' \in C^j$ be arbitrary. Denote by $\vec{w}[j, c']$ the subsequence of vectors w on $\vec{w}$ where the j-th position has value c'. Obviously, every w_i for $i > 0$ must be contained in at least one $\vec{w}[j, c']$. We show that each $\vec{w}[j, c']$ is finite. As there is a finite number of choices of j' and c', this proves the claim.

Denote $K := \{1, \dots, N+1\} \setminus \{j\}$. Construct the vector sequence $\vec{w}[j, c']|_K$ by projecting each element of $\vec{w}[j, c']$ onto the position subset K. By construction, as $\vec{w}[j, c']$ is a descending sequence, and because the elements of $\vec{w}[j, c']$ all agree on the single position j that is outside K, we have that $\vec{w}[j, c']|_K$ is a descending sequence. As $\vec{w}[j, c']|_K$ is a sequence of vectors with N positions, by induction assumption, $\vec{w}[j, c']|_K$ is finite. Hence $\vec{w}[j, c']$ is finite as desired, concluding the argument. $\square$

We next prove that decoupled states capture the solvability of member states:

Lemma 3. *Let Π be a planning task, $\mathcal{F}$ a factoring for Π, and $s^{\mathcal{F}}$ a decoupled state. Then $s^{\mathcal{F}}$ is solvable if and only if at least one $s \in [s^{\mathcal{F}}]$ is solvable.*

Proof. From left to right, say the decoupled goal state $t^{\mathcal{F}}$ is reachable from $s^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{F}}$. Clearly, there exists a goal state $t \in [t^{\mathcal{F}}]$. By Lemma 18, there exists an embedded state $\widehat{s} \in \widehat{[s^{\mathcal{F}}]}$ such that $\widehat{t}$ is reachable from $\widehat{s}$ in $\Theta_{\Pi}^{\mathcal{F}}$. By Lemma 19, t is reachable from s in Π. Hence the state s is solvable as desired.

From right to left, say the goal state t is reachable from s in Π. By Lemma 19, $\widehat{t}$ is reachable from $\widehat{s}$ in $\Theta_{\Pi}^{\mathcal{F}}$. By Lemma 18, there exists a decoupled state $t^{\mathcal{F}}$ such that $\widehat{t} \in [\widehat{t^{\mathcal{F}}}]$ and $t^{\mathcal{F}}$ is reachable from $s^{\mathcal{F}}$ in $\Theta_{\Pi}^{\mathcal{F}}$. As t is a goal state, $t^{\mathcal{F}}$ is a decoupled goal state, as desired. □

Finally, we show the hardness of hypercube pruning:

Proposition 5. *Given a planning task Π and a factoring $\mathcal{F}$ for Π, it is co-**NP**-complete to decide whether reachable decoupled states $t_1^{\mathcal{F}}, \dots, t_n^{\mathcal{F}}$ cover a reachable decoupled state $s^{\mathcal{F}}$.*

Proof. Membership follows directly from the results by Hoffmann and Kupferschmid [2005] for general hypercube covering problems.

Hardness follows by reduction from the complement of SAT, extending Hoffmann and Kupferschmid's argument by a simple construction of Π and $\mathcal{F}$. Assume a CNF formula ϕ with propositional variables $P_1, \dots, P_n$ and clauses $C_1, \dots, C_m$. The construction of Π includes state variables $p_1, \dots, p_n$, each with domain $\{u, 0, 1\}$ where u is the initial value; there furthermore is a variable c with domain $\{u, 0, 1, \dots, m\}$. The goal does not matter for our purposes. The factoring $\mathcal{F}$ has center $\{c\}$ and leaves $\{\{p_1\}, \dots, \{p_n\}\}$.

The actions are as follows. For each clause C_j there is a center action a_j^C which is applicable to the initial state, and which allows to generate a hypercube corresponding to the truth-value assignments disallowed by c_j. Specifically, we set $\mathsf{pre}(a_j^C) = \{c = u\}$, and $\mathsf{eff}(a_j^C) = \{c = j\}$. Furthermore, we include leaf actions a_l^L, one for each literal $l \in C_j$, with $\mathsf{pre}(a_l^C) = \{c = j\}$, and $\mathsf{eff}(a_l^C) = \{\bar{l}\}$ where $\bar{l}$ is the opposite of l, i. e., $p_i = 1$ if $l = \neg P_i$, and $p_i = 0$ if $l = P_i$. Finally, we include leaf actions a_{ij0}^L and a_{ij1}^L for each variable P_i that does not occur in C_j, with $\mathsf{pre}(a_{ij0}^L) = \mathsf{pre}(a_{ij1}^L) = \{c = j\}$, $\mathsf{eff}(a_{ij0}^L) = \{p_i = 0\}$, and $\mathsf{eff}(a_{ij1}^L) = \{p_i = 1\}$. Observe that, once a_j^C has been applied, the hypercube $t_j^{\mathcal{F}}$ of reached leaf states over the variables p_i corresponds exactly to those assignments over P_i which do not satisfy C_j.

We finally include a center action a_0^C which is applicable to the initial state, and allows to generate a hypercube corresponding to *all* truth-value assignments. Specifically, we set $\mathsf{pre}(a_0^C) = \{c = u\}$, and $\mathsf{eff}(a_0^C) = \{c = 0\}$, and we include leaf actions a_{i00}^L and a_{i01}^L for each $1 \leq i \leq n$, with $\mathsf{pre}(a_{i00}^L) = \mathsf{pre}(a_{i01}^L) = \{c = 0\}$, $\mathsf{eff}(a_{i00}^L) = \{p_i = 0\}$, and $\mathsf{eff}(a_{i01}^L) = \{p_i = 1\}$. Observe that, once a_0^C has been applied, the hypercube $s^{\mathcal{F}}$ of reached leaf states over the variables p_i corresponds exactly to the space of all assignments over P_i.

Consider the time point in search where search has explored each of the alternatives $a_1^C, \dots, a_m^C$ (applied each of these actions to the initial state separately), and now explores the alternative a_0^C. Then all-visited hypercube pruning checks whether $t_1^{\mathcal{F}}, \dots, t_m^{\mathcal{F}}$ cover $s^{\mathcal{F}}$. The latter is the case iff ϕ is unsatisfiable. □

A.3 Relation to Petri-net Unfolding

A.3.1 Technical Background – Details

We spell out the concepts previously only outlined, and we give additional notations as needed in our proofs. Our definitions loosely follow Bonet et al. [2014].

Petri-Net Unfolding

A *net* N is a tuple $N = \langle P, T, F\rangle$, where P and T are sets of *places* and *transitions*. $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. For $z \in P \cup T$, we denote $\mathsf{pre}(z) := \{y \mid (y, z) \in F\}$ and $\mathsf{eff}(z) := \{y \mid (z, y) \in F\}$. For $Z \subset P \cup T$, we denote $\mathsf{pre}(Z) := \bigcup_{z\in Z} \mathsf{pre}(z)$ and $\mathsf{eff}(Z) := \bigcup_{z\in Z} \mathsf{eff}(z)$. A set of places $M \subseteq P$ is called a *marking*.[1] A Petri net $\Sigma = \langle N, M_0\rangle$ is a pair of a net $N = \langle P, T, F\rangle$ and *initial marking* $M_0 \subseteq P$. By $\preceq$, we denote the reflexive transitive closure of the flow relation F. Two nodes $y, y' \in P \cup T$ are *in conflict*, denoted $y\#y'$, if there exist distinct $t, t' \in T$ s.t. $\mathsf{pre}(t) \cap \mathsf{pre}(t') \neq \emptyset$, $t \preceq y$, and $t' \preceq y'$. Two nodes $y, y' \in P \cup T$ are *concurrent*, denoted $y \parallel y'$, if neither $y\#y'$ nor $y \preceq y'$ nor $y' \preceq y$.

The unfolding procedure builds a *branching process*, which is an *occurrence net* labeled with the places and transitions in Σ. An occurrence net $ON = \langle B, E, G\rangle$ is a net where B and E are called *conditions* and *events*, corresponding to places and transitions in a net. Occurrence nets have the following properties: they are acyclic, i. e., $\preceq$ is a partial order; for every $b \in B : |\mathsf{pre}(b)| \leq 1$; for every $y \in B \cup E$, $\neg(y\#y)$ and there are finitely many y' s.t. $y' \prec y$, where $\prec$ is the transitive closure of G. $\prec$ is called the *causality relation*, and an event f with $f \prec e$ is called a causal predecessor of e. $Min(ON)$ is the set of $\prec$-minimal elements of $B \cup E$.

A branching process Δ of a Petri net Σ is a pair $\Delta = \langle ON, \phi\rangle$ of an occurrence net ON and a homomorphism ϕ from ON to Σ. It is required that: $\phi : B \cup E \to P \cup T$ is a mapping from conditions/events to places/transitions such that $\phi(B) \subseteq P$, $\phi(E) \subseteq T$; for all $e \in E$ the restriction of ϕ on $\mathsf{pre}(e)$ is a bijection between $\mathsf{pre}(e)$ and $\phi(\mathsf{pre}(e))$ and the restriction of ϕ on $\mathsf{eff}(e)$ is a bijection between $\mathsf{eff}(e)$ and $\phi(\mathsf{eff}(e))$; $\phi(Min(ON)) = M_0$; and for all $e, f \in E$, if $\mathsf{pre}(e) = \mathsf{pre}(f)$ and $\phi(e) = \phi(f)$ then $e = f$. We say that $x \in B \cup E$ is *labeled* with y if $\phi(x) = y$.

A set of conditions D is called a *co-set* if for all $d \neq d' \in D : d \parallel d'$. A set of events $C \subseteq E$ is *causally closed* if for every $e \in C$, $f \prec e$ implies $f \in C$. A *configuration* C is a finite set of events that is causally closed and free of conflicts ($\forall e, f \in C : \neg(e\#f)$). By $[e] := \{f \mid f \preceq e\}$ we denote the *local configuration* of an event $e \in E$. For a configuration C, *Mark*$(C) := \phi((Min(ON) \cup \mathsf{eff}(C)) \setminus \mathsf{pre}(C))$ is a reachable marking of Σ. Intuitively, a configuration corresponds to a partially ordered plan.

[1]We only consider *safe* nets, where each place can hold only one token at a time, so that defining markings as sets of places is possible.

An event e is a *cut-off* if there exists a configuration C in Δ such that $\mathit{Mark}(C) = \mathit{Mark}([e])$. An event $e \in E$ labeled with a transition t is a possible *extension* of a configuration C in Δ if $C \cup \{e\}$ is a configuration, and there exists a co-set D in Δ such that no event in $\mathsf{pre}(D)$ is a cut-off, $|D| = |\mathsf{pre}(t)|$, $\phi(D) = \mathsf{pre}(t)$, and Δ contains no event e' with $\mathsf{pre}(e') = D$ where $\phi(e') = t$. We then say that e *fires* in C.

The *unfolding* process for Σ incrementally builds a branching process called a *complete prefix*, denoted Unf_Σ. The process starts from $Min(ON)$, and adds possible extensions while ones exist. The extensions e are added according to an order $\ll$ over their local configurations $[e]$. In each step, the $\ll$-minimal event e is considered. If e is not a cut-off, then new instances of $\mathsf{eff}(\phi(e))$ are added to Unf_Σ. Upon termination, all reachable markings of Σ are represented by a configuration in Unf_Σ [McMillan, 1992].

If $\ll$ is a well-founded order and satisfies certain conditions (see Def. 3 in Bonet et al. [2014]), then the number of non-cut-off events in Unf_Σ is upper-bounded by the number of reachable markings in Σ. We will consider such $\ll$ throughout. We define the size of Unf_Σ as $|\mathit{Unf}_\Sigma| := |B|$.

A planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ can be encoded as a Petri net $\Sigma(\Pi) = \langle \langle P, T, F \rangle, M_0 \rangle$. Facts are encoded as places. Actions $a \in \mathcal{A}$ are encoded as transitions $t \in T$ with $\mathsf{pre}(t) = \mathsf{pre}(a)$ and $\mathsf{eff}(t) = \mathsf{eff}(a)$, adding redundant effects $\mathsf{eff}(a)[v] = \mathsf{pre}(a)[v]$ for prevail conditions. In this section, we do not consider actions with effect-only variables v, i. e., $v \in \mathsf{vars}(\mathsf{eff}(a))$ but $v \notin \mathsf{vars}(\mathsf{pre}(a))$. Such actions can cause an exponential blow-up in the Petri net itself, as mentioned in Chapter 7.2. We assume this encoding throughout, and refer to its unfolding as the *unfolding of* Π, denoted Unf_Π. We identify facts with places, actions with transitions, and (partial) states with markings.

Compatibility of Orders

We next define the compatibility of orders formally. To do so, we need the notion of a center sub-configuration. Given a configuration $C = \{e_1, \dots, e_n\}$ and a factoring $\mathcal{F}$, by $C^C := \{e_i \mid e_i \in C \land \phi(e_i) \in \mathcal{A}^C\}$ we denote the sub-configuration of C that consists of center events only. We say that a center path π^C *extends* C^C if there exists a linearization of C^C that is a sub-sequence of π^C.

Throughout this section, whenever we require compatible search orders, we assume strict total orders $\ll$ for both decoupled search and unfolding. This guarantees that, in every step in the search, there is a unique action/event that is selected. If this is not the case, e. g., there is no strict order between two possible center extensions, both techniques are exponentially separated from each other as shown in Theorem 16 and 17.

A search order $\ll_U$ for unfolding is a strict total order over local configurations $[e]$. At each point in the unfolding process, the $\ll_U$-minimal possible event e is added. A search order $\ll_D$ for decoupled search is a strict total order over center paths, where each step considers the $\ll_D$-minimal possible expansion. A pair of search orders $(\ll_U, \ll_D)$

is *compatible* if *(O1)* $\ll_U$ always orders new leaf events before new center events, and *(O2)* $\ll_D$ and $\ll_U$ agree on center paths in the sense that a $\ll$-minimal event in the unfolding corresponds to $\ll_D$-minimal center paths adding the corresponding center action to the decoupled state space.

Formally, we have (O1) if for all pairs of possible extensions (e_1, e_2) and their local configurations $([e_1], [e_2])$, where $\phi(e_1) \in \mathcal{A}^{\mathcal{L}}_{\not{C}}$ and $\phi(e_2) \in \mathcal{A}^C$, it holds that $[e_1] \ll_U [e_2]$. To formalize (O2), say that the $\ll$-minimal possible event e is a center event, i. e., $\phi(e) = a^C \in \mathcal{A}^C$. Consider the center paths π^C which extend $[e]^C \setminus \{e\}$, and define the set $\mathcal{P}$ as all possible expansions in the decoupled state space that have the form $\pi^C \circ \langle a^C \rangle$. We require that, when ignoring ordering relations inside $\mathcal{P}$, all elements of $\mathcal{P}$ are $\ll$-minimal among the possible expansions.

Regarding (O1), in case we deal with planning tasks in +P, we remark that a leaf event e, where $\phi(e) \in \mathcal{A}^{\mathcal{L}}_{\not{C}}$, might have an effect on the center. Such effects result from prevail conditions of e on the center, that require a redundant effect that adds back a token in the Petri net encoding. We still consider such events as leaf events.

A.3.2 Proofs

We give the full proofs of our theorems, covering first the separation results, then the domination results.

Separation Theorems

Theorem 14. *There exists a family of tasks Π^n in +P, with factorings in -M and search orders in -O, where $|\Theta^{\mathcal{RF}}_{\Pi^n}|$ is polynomial in n while $|Unf_{\Pi^n}|$ is exponential in n.*

Proof. One family as claimed is our illustrative running example, $\Pi^n = \langle \mathcal{V}^n, \mathcal{A}^n, \mathsf{cost}^n, \mathcal{I}^n, \mathcal{G}^n \rangle$ defined as follows. $\mathcal{V}^n = \{T, p_1, \dots, p_n\}$ where $\mathcal{D}(T) = \{l, r\}$ and $\mathcal{D}(p_i) = \{l, r, T\}$. The initial state is $\mathcal{I}^n = \{T = l, p_1 = l, \dots, p_n = l\}$. The goal does not matter here. The actions are $\mathcal{A}^n = \{drive(x, y) \mid (x, y) \in \{(l, r), (r, l)\}\} \cup \{load(i, z), unload(i, z) \mid 1 \leq i \leq n, z \in \{l, r\}\}$ where $\mathsf{pre}(drive(x, y)) = \{T = x\}$, $\mathsf{eff}(drive(x, y)) = \{T = y\}$, $\mathsf{pre}(load(i, z)) = \{T = z, p_i = z\}$, $\mathsf{eff}(load(i, z)) = \{p_i = T\}$, and $\mathsf{pre}(unload(i, z)) = \{T = z, p_i = T\}$, $\mathsf{eff}(unload(i, z)) = \{p_i = z\}$.

Assume the factoring $\mathcal{F}$ with center $C = \{T\}$ and leaves $\mathcal{L} = \{\{p_1\}, \dots, \{p_n\}\}$. The number of decoupled states is $\#\Theta^{\mathcal{RF}}_{\Pi^n} = 3$: after applying $drive(l, r)$ and $drive(r, l)$, all leaf states are reached. $\Theta^{\mathcal{F}}_{\Pi}$ contains $|\Theta^{\mathcal{F}}_{\Pi}| = 3 + 2n + 3n + 3n = 8n + 3$ factor states.

The size of the unfolding prefix $|Unf_{\Sigma}|$, however, is exponential in n. Any $load(i, l)$ event that fires in the initial state consumes an instance of the condition $(T = l)$, and produces a new instance of that condition. As the consumed instance can be any instance produced beforehand, the number of instances in the Petri net doubles in each step. □

Theorem 15. *There exists a family of tasks Π^n in -P, with factorings in +M and search orders in -O, where $\#\Theta^{\mathcal{RF}}_{\Pi^n}$ is exponential in n while $|Unf_{\Pi^n}|$ is polynomial in n.*

We prove the following stronger claim:

Lemma 21. *There exists a family of tasks Π^n in -P, with factorings in +M and search orders in -O, where $\#\Theta^{\mathcal{RF}}_{\Pi^n}$ is exponential in n for every family of factorings $\mathcal{F}^n$, while $|Unf_{\Pi^n}|$ is polynomial in n.*

Proof. Consider $\Pi^n = \langle \mathcal{V}^n, \mathcal{A}^n, \mathsf{cost}^n, \mathcal{I}^n, \mathcal{G}^n \rangle$ as follows. $\mathcal{V}^n = \{v_1, \dots, v_n\}$, where $\mathcal{D}(v_i) = \{0, 1, 2\}$ for $1 \leq i \leq n$. The initial state is $\mathcal{I}^n = \{v_1 = 0, \dots, v_n = 0\}$. The actions are $\mathcal{A}^n = \{a_0\} \cup \{a^{12}_i \mid 1 \leq i \leq n\} \cup \{a^{12}_{ij} \mid 1 \leq i, j \leq n\}$ where $\mathsf{pre}(a_0) = \{v_1 = 0, \dots, v_n = 0\}$ and $\mathsf{eff}(a_0) = \{v_1 = 1, \dots, v_n = 1\}$; $\mathsf{pre}(a^{12}_i) = \{v_i = 1\}$ and $\mathsf{eff}(a^{12}_i) = \{v_i = 2\}$; $\mathsf{pre}(a^{12}_{ij}) = \{v_i = 0, v_j = 1\}$ and $\mathsf{eff}(a^{12}_{ij}) = \{v_i = 2, v_j = 2\}$.

The unfolding prefix Unf_Σ has size $|Unf_\Sigma| = 3n$, with a single condition b for every reachable fact. $\#\Theta^{\mathcal{RF}}_{\Pi^n}$ is exponential in n as claimed. Observe that the a^{12}_{ij} actions have an unreachable precondition, yet their presence means that, in any star factoring, there can be at most one leaf: if there were two leaves F_i and F_j containing v_i and v_j respectively, then the action a^{12}_{ij} would incur a direct dependency across F_i and F_j, in contradiction. Thus, for any family $\mathcal{F}^n = \{C^n, L^n\}$ of star factorings (where L^n may not be present for some values of n), $\max(|C^n|, |L^n|) \in \Omega(n)$. So $\#\Theta^{\mathcal{RF}}_{\Pi^n}$ is exponential in n since it has to enumerate all applications of a^{12}_i actions for a linear number of variables v_i. $\square$

Theorem 16. *There exists a family of tasks Π^n in -P, with factorings in -M and search orders in +O, where $|\Theta^{\mathcal{RF}}_{\Pi^n}|$ is polynomial in n while $|Unf_{\Pi^n}|$ is exponential in n.*

Proof. We construct a task family $\Pi^n = \langle \mathcal{V}^n, \mathcal{A}^n, \mathsf{cost}^n, \mathcal{I}^n, \mathcal{G}^n \rangle$ as follows. The variables are $\mathcal{V}^n = \{c, l_1, \dots, l_n\}$, where $\mathcal{D}(c) = \{0, 1\}$ and $\mathcal{D}(l_i) = \{0, 1, 2\}$. The initial state is $\mathcal{I}^n = \{c = 0, l_1 = 0, \dots, l_n = 0\}$. The actions are $\mathcal{A}^n = \{a^C_{01all2}, a^C_{10}\} \cup \{a^C_{01i01}, a^L_{i20}, a^L_{i21} \mid 1 \leq i \leq n\}$. The action preconditions and effects are: $\mathsf{pre}(a^C_{01all2}) = \{c = 0, l_1 = 0, \dots, l_n = 0\}$ and $\mathsf{eff}(a^C_{01all2}) = \{c = 1, l_1 = 2, \dots, l_n = 2\}$; $\mathsf{pre}(a^C_{10}) = \{c = 1\}$ and $\mathsf{eff}(a^C_{10}) = \{c = 0\}$; $\mathsf{pre}(a^C_{01i01}) = \{c = 0, l_i = 0\}$ and $\mathsf{eff}(a^C_{01i01}) = \{c = 1, l_i = 1\}$; $\mathsf{pre}(a^L_{i20}) = \{l_i = 2\}$ and $\mathsf{eff}(a^L_{i20}) = \{l_i = 0\}$; $\mathsf{pre}(a^L_{i21}) = \{l_i = 2\}$ and $\mathsf{eff}(a^L_{i21}) = \{l_i = 1\}$.

Assume the factoring $\mathcal{F}$ with center $C = \{c\}$ and leaves $\mathcal{L} = \{\{l_1\}, \dots, \{l_n\}\}$. After applying a^C_{01all2}, exploration of the leaf actions a^L_{i20} and a^L_{i21} reaches all variable values and thus a compact representation of reachability. We construct the search orders $\ll$ so that decoupled search finds this compact representation, but unfolding does not.

We postpone configurations containing leaf events until no more center-only configurations are available (thus violating constraint (O1) of compatible orders); and we constrain the order on center actions to start with the sequence $\langle a^C_{01all2}, a^C_{10} \rangle$. Precisely: if C_l contains an event e labeled by $\phi(e) = a \in \mathcal{A}^{\mathcal{L}}_{\not{C}}$, but C does not contain such an event, then $C \ll C_l$; denoting $C_1 = \{a^C_{01all2}\}$ and $C_2 = \{a^C_{01all2}, a^C_{10}\}$, we set $C_1 \ll C_2 \ll C \in \mathit{Unf}_\Sigma \setminus \{C_1, C_2\}$. Inside these constraints, $\ll$ can be arbitrary.

With this search order, decoupled search first generates $s^\mathcal{F} = \mathcal{I}^\mathcal{F}[\![a^C_{01all2}]\!]$, where application of the leaf actions a^L_{i20} and a^L_{i21} reaches all values of the leaf variables. Then decoupled search generates $t^\mathcal{F} = s^\mathcal{F}[\![a^C_{10}]\!]$. After that, the process stops: $s^\mathcal{F}$ covers everything with center state $c = 1$, $t^\mathcal{F}$ covers everything with center state $c = 0$. The decoupled state space has $\#\Theta^{\mathcal{RF}}_{\Pi^n} = 3$ states, and thus polynomial size.

The unfolding prefix Unf_Σ, however, has size exponential in n. The unfolding starts with the center events a^C_{01all2} and a^C_{10}. Thereafter, given $\ll$, it prefers to explore the center events a^C_{01i01} rather than the leaf events a^L_{i2x}. The unfolding thus has to set each leaf variable separately to 1, using a^C_{01i01}. Every step a^C_{01i01} sets c to 1, and must be followed by a^C_{10} setting c back to 0. In doing so, a^C_{01i01} consumes an instance of the condition $c = 0$, and a^C_{10} generates a new instance of that condition. As the consumed instance can be any instance produced beforehand, the number of instances in the Petri net doubles in each step. □

Theorem 17. *There exists a family of tasks Π^n in -P, with factorings in -M and search orders in +O, where $\#\Theta^{\mathcal{RF}}_{\Pi^n}$ is exponential in n while $|\mathit{Unf}_{\Pi^n}|$ is polynomial in n.*

Proof. We adapt the task Π^n used in the proof of Theorem 16. We add a new variable l with domain $\{0, 1\}$ and initial value 0. We include a new action a^L_{01} with precondition $\{l = 0\}$ and effect $\{l = 1\}$. We add the fact $l = 1$ into the preconditions of all actions a^C_{01i01}, and we add $l = 0$ into the effects of these actions. In this modified task, to enter the exponential part of the search space, the leaf action a^L_{01} must be applied first. Decoupled search always applies leaf actions first. If we violate (O1) however, unfolding can avoid this.

Precisely, we constrain $\ll$ to order configurations containing (an event labeled) a^C_{01all2} behind all configurations containing any of a^C_{01i01}; and to order configurations containing a^L_{01} behind all other configurations. Decoupled search then expands the leaf action a^L_{01} at $\mathcal{I}^\mathcal{F}$, enabling the a^C_{01i01} actions, thus forcing the search into exploring the search sub-space using these actions. This sub-space contains a different decoupled state for every subset of leaf states so is exponentially large. Yet unfolding prefers to do anything other than adding a^L_{01}, so initially adds a^C_{01all2} and then expands the a^L_{i20} and a^L_{i21} actions, which together with a single a^C_{10} event and a single a^L_{01} event represent all reachable markings. □

Domination Theorems

We first analyze the case of singleton components, then that where there are no prevail conditions. For the first case, we show that if the center factor is singleton, then no two events that affect the center are concurrent. This directly translates to paths and configurations that affect the center, so there is a direct correspondence between center paths and center configurations. Given this property, unfolding cannot combine reached conditions across decoupled states, so it exploits the concurrency across leaves as decoupled search. For the case without prevail conditions, we introduce the concept of *compatible steps*, showing that with compatible orders, in each such step the same set of states is reached by unfolding and decoupled search.

By $\hat{a}$ we denote an *occurrence* of an action a in $\Theta_{\Pi}^{\mathcal{RF}}$, i. e., center action $a^C \in \mathcal{A}^C$ inducing a new decoupled state $s^{\mathcal{F}}$, or a leaf action $a^L \in \mathcal{A}^{\mathcal{L}}$ inducing a leaf state in a decoupled state $s^{\mathcal{F}}$. By $\hat{p}$ we denote an occurrence of a factor state p, i. e., a center state or a reached leaf state in a decoupled state.

Theorem 18. *For Π in +P, factorings with singleton center factor (-M), and search orders in -O, with hypercube pruning $\#\Theta_{\Pi}^{\mathcal{RF}} \leq |Unf_{\Pi}|$.*

The proof shows how to embed $\Theta_{\Pi}^{\mathcal{RF}}$ into Unf_{Π}. Theorem 18 follows directly from the following two Lemmas.

Lemma 22. *Let Π be a planning task and $\mathcal{F}$ a factoring with singleton center factor $|C| = 1$. Let Unf_{Π} be the unfolding of Π. Then every pair of distinct events $e_1 \neq e_2$ affecting the center factor C is not concurrent, i. e., $e_1 \not\parallel e_2$ if $\mathsf{vars}(\phi(\mathsf{eff}(e_1))) \cap C \neq \emptyset$ and $\mathsf{vars}(\phi(\mathsf{eff}(e_2))) \cap C \neq \emptyset$.*

Proof. Note first that every transition with an effect on a variable v necessarily has a precondition on v, too, i. e., $F \cap \mathsf{vars}(\mathsf{pre}(t)) = F \cap \mathsf{vars}(\mathsf{eff}(t))$. Correspondingly for events in Unf_{Π}. Observe further that for every variable $v \in \mathcal{V}$, we have $c_0 \prec e$, where $\phi(c_0) = \mathcal{I}[v]$, for all events e where $v \in \mathsf{vars}(\phi(\mathsf{pre}(e)))$ (conditions that encode assignments to v must result from the initial assignment to v).

Now say that two events $e_1 \neq e_2$ in Unf_{Π} both affect the center factor C, i. e., $\mathsf{vars}(\phi(\mathsf{eff}(e_1))) \cap C \neq \emptyset$ and $\mathsf{vars}(\phi(\mathsf{eff}(e_2))) \cap C \neq \emptyset$. As C is singleton component by prerequisite, this means that $\mathsf{vars}(\phi(\mathsf{eff}(e_1))) \cap \mathsf{vars}(\phi(\mathsf{eff}(e_2))) \supseteq \{v\}$ where $C = \{v\}$. Therefore, with the above, we have that either (i) there exist $e_1' \neq e_2'$, where $e_1' \preceq e_1$, $e_2' \preceq e_2$, with $\mathsf{pre}(e_1') \cap \mathsf{pre}(e_2') \supseteq \{c_0\} \neq \emptyset$, so e_1 and e_2 are in conflict; or (ii) $e_1 \prec e_2$ or $e_2 \prec e_1$, so one is a causal predecessor of the other. $\square$

Lemma 23. *Let Π be a task in +P, and $\mathcal{F}$ a factoring with singleton center factor $C \in \mathcal{F}$. Let $\Theta_{\Pi}^{\mathcal{RF}}$ and Unf_{Π} be generated using compatible orders $\ll$, with hypercube pruning for $\Theta_{\Pi}^{\mathcal{RF}}$. Then decoupled states in $\Theta_{\Pi}^{\mathcal{RF}}$ can be injectively mapped to non-cut-off events in Unf_{Π}.*

Proof. Given the definition of compatible orders, we can view $\Theta_{\Pi}^{\mathcal{RF}}$ and Unf_{Π} as being built by, iteratively, (1) adding a new center event e to Unf_{Π} and adding corresponding possible expansions in $\Theta_{\Pi}^{\mathcal{RF}}$, at all center paths that extend $[e]^C \setminus \{e\}$; and (2) adding all possible leaf events to Unf_{Π} in any order.

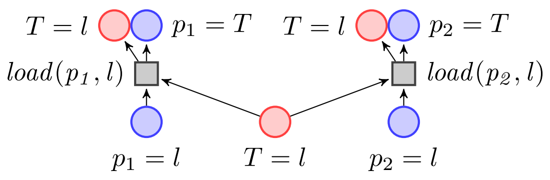


Figure A.1: An incomplete prefix of the unfolding of the logistics example.

Observe that, in (1), always at most one[2] decoupled state is added to $\Theta_\Pi^{\mathcal{RF}}$. This is because, by Lemma 22, different configurations that affect the center factor are not concurrent. In particular, for a center event e, $[e]^C \setminus \{e\}$ is totally ordered (it is a path through the state space of the single center variable). The only center paths π^C that extend $[e]^C \setminus \{e\}$ are therefore ones that append some postfix π^C_{post} to $[e]^C \setminus \{e\}$. By construction, as $\Theta_\Pi^{\mathcal{RF}}$ and *Unf*$_\Pi$ are built iteratively from (1) and (2), π^C_{post} must be empty so the only possible π^C is $[e]^C \setminus \{e\}$ itself.

Given this, we can map the decoupled state at the new center action (if one is added) to e. The mapping is injective as e is a new event each time (1) is applied.

Let now $s^{\mathcal{F}}$ in $\Theta_\Pi^{\mathcal{RF}}$ be arbitrary. Denote by D the prefix of $\Theta_\Pi^{\mathcal{RF}}$ generated prior to $s^{\mathcal{F}}$, and by U the corresponding prefix of *Unf*$_\Pi$. As $s^{\mathcal{F}}$ is not pruned by hypercube pruning, there exists $s \in [s^{\mathcal{F}}]$ not contained in $[t^{\mathcal{F}}]$ for any $t^{\mathcal{F}}$ in D. Let $\hat{a}$ be an action occurrence in $s^{\mathcal{F}}$ that generates s, i. e., either the center action application leading to $s^{\mathcal{F}}$ or a leaf action application setting a leaf L to $s[L]$. Then the corresponding event e is a non-cut-off event in *Unf*$_\Pi$, because U cannot contain reachable markings (states) not contained in D. The latter is true because, first, component states in D are generated in the same order as the corresponding conditions in U; and second, with Lemma 22 the conditions in U cannot be combined into new co-sets across decoupled states, because the center sub-configuration leading to a condition that is part of a decoupled state $t^{\mathcal{F}}$ is not concurrent with the center configuration of $s^{\mathcal{F}}$. □

Example 13. *Consider our logistics example with $n = 2$ packages. An incomplete prefix is shown in Figure A.1, where all leaf events that are possible in the initial marking have fired, namely $load(p_1, l)$ and $load(p_2, l)$. In this prefix, there are three center events possible that drive the truck from l to r, consuming one of the $T = l$ conditions. In the decoupled state space, however, these three events are encoded by a single center path $\langle drive(r) \rangle$. Thus, only for the first $drive(r)$ event a decoupled state is added, the remaining two events do not lead to a change in the decoupled state space.*

Theorem 19. *For Π in -P, factorings in +M, and search orders in -O, $|Unf_\Pi| \leq |\Theta_\Pi^{\mathcal{RF}}|$.*

We will first illustrate the advantage of unfolding on planning tasks with non-singleton center component in Example 14, then introduce compatible steps as a means to capture

[2]See Example 13 for a case where no state is added.

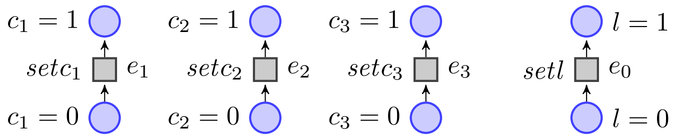


Figure A.2: The complete unfolding prefix U_3 of the planning task from Example 14 and Example 15 for $n = 3$.

the set of new states reached by an expansion step given compatible orders. Finally, we give the full details of the proof of Theorem 19. The proof shows how to surjectively map factor states in $\Theta_\Pi^{\mathcal{RF}}$ to *factor co-sets* in *Unf*$_\Pi$, showing that the number of factor states is at least as high as that of factor co-sets. The analysis is decomposed into two steps, first showing a correspondence across hypothetical *non-pruned* infinite structures, then showing that this correspondence persists in the actual structures. The non-pruned *Unf*$_\Pi$ expands cut-off events. The non-pruned $\Theta_\Pi^{\mathcal{RF}}$ does not prune decoupled states, and within each decoupled state does not do duplicate checking across leaf-factor states. Note that these structures can be built incrementally by choosing applicable center and leaf expansions non-deterministically.

Example 14. *Consider the following family of planning tasks Π^n in -P: $\Pi^n = \langle \mathcal{V}^n, \mathcal{A}^n, \mathsf{cost}^n, \mathcal{I}^n, \mathcal{G}^n \rangle$, where $\mathcal{V}^n = \{c_1, \dots, c_n, l\}$, with $\mathcal{D}(v) = \{0, 1\}$ for all $v \in \mathcal{V}^n$; and initially all variables have value $\mathcal{I}^n[v] = 0$. There are $2n{+}1$ actions $\mathcal{A}^n = \{setc_1, \dots setc_n, resetc_1, \dots, resetc_n, setl\}$, where $\mathsf{pre}(setc_i) = \{c_i{=}0\}$, $\mathsf{eff}(setc_i) = \{c_i{=}1\}$, $\mathsf{pre}(resetc_i) = \{c_i{=}1, l{=}1\}$, $\mathsf{eff}(resetc_i) = \{c_i{=}0, l{=}0\}$, and $\mathsf{pre}(setl) = \{l{=}0\}$, $\mathsf{eff}(setl) = \{l{=}1\}$.*

A complete prefix of the unfolding of Unf$_{\Pi^n}$ with $n = 3$ is shown in Figure A.2. Only the events $setc_i$ and $setl$ are non-cut-offs, all events corresponding to $resetc_i$ actions are cut-offs. This is because all $set{*}$ transitions are concurrent, so any subset of these events in Unf$_{\Pi^n}$ is a configuration, that represents a state of Π^n. Obviously, all states of Π can be represented using the respective actions.

In decoupled search, with the factoring $\mathcal{F} = \{C = \{c_1, \dots, c_n\}, L = \{l\}\}$ in +M, all interleavings of the center actions $setc_i$ are enumerated, leading to a total of 2^n decoupled states with different center state, so none of them can be pruned. This behavior occurs whenever a subset of the actions that affects a factor $F \in \mathcal{F}$ is concurrent. Then the representation size of the component state space of F, i. e., the number of conditions vs. the number of component states, can be exponentially smaller in the unfolding.

As a last new concept, we introduce the notion of *compatible steps* to capture all new events and action occurrences induced by a center event, when constructing *Unf*$_\Pi$ and $\Theta_\Pi^{\mathcal{RF}}$, for Π in -P, with compatible orders $\ll_U$ and $\ll_D$. In an unfolding prefix U, a compatible step consists of all events caused by adding the $\ll_U$-minimal center event e that can fire in U and the following leaf events. Formally, a compatible step is the set of

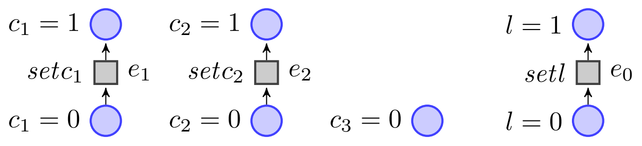


Figure A.3: The prefix U_2 of the unfolding of the task from Example 15 for $n = 3$.

events $\mathcal{CS}_U^e := \{e\} \cup \{e' \mid \phi(e') \in \mathcal{A}_{\not\mathcal{C}}^{\mathcal{L}} \wedge [e]^C = [e']^C \wedge e \prec e'\}$ that contains e and all leaf events e' enabled by e.

In a decoupled state-space prefix D, this corresponds to appending $a^C = \phi(e)$ to all center paths π^C that extend $[e]^C \setminus \{e\}$, generating new decoupled states $s^{\mathcal{F}}$. Let $\mathcal{P}$ be the set of all possible expansions in D that have the form $\pi^C \circ \langle a^C \rangle$. A compatible step of a center event e in a decoupled state-space prefix D is the set of center action occurrences $\mathcal{CS}_D^e := \{\hat{a}^C \mid \pi^C \circ \langle a^C \rangle \in \mathcal{P}\}$ that includes all possible $\ll_D$-minimal $\hat{a}^C$ inducing decoupled states $s^{\mathcal{F}}$ reached on center paths $\pi^C \circ \langle a^C \rangle$ where π^C extends $[e]^C \setminus \{e\}$.

Example 15. *Figures A.2, A.3, and A.4 show two prefixes U_2, U_3 of the unfolding, respectively D_2, D_3 of the decoupled state space, of Π^n from Example 14 for $n = 3$. For illustration purposes, we only show the center state of each decoupled state in Figure A.4, denoting, e. g., the state $\{c_1{=}0, c_2{=}1, c_3{=}0\}$ by* 010. *In all decoupled states, the leaf states $l = 0$ and $l = 1$ are reached.*

In the initial prefixes U_0 and D_0, there is only a single event e_0, respectively action occurrence $\hat{a}_0$, for the leaf action setl. The prefix D_2 (left) corresponds to the state space after performing the two compatible steps $\mathcal{CS}_{D_0}^{setc_1}$ and $\mathcal{CS}_{D_1}^{setc_2}$. In the unfolding this only generates the events e_1 and e_2 (see Figure A.3), where $\phi(e_i) = setc_i$. The two compatible steps generate three decoupled states, namely the states 100, 010, *and* 110 *that can be reached using interleavings of the actions $setc_1 = \phi(e_1)$ and $setc_2 = \phi(e_2)$. In D_2, note that $setc_2$ is appended to all center paths that extend $[e_2] \setminus \{e_2\} = \emptyset$. This holds for all center paths, so $setc_2$ is appended to $\langle setc_1 \rangle$ and to the empty center path.*

Say the prefixes D_2 and U_2 are extended by $\mathcal{CS}^{setc_3}$. Then, in U_3, this only generates an event e_3 for $setc_3$, with all events in U_3 being concurrent. In D_3, four new decoupled states are generated, as highlighted in red in Figure A.4 (right). In $\mathcal{CS}_{D_2}^{setc_3}$, $setc_3$ is appended to all center paths that extend the center sub-configuration $[e_3]^C \setminus \{e_3\} = \emptyset$, which has no ordering constraints. Every *center path (including the empty path) extends $[e_3]^C$, so $setc_3$ is appended to all of them.*

Compatible steps capture exactly the behavior of search expansions in unfolding and decoupled search when following a compatible order. A compatible step corresponds to adding the $\ll_U$-minimal possible center event e in the unfolding and is composed of e and all leaf events it enables. In decoupled search, e corresponds to a set of center action occurrences $\hat{a}^C$ that generate new decoupled states $s^{\mathcal{F}}$ by appending $a^C = \phi(e)$ to all center paths π^C where π^C extends $[e]^C \setminus \{e\}$.

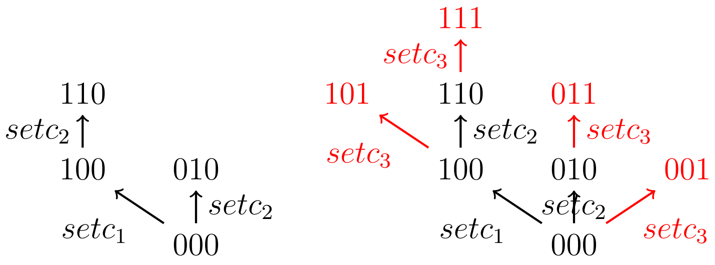


Figure A.4: Two prefixes, D_2 (left) and D_3 of the center state space of the task from Example 15 for $n = 3$.

We next prove that, when incrementally building an unfolding prefix U and a decoupled state-space prefix D with compatible orders, when expanding one compatible step after the other in both unfolding and decoupled search, after every step i if a state s is reachable in D_i, then it is also reachable in U_i.

Lemma 24. *Let Π be a task in -P, and $\mathcal{F}$ a factoring in +M. Let U_i and D_i be the prefixes of Unf$_\Pi$ and $\Theta_\Pi^{\mathcal{RF}}$, respectively, generated by i iterated compatible steps. Then all states reached in D_i are also reached in U_i.*

Proof. We show that, after every compatible step $\mathcal{CS}^e_{i-1}$ of a center event e, a state s is reached in U_i if it is reached in D_i. The proof goes by induction over the number of compatible steps i. For the base case $i = 0$, the claim holds trivially, since in both U_0 and D_0 exactly those states are reached that are reachable with leaf-only actions from the initial state.

For the inductive case, assume that the new state s is reached in D_i in a decoupled state $s^{\mathcal{F}}$ by an action occurrence $\hat{a}'$ of the compatible step $\mathcal{CS}^e_{D_{i-1}}$. By definition, $\pi^C(s^{\mathcal{F}})$ is composed of $\pi^C \circ \langle a^C \rangle$, where $a^C = \phi(e)$ and π^C extends $[e]^C \setminus \{e\}$. Denote by $t^{\mathcal{F}}$ the predecessor of $s^{\mathcal{F}}$ in D_{i-1}, so $\pi^C(t^{\mathcal{F}})$ extends $[e]^C \setminus \{e\}$ and $\pi^C(t^{\mathcal{F}}) \circ \langle a^C \rangle = \pi^C(s^{\mathcal{F}})$. By induction hypothesis, all states $s' \in [t^{\mathcal{F}}]$ in D_{i-1} are also reached by a configuration C' in U_{i-1}. Let C be the configuration in U_{i-1} that reaches the last predecessor t of s in $t^{\mathcal{F}}$.

If the occurrence $\hat{a}'$ that generates s in $s^{\mathcal{F}}$ corresponds to a^C, then the configuration $C \cup \{e\}$ reaches s in U_i. Otherwise, assume $\hat{a}'$ corresponds to a leaf-only action $a^L \in \mathcal{A}^L_{\not{C}}$ of leaf L. Denote by t_0 the state $t[\![a^C]\!]$ and by C_0 the configuration $C \cup \{e\}$, both reached in D_i, respectively U_i. Denote by π^L the sequence of L-only actions within $s^{\mathcal{F}}$ behind a^C on the path to s, and by e^L_k the corresponding events of the actions a^L_k, so $\phi(e^L_k) = a^L_k$ for all $1 \leq k \leq n$. Finally, denote by $t_1, \dots, t_n$ the states $t_0[\![a^L_1]\!], \dots, t_0[\![\langle a^L_1, \dots, a^L_n \rangle]\!]$, where $t_n = s$. We prove by induction that if the state t_j is reached in D_i, then there exists a corresponding co-set Q_j, where $\phi(Q_j) = t_j$, in U_i.

For the base case, remember that t_0 is reached in D_i, and that C_0 reaches t in U_i. Denote by Q_0 the co-set at the end of C_0, where $\phi(Q_0) = t_0$. Then, since a^L_1 is applicable in t_0, because π^L is a valid sub-sequence on the path to s, reaching t_1, we get that e_1^L can fire in Q_0 reaching $Q_1 = (Q_0 \setminus \mathsf{pre}(e_1^L)) \cup \mathsf{eff}(e_1^L)$, where $\phi(Q_1) = t_1$. Observe that, in case e_1^L is concurrent to e, then it already exists in U_{i-1}, and no new event for a_1^L is added in U_i. In any case, $C_0 \cup \{e_1^L\}$ is a configuration in U_i that reaches t_1.

For the inductive case, let a_j^L be the action that is applied in t_{j-1}. By hypothesis, the co-set Q_{j-1} is reached in U_i by configuration $C_{j-1} = C_0 \cup \{e_1^L, \ldots, e_{j-1}^L\}$. Again, since π^L is a valid path, so a_j^L is applicable in t_{j-1}, e_j^L can fire in Q_{j-1} reaching Q_j. As before, if e_j^L is concurrent to e and all leaf events e_k^L for $k < j$, then it already exists in U_{i-1} and no new event is added. We get that $C_j = C_{j-1} \cup \{e_j^L\}$ is a configuration in U_i that reaches t_j.

If s is generated by leaf action occurrences of multiple leaves in $s^{\mathcal{F}}$, we can apply the above reasoning to every leaf separately. We can construct the overall configuration C_n by adding the leaf events of all involved leaves, since, without prevail conditions, the events affecting different leaves (and not the center) are concurrent. $\square$

We use the following notations for the proof of Theorem 19. A factor co-set P is a co-set where $\mathsf{vars}(\phi(P)) = F$ for some $F \in \mathcal{F}$. We write $P[F]$ to indicate the factor F concerned, and given an arbitrary co-set Q we write $Q[F]$ for the restriction of Q to conditions over the variables F. We write $[Q]$ for the configuration supporting Q. We write $p[F]$ to indicate that a factor state p is over factor F.

Proof. The proof has two parts: first, we consider the non-pruned $\Theta_{\Pi}^{\mathcal{RF}}$ and Unf_{Π}; then we analyze cut-off events vs. hypercube pruning.

For the first part, we prove that *(*) there is a surjective mapping g where (a) for every $\hat{p}$, $\phi(g(\hat{p})) = p$; (b) for every co-set Q, there is at least one $s^{\mathcal{F}}$ where $\pi^C(s^{\mathcal{F}})$ extends $[Q]^C$; and (c) for every such $s^{\mathcal{F}}$ and every F, there is $\hat{p}[F]$ in $s^{\mathcal{F}}$ where $g(\hat{p}[F]) \supseteq Q[F]$.* Note that in (c), we do not have $g(\hat{p}[F]) = Q[F]$, because Q may instantiate only a subset of the variables of F.

We prove (*) by structural induction over an incremental construction of $\Theta_{\Pi}^{\mathcal{RF}}$ alongside the construction of Unf_{Π}. D and U denote the current prefix of $\Theta_{\Pi}^{\mathcal{RF}}$ and Unf_{Π} respectively, during the construction.

The induction base case is simple: U is then the set $Min(ON)$ of $\prec$-minimal elements of $B \cup E$. This contains exactly one condition b for every state variable v, with $\phi(b) = \mathcal{I}[v]$. The factor co-sets $P[F]$ here match exactly the factor states $\mathcal{I}[F]$ for $F \in \mathcal{F}$. We construct D as the non-expanded initial decoupled state $\mathcal{I}_0^{\mathcal{F}}$. Defining $g(\mathcal{I}[F]) := P[F]$, we obviously get (a) – (c).

For the inductive case, say that U' results from U by adding event e. We denote $a := \phi(e)$. By hypothesis, we have a mapping g from D to U satisfying (*). We show how to extend D and g to suitable D' and g' respectively.

We construct D' by, for every $s^{\mathcal{F}}$ where $\pi^C(s^{\mathcal{F}})$ extends $[\mathsf{pre}(e)]^C$, *extending $s^{\mathcal{F}}$ with a*, as follows. If a is a leaf action, then (i) we apply a to every factor state p in

$s^{\mathcal{F}}$ where $p \models \mathsf{pre}(a)$. If a is a center action and $s^{\mathcal{F}} \models \mathsf{pre}(a)$, then we apply a to $s^{\mathcal{F}}$, resulting in a new successor $t^{\mathcal{F}}$. In the latter, (ii) we add the updated center state; (iii) for every leaf factor L affected by a, and for every $s^L \in S^L$ where $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$ and $s^L \models \mathsf{pre}(a)[L]$, we add s^L updated with $\mathsf{eff}(a)[L]$; (iv) for every (leaf) factor L not affected by a, we add to $t^{\mathcal{F}}$ occurrences of actions $a^L \in \mathcal{A}^L_{\not{C}}$ reaching all of $s^L \in S^L$ where $\mathsf{prices}(s^{\mathcal{F}})[s^L] < \infty$. The latter is possible because, without prevail conditions, no such a^L has preconditions on the center.

Observe that this construction of D builds several decoupled states in a parallel manner, in difference to the actual construction of (pruned) $\Theta^{\mathcal{RF}}_{\Pi}$ during search. However, the construction of D complies with the unfolding search order.

Regarding the construction of g': For (i) – (iii), let $\hat{p'}[F]$ be a new factor state occurrence added to D' by an occurrence $\hat{a}$ of a, and let $\hat{p}[F]$ be the factor state occurrence that $\hat{a}$ is applied to. By hypothesis, $P[F] := g(\hat{p}[F])$ is a factor co-set and $P[F] \supseteq \mathsf{pre}(e)[F]$. Let $P'[F] := (P[F] \setminus \mathsf{pre}(e)[F]) \cup \mathsf{eff}(e)[F]$. Then $P'[F]$ is a co-set in U'. We set $g'(\hat{p'}[F]) := P'[F]$. For (iv), i. e., a factor state occurrence $\hat{p'}[L]$ of $p'[L] \in S^L$, where $\mathsf{prices}(s^{\mathcal{F}})[p'[L]] < \infty$, added to D', we define $g'(\hat{p'}[L]){:=}g(\hat{p}[L])$, where $\hat{p}[L]$ is $p'[L]$'s occurrence in $s^{\mathcal{F}}$.

We next show that g' has the desired properties (*) on D' and U'. Obviously, (a) is given by construction.

To see that g' is surjective, note that any new factor co-set $P'[F]$ in U' must result from a factor co-set $P[F]$ in U through $P'[F] := (P[F] \setminus \mathsf{pre}(e)[F]) \cup \mathsf{eff}(e)[F]$ where $\mathsf{eff}(e)[F] \neq \emptyset$ and thus $\mathsf{pre}(e)[F] \neq \emptyset$. Let $Q := P[F] \cup \mathsf{pre}(e)$. Then Q is a co-set in U as otherwise $P'[F]$ could not be a co-set in U'. By hypothesis (b), there is at least one $s^{\mathcal{F}}$ in D where $\pi^C(s^{\mathcal{F}})$ extends $[Q]^C$. By hypothesis (c), for every F there is $\hat{p}[F]$ in $s^{\mathcal{F}}$ where $g(\hat{p}[F]) \supseteq Q[F] = P[F]$, which implies with hypothesis (a) that $g(\hat{p}[F]) = P[F]$.

It remains to show that for every factor co-set $P'[F]$ there exists a factor state $\phi(P'[F])$ that is mapped to $P'[F]$ by g'. As $Q \supseteq \mathsf{pre}(e)$, we have that $\pi^C(s^{\mathcal{F}})$ extends $[\mathsf{pre}(e)]^C$. Thus $s^{\mathcal{F}}$ has been extended with $a = \phi(e)$. If a is a leaf action, then, because there are no prevail conditions and thus no Petri net outputs of e on the center, F must be the respective leaf factor L. We have $p[L] \models \mathsf{pre}(a)$, so a was applied to $p[L]$ by (i), generating the outcome state $\phi(P'[L])$ which is mapped by g' to $P'[F]$ as desired. If a is a center action, then, because there are no prevail conditions and thus no Petri net outputs of e on factors not affected by a, F must be either (ii) the center or (iii) a leaf factor L affected by a. In both cases, a was applied to $p[F]$, generating the outcome state $\phi(P'[F])$ which is mapped by g' to $P'[F]$ as desired.

Let now Q' be any new co-set in U'. We must show that (b) and (c) hold for Q'. Observe first that Q' must result from $Q := (Q' \setminus \mathsf{eff}(e)) \cup \mathsf{pre}(e)$ in U, and that Q is a co-set in U.

Regarding (b): By hypothesis (b), there is at least one $s^{\mathcal{F}}$ where $\pi^C(s^{\mathcal{F}})$ extends

$[Q]^C$. If a is a leaf action, there is nothing to show as, then, $[Q]^C = [Q']^C$. Say that a is a center action. By construction, $s^{\mathcal{F}}$ has been extended with a, producing a new successor $t^{\mathcal{F}}$. Clearly, $\pi^C(t^{\mathcal{F}})$ extends $[Q']^C$.

Regarding (c): Let $t^{\mathcal{F}}$ in D', where $\pi^C(t^{\mathcal{F}})$ extends $[Q']^C$, be arbitrary. We need to show that for every such $t^{\mathcal{F}}$ and every F, there is $\hat{p'}[F]$ in $t^{\mathcal{F}}$ where $g(\hat{p'}[F]) \supseteq Q'[F]$. First, say that a is a leaf action. Then D contains $s^{\mathcal{F}}$ with $\pi^C(s^{\mathcal{F}}) = \pi^C(t^{\mathcal{F}})$, namely the same decoupled state but yet with less leaf states. Let F be arbitrary. By hypothesis (c), there is $\hat{p}[F]$ in $s^{\mathcal{F}}$ where $g(\hat{p}[F]) \supseteq Q[F]$. Say that a affects L. If $F \neq L$, then, as there are no prevail conditions and thus no outputs of e on any factor other than L, $Q[F] = Q'[F]$ and we are done. Say that $F = L$. Then, as $Q \supseteq \mathsf{pre}(e)$, we have $p[L] \models \mathsf{pre}(a)$ so a was applied to $p[L]$ by (i). The outcome state $p'[L]$ is mapped by g' to a co-set $P'[L]$ in U', where $P'[L] \supseteq Q'[L]$ as needed.

Finally, say that a is a center action. Then $t^{\mathcal{F}}$ was generated by extending $s^{\mathcal{F}}$ in D with a. Let F be arbitrary. By hypothesis (c), there is $\hat{p}[F]$ in $s^{\mathcal{F}}$ where $g(\hat{p}[F]) \supseteq Q[F]$. If a affects F, then similar to the above we have $p[F] \models \mathsf{pre}(a)[F]$, so a was applied to $p[F]$ by either (ii) or (iii), and the outcome state $p'[F]$ in $t^{\mathcal{F}}$ is mapped by g' to a co-set $P'[F] \supseteq Q'[F]$ in U' as needed. If a does not affect F, then as above $Q[F] = Q'[F]$. In that case, due to construction (iv), $t^{\mathcal{F}}$ contains a new occurrence of $p[F]$, mapped by g' to $g(\hat{p}[F])$ which concludes the argument.

For the second part of the proof, consider now the pruned versions of $\Theta_{\Pi}^{\mathcal{R}\mathcal{F}}$ and Unf_{Π}, built using compatible orders $\ll$.

Assume that e is a non-cut-off event in Unf_{Π}. Consider the construction step where e is added, and denote D, D' and U, U' as above. Consider the decoupled states $s^{\mathcal{F}}$ extended with $a := \phi(e)$ in D' by the above construction. For every such $s^{\mathcal{F}}$, and for every factor F affected by a, there is a factor-state occurrence $\hat{p}[F]$ in $s^{\mathcal{F}}$ mapped to a factor co-set $P[F] := g(\hat{p}[F])$ where $P[F] \supseteq \mathsf{eff}(e)[F]$ and in particular $|P[F]| \geq |\mathsf{eff}(e)[F]|$.

Observe that, for any other event e', the factor-state occurrences $\hat{p'}[F]$ identified in the same manner must map to different factor co-sets $P'[F] \neq P[F]$, simply because every construction step of kinds (i) – (iii) maps to factor co-sets including newly generated conditions. Therefore, to prove the main claim it now suffices to show that at least one $s^{\mathcal{F}}$ as above is not pruned by hypercube pruning.

Observe that the iterative addition of conditions and factor states as per the construction above follows exactly the definition of compatible orders, where the leaf states within each decoupled state are added step-by-step. As each addition step for center events is exactly the addition of a compatible step, we can invoke Lemma 24.

As e is a non-cut-off event in Unf_{Π}, the state $s = \phi([e])$ is not reached in U. With Lemma 24, there cannot exist a decoupled state $s^{\mathcal{F}}$ in D where $s \in [s^{\mathcal{F}}]$. The decoupled states $t^{\mathcal{F}}$ where $\pi^C(t^{\mathcal{F}})$ extends $[e]^C$ are generated by the compatible step $\mathcal{CS}_D^{e'}$, where $e \in \mathcal{CS}_U^{e'}$. So at least one decoupled state exists in D' after $\mathcal{CS}_D^{e'}$ where $s \in [s^{\mathcal{F}}]$. Since

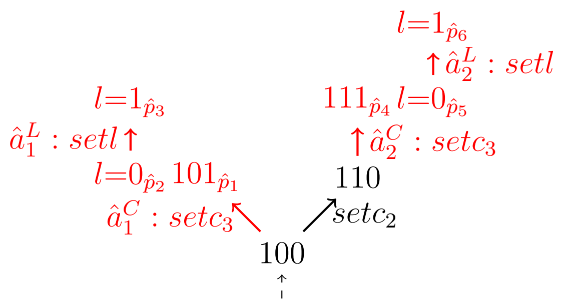


Figure A.5: Illustration of the surjective mapping g in the decoupled state space used in the proof of Theorem 19.

s is not reached in D, at least one of these states is not pruned by hypercube pruning, which is what we needed to prove. □

Example 16. *Consider again the construction from Example 15, where the event e_3 adds an instance of the $setc_3$ action to U. We illustrate the action and factor state occurrences that our construction adds to D in Figure A.5. For practical reasons, we only show two of the four newly generated center states, namely $\{c_1{=}1, c_2{=}0, c_3{=}1\}$ and $\{c_1{=}1, c_2{=}1, c_3{=}1\}$. The action occurrences that generate the other two decoupled states, and the mapping g' for the other factor state occurrences are analogous. The added center action occurrences in the figure are $\hat{a}^C_1$ and $\hat{a}^C_2$, generating the factor states $\hat{p}_1$ and $\hat{p}_4$. Denote by $c_{v=x}$ the condition in U_3 in Figure A.2 that corresponds to the fact $v = x$. Note that there is always only a single such condition for each fact. Then in the extended mapping g' we get $g'(\hat{p}_1) = \{c_{c_1=1}, c_{c_2=0}, c_{c_3=1}\}$, $g'(\hat{p}_4) = \{c_{c_1=1}, c_{c_2=1}, c_{c_3=1}\}$, $g'(\hat{p}_2) = g'(\hat{p}_5) = \{c_{l=0}\}$, and $g'(\hat{p}_3) = g'(\hat{p}_6) = \{c_{l=1}\}$. Analogously, g' is extended for the other two decoupled states generated when appending $setc_3$ to all center paths in D_2.*

A.4 Relation to Stubborn-Sets Pruning

A.4.1 Technical Background – Details

We give the background required for our proofs, loosely following the definitions of Sievers and Wehrle [2021]. First, we formally define the notion of necessary enabling sets, and action interference, which form the basis of stubborn sets pruning.

For a planning task Π and a state s of Π, by $\mathsf{app}(s)$ we denote the set of actions applicable in s, i. e., $\mathsf{app}(s) := \{a \in \mathcal{A} \mid s \models \mathsf{pre}(a)\}$.

Definition 63 (Necessary Enabling Set)**.** *Let Π be a planning task. For an action a and a state s such that $a \notin \mathsf{app}(s)$, a* necessary enabling set for a in s *is a set of* *actions $N \subseteq \mathcal{A}$ such that for every path $\langle a_1, \ldots, a_n \rangle$ applicable in s, if $a_i = a$ then $\{a_1, \ldots, a_{i-1}\} \cap N \neq \emptyset$.*

For a state s such that $s \not\models \mathcal{G}$, a necessary enabling set for $\mathcal{G}$ in s *is a set of actions $N \subseteq \mathcal{A}$ such that for every plan $\langle a_1, \ldots, a_n \rangle$ for s, $\{a_1, \ldots, a_n\} \cap N \neq \emptyset$.*

To formalize the notion of interference, we first define when an action enables or disables another, and action conflicts:

Definition 64 (Enabling, Disabling, Conflicting Actions)**.** *Let Π be a planning task, and let a_1, a_2 be actions, and s a state of Π, where $a_1 \in \mathsf{app}(s)$. Then:*

- a_1 disables a_2 in s *if $a_2 \in \mathsf{app}(s)$ and $a_2 \notin \mathsf{app}(s[\![a_1]\!])$.*
- a_1 disables a_2 on fact $\{v = p\}$ in s *if $a_2 \in \mathsf{app}(s)$, $\mathsf{pre}(a_2)[v] = p$, and $s[\![a_1]\!][v] \neq p$.*
- a_1 enables a_2 on fact $\{v = p\}$ in s *if $s[v] \neq p$, $\mathsf{pre}(a_2)[v] = p$, and $s[\![a_1]\!][v] = p$.*
- a_1 *and* a_2 conflict in s *if $a_2 \in \mathsf{app}(s)$, $a_1 \in \mathsf{app}(s[\![a_2]\!])$, $a_2 \in \mathsf{app}(s[\![a_1]\!])$, and $s[\![\langle a_1, a_2 \rangle]\!] \neq s[\![\langle a_2, a_1 \rangle]\!]$.*

An action *enables* another on a fact p in a state, if it achieves the missing precondition fact p. Analogously, an action *disables* another action (on a fact p) in a state if it makes a precondition (precondition p) false. Two actions conflict in a state if both are applicable in any order, but the outcome states differ.

We are now ready to define action interference:

Definition 65 (Action Interference)**.** *Let Π be a planning task, and let a_1, a_2 be actions, and s a state of Π, such that both actions are applicable in s. Then a_1* weakly interferes with a_2 in s *if:*

- *a_1 disables a_2 in s, or*
- *a_1 and a_2 conflict in s.*

We say that a_1 interferes with a_2 in s *if:*

- *a_1 weakly interferes with a_2 in s, or*
- *a_2 disables a_1 in s.*

Action a_1 weakly interferes *with a_2, if there exists a variable $v \in \mathsf{vars}(\mathsf{eff}(a_1))$ such that (1) $v \in \mathsf{vars}(\mathsf{pre}(a_2))$ and $\mathsf{eff}(a_1)[v] \neq \mathsf{pre}(a_2)[v]$, or (2) $v \in \mathsf{vars}(\mathsf{eff}(a_2))$ and $\mathsf{eff}(a_1)[v] \neq \mathsf{eff}(a_2)[v]$.*

We distinguish two concepts of interference, *state-based* interference, and *syntactic* interference. For the former, two actions interfere if there exists a state in which they interfere; for the latter, this additional condition is dropped. Thus, syntactic interference is strictly weaker than state-based interference in the sense that two actions that are in state-based interference always syntactically interfere, but not vice versa.

While interference is an "undirected" relation, i. e., a_1 interferes with a_2 in s iff a_2 interferes with a_1 in s, this is not the case for weak interference, which is directed.

We next define three variants of stubborn set pruning previously introduced for planning. These variants only differ in the notion of interference they employ for applicable actions in the stubborn set.

Definition 66 (Stubborn Sets). *Let Π be a planning task and let s be a solvable non-goal state of Π. Let $T \subseteq \mathcal{A}$ be a set of actions such that the following conditions hold:*

(a) T contains a necessary enabling set for $\mathcal{G}$ in s.

(b) For every $a \in T$, where $a \notin \mathsf{app}(s)$: T contains a necessary enabling set for a in s.

Then T is:

- *a* strong stubborn set, *if for every $a \in T$, where $a \in \mathsf{app}(s)$: T contains all actions a' such that a interferes with a' in any state s' of Π.*

- *a* compliant stubborn set, *if for every $a \in T$, where $a \in \mathsf{app}(s)$: T contains all actions a' such that a weakly interferes with a'.*

- *a* weak stubborn set, *if for every $a \in T$, where $a \in \mathsf{app}(s)$: T contains all actions a' such that a weakly interferes with a' in any state s' of Π, and for all facts $(v = p) \in \mathsf{pre}(a)$:*

  (i) T contains all actions a' such that a' disables a on $\{v = p\}$ in any state s' of Π, or

  (ii) T contains all actions a' such that a' enables a on $\{v = p\}$ in any state s' of Π.

For all variants of stubborn sets (a) ensures that the actions in T make progress to the goal, and (b) this progress chains back to the current state s. The different conditions on action interference lead to different pruning behaviour of the three stubborn set instantiations.

A.4.2 Proofs

Theorem 20. *Decoupled search is exponentially separated from all three variants of stubborn sets.*

Proof. Consider again the variant of our example from Chapter 6.1.1, with a truck and n packages on a map with two locations l and r. Recall that initially everything is at l and the goal is for all packages to be at r. On top of the usual drive and (un)load actions, we introduce a new action $reset()$ with empty precondition and effect that the truck and all packages are at l.

There are only 3 reachable decoupled states: in $\mathcal{I}^{\mathcal{F}}$, all packages can be at l or loaded; via $drive(r)$ we reach state $s_1^{\mathcal{F}}$ where additionally the packages can be at r with a price of 2. From $s_1^{\mathcal{F}}$ we can drive back to l, yielding $s_2^{\mathcal{F}}$, where prices do not change any more. In all states, we can apply $reset()$, which results in a state that is identical to $\mathcal{I}^{\mathcal{F}}$, so is pruned.

All of the stubborn sets variants, however, result in an exponential search space. By $S_{T=l}$ we denote the set of states where the truck is at l and the packages are either loaded or at l. We next show that the search will generate all states in $S_{T=l}$. This results in $> 2^n$ visited states for all stubborn sets variants, which proves the claim. The key observation is that $reset()$ is in conflict (and therefore (weakly) interferes) with any $load(p_i, l)$ actions in all states in $S_{T=l}$ where $load(p_i, l)$ is applicable (condition (2) in Definition 65 fires for compliant stubborn sets). Therefore, since $reset()$ is always applicable, whenever such a load action is in the stubborn set, via $reset()$ all other load actions are added, too. Thus, until all packages are loaded, the search branches over all applicable $load(p_i, l)$ actions.

An algorithm computing stubborn sets could, however, decide to instantiate condition (a) of Definition 66 with $drive(r)$, instead, since it is also part of any plan for all states in $S_{T=l}$. Nevertheless, since $drive(r)$ is in conflict with $reset()$, too, again all load actions end up in the stubborn set, concluding the proof. □

Theorem 21. *All three variants of stubborn sets are exponentially separated from decoupled search.*

Proof. Consider again the task family Π^n from the proof of Lemma 21 with $\Pi^n = \langle \mathcal{V}^n, \mathcal{A}^n, \mathsf{cost}^n, \mathcal{I}^n, \mathcal{G}^n \rangle$ as follows $\mathcal{V}^n = \{v_1, \dots, v_n\}$, where $\mathcal{D}(v_i) = \{0, 1, 2\}$ for $1 \leq i \leq n$. The initial state is $\mathcal{I}^n = \{v_1 = 0, \dots, v_n = 0\}$, the goal state is $\mathcal{G}^n = \{v_1 = 2, \dots, v_n = 2\}$. The actions are $\mathcal{A}^n = \{a_0\} \cup \{a_i^{12} \mid 1 \leq i \leq n\} \cup \{a_{ij}^{12} \mid 1 \leq i, j \leq n\}$ where $\mathsf{pre}(a_0) = \{v_1 = 0, \dots, v_n = 0\}$ and $\mathsf{eff}(a_0) = \{v_1 = 1, \dots, v_n = 1\}$; $\mathsf{pre}(a_i^{12}) = \{v_i = 1\}$ and $\mathsf{eff}(a_i^{12}) = \{v_i = 2\}$; $\mathsf{pre}(a_{ij}^{12}) = \{v_i = 0, v_j = 1\}$ and $\mathsf{eff}(a_{ij}^{12}) = \{v_i = 2, v_j = 2\}$.

First, a_0 is applied to $\mathcal{I}^n$. For the successor state $s_1 := \mathcal{I}^n[\![a_0]\!]$, stubborn sets pick a variable v_i with unsatisfied goal, and add the necessary enabling set actions a_i^{12}, a_{ij}^{12},

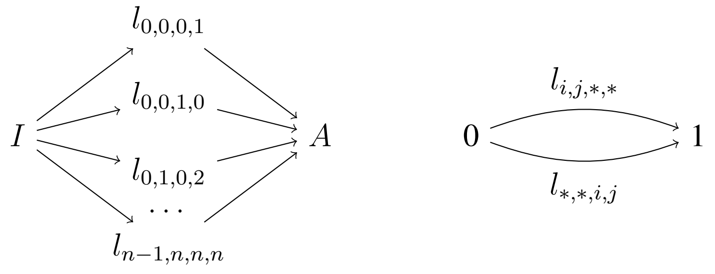


Figure A.6: Illustration of the task family used in the proof of Theorem 22 that exponentially separates decoupled search from symbolic breadth-first search.

and a^{12}_{ji} to the stubborn set T. Since no a^{12}_{kl} action is ever applicable, their only enabler a_0 is added to T. Except a^{12}_i, no action in T is applicable in s_1, but all actions interfering (for any form of interference) with a^{12}_i are already in T, so it is the only action applied to s_1. The same happens in all successors of s_1.

The decoupled state space is exponential in n as shown in the proof of Lemma 21. □

A.5 Relation to Symbolic State Representation

Theorem 22. *Decoupled search is exponentially separated from symbolic breadth-first search with BDDs.*

Proof. Consider the following planning task family Π^n with center factor $C = \{c\}$ and n^2 leaves $L_{i,j} = \{x_{i,j}\}$ that represent the cells in a $n \times n$ grid. Each leaf has two states $0, 1$, which can be represented in a BDD by a variable $x_{i,j}$. The center variable has the domain $\mathcal{D}(c) = \{I, A\} \cup \{l_{i,j,i',j'} \mid \text{cell } (i,j) \text{ is adjacent to } (i',j')\}$, where value $l_{i,j,i',j'}$ encodes that the pair of grid cells (i,j) and (i',j') are adjacent in the grid. Initially, all leaves have value 0, and the center is I. The goal does not matter for our purpose here.

Figure A.6 shows the component state spaces for the center (left) and a leaf (right). These arise from the following actions: there are two types of center actions, $a^C_{I,i,j,i',j'}$, where $\mathsf{pre}(a^C_{I,i,j,i',j'}) = \{c = I\}$ and $\mathsf{eff}(a^C_{I,i,j,i',j'}) = \{c = l_{i,j,i',j'}\}$, and $a^C_{i,j,i',j',A}$, where $\mathsf{pre}(a^C_{i,j,i',j',A}) = \{c = l_{i,j,i',j'}\}$ and $\mathsf{pre}(a^C_{i,j,i',j',A}) = \{c = A\}$. For each leaf variable $x_{i,j}$ and pair of adjacent cells (i,j) and (i',j'), there are two actions, both with effect $\{x_{i,j} = 1\}$, namely $a^L_{i,j,i',j'}$ with precondition $\{c = l_{i,j,i',j'}, x_{i,j} = 0\}$, and $a^L_{i',j',i,j}$ precondition $\{c = l_{i',j',i,j}, x_{i,j} = 0\}$.

This task has a polynomial number of center paths and therefore trivially a polynomial-size decoupled state space.

Consider now symbolic breadth-first search, and the set of states with distance 4 from the initial state. These states arise from moving the center to some $l_{i,j,i',j'}$, moving two adjacent leaves to 1, and moving the center to A. As all the states are in the same center state, we can ignore the impact of c on BDD size (it does not matter where the center bits are in the variable ordering, they only multiply the size of the BDD by a logarithmic factor, $log(n^2)$).

Now, consider the leaf variables. The function that represents our state set is the function that represents the set of all pairs of adjacent variables in a grid. Note that we cannot just represent it by using only $c = A$, because this characterizes the state set where each cells would be adjacent to all other cells. This function requires an exponential number of BDD nodes [Edelkamp and Kissmann, 2011]. $\square$

Appendix B

Full Proofs of Part III

B.1 Decoupled Strong Stubborn Sets

B.1.1 DSSS Special Case Topologies

Lemma 11. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state, let L be a leaf factor, and let p be a partial assignment to L reached in $s^{\mathcal{F}}$. Let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$, and let π be a global plan given $\pi^{\mathcal{F}}$. Let A be a fork-price frontier set for p in $s^{\mathcal{F}}$.*

Define a_t, π^{past}, and π^{future} as before. If there exists $k > t$ such that, denoting by $\langle a_1^L, \dots, a_i^L \rangle$ and $s_0^L, \dots, s_i^L$ the L-actions, respectively states, in π prior to a_k, we have $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_i^L]$ and $s_i^L \models p$, then $\{a_t, \dots, a_{k-1}\} \cap A \neq \emptyset$.

Proof. Denote by l the index of the first action on π^L where a_l^L is on π^{future}. Since we have that $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_i^L]$ by prerequisite, a_i^L must be on π^{future}, i. e., we must have $i \geq l$. Let $j \geq l$ be the smallest index where $\mathsf{cost}(\langle a_1^L, \dots, a_j^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_j^L]$. Consider the transition $s_{j-1}^L \xrightarrow{a_j^L} s_j^L$ on π^L. As $j \geq l$, this transition is part of π^{future}. We prove that *(*) $s_{j-1}^L \xrightarrow{a_j^L} s_j^L$ is a fork-price frontier transition for p in $s^{\mathcal{F}}$.*

Because j is the smallest index where $\mathsf{cost}(\langle a_1^L, \dots, a_j^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_j^L]$, it follows that $\mathsf{prices}(s^{\mathcal{F}})[s_{j-1}^L] \leq \mathsf{cost}(\langle a_1^L, \dots, a_{j-1}^L \rangle)$. Observe that, therefore, s_{j-1}^L is reached in $s^{\mathcal{F}}$, because its price is upper-bounded by a finite value. Furthermore, observe that, adding the cost of a_j^L on both sides of the inequality $\mathsf{prices}(s^{\mathcal{F}})[s_{j-1}^L] \leq \mathsf{cost}(\langle a_1^L, \dots, a_{j-1}^L \rangle)$, we get $\mathsf{prices}(s^{\mathcal{F}})[s_{j-1}^L] + \mathsf{cost}(a_j^L) \leq \mathsf{cost}(\langle a_1^L, \dots, a_j^L \rangle)$. As, by construction, $\mathsf{cost}(\langle a_1^L, \dots, a_j^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_j^L]$, we obtain that $\mathsf{prices}(s^{\mathcal{F}})[s_{j-1}^L] + \mathsf{cost}(a_j^L) < \mathsf{prices}(s^{\mathcal{F}})[s_j^L]$.

Furthermore, $s_{j-1}^L \xrightarrow{a_j^L} s_j^L$ lies on the L-path π^L from $\mathcal{I}[L]$ to s_i^L where $s_i^L \models p$. To show (*), it remains to prove that (a) $\mathsf{center}(s^{\mathcal{F}}) \not\models \mathsf{pre}(a_j^L)[C]$ and (b) π^L is simple.

Regarding (a), because a_j^L affects L, in a fork topology a_j^L cannot affect the center. Hence, $\mathsf{prices}(s^{\mathcal{F}})[s_{j-1}^L] + \mathsf{cost}(a_j^L) < \mathsf{prices}(s^{\mathcal{F}})[s_j^L]$ implies that a_j^L cannot be reached in $s^{\mathcal{F}}$. We do know, however, that s_{j-1}^L is reached in $s^{\mathcal{F}}$. As the only non-L precondition of a_j^L must be on C, (a) follows.

Regarding (b), in a fork topology, any cheapest compliant leaf path is simple, because without center preconditions nor effect on the leaf there is no reason to visit the same leaf state twice.

We have now proved (*). Given this, as A is a goal-price frontier set for L in $s^{\mathcal{F}}$, by definition we know that the segment $\langle a_j^L, \dots, a_i^L \rangle$ of π^L between s_{j-1}^L and s_i^L must contain an action from A. The claim follows as, with $j \geq l$ and by definition of a_i^L, $\langle a_j^L, \dots, a_i^L \rangle$ is a subsequence of $\langle a_t, \dots, a_{k-1} \rangle$. □

Definition 67 (Fork-Past-Maximality). *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a decoupled state, let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$, and let π be a global plan given $\pi^{\mathcal{F}}$.*

Define a_t, π^{past}, and π^{future} as before. We say that π is fork-past-maximal *if, for every leaf factor L and for every $k > t$ where $a_k \in \mathcal{A}^L$ is reached in $s^{\mathcal{F}}$, denoting by $\langle a_1^L, \dots, a_i^L \rangle$ and $s_0^L, \dots, s_i^L$ the L-actions, respectively states, in π prior to a_k, we have* $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_i^L]$.

Lemma 25. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a decoupled state, let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$, and let π be a global plan given $\pi^{\mathcal{F}}$. Then there exists a fork-past-maximal global plan π' given $\pi^{\mathcal{F}}$ where* $\mathsf{cost}(\pi') \leq \mathsf{cost}(\pi)$.

Proof. We obtain such a π' as follows. Start with $\pi' := \pi$. If π' is past-maximizing, stop. Else, select a counter-example L and k, and denote $\langle a_1^L, \dots, a_i^L \rangle$ and $s_0^L, \dots, s_i^L$ as in Definition 67. As $\mathsf{cost}(\langle a_1^L, \dots, a_i^L \rangle) \geq \mathsf{prices}(s^{\mathcal{F}})[s_i^L]$, there exists a $\pi^C(s^{\mathcal{F}})$-compliant L-path π_i^L that ends in s_i^L. So π' remains a plan when removing $\langle a_1^L, \dots, a_i^L \rangle$ from π', and inserting π_i^L as a subsequence of π^{past}. Further, we can move a_k to the end of π^{past}, because its L precondition is achieved by π^L, and its C precondition (if any) is true in $\mathcal{I}[\![\pi^{past}]\!]$ as $\mathsf{pre}(a_k)$ is reached in $s^{\mathcal{F}}$. Now, iterate. This algorithm terminates as, after each step, there is one action less in π^{future}. The outcome π' satisfies the claim as we have not changed the center-action subsequence, and the permuted leaf paths are still compliant with that sequence. □

Theorem 25. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a fork factoring. Let $s^{\mathcal{F}}$ be a solvable decoupled state for which $\langle\rangle$ is not an optimal decoupled plan, and let $\pi^{\mathcal{F}}$ be a strongly optimal decoupled plan for $s^{\mathcal{F}}$. Let $A_s^{\mathcal{F}}$ be an FDSSS for $s^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ contains a center action starting a permutation of $\pi^{\mathcal{F}}$.*

Proof. Let π be a global plan for $\pi^{\mathcal{F}}$, and assume, without loss of generality by Lemma 25, that π is fork-past-maximal. Denote $\pi = \langle a_1, \dots, a_n \rangle$. As above, let a_t be the starting action of $\pi^C(\pi^{\mathcal{F}})$ in π, and denote $\pi^{past} := \langle a_1, \dots, a_{t-1}\rangle$ and $\pi^{future} := \langle a_t, \dots, a_n\rangle$. Then the following properties hold:

(a) There must be an action a shared between $A_s^{\mathcal{F}}$ and π^{future}, i. e., $a \in A_s^{\mathcal{F}} \cap \{a_t, \dots, a_n\}$.

If $\mathcal{G}$ is not reached in $s^{\mathcal{F}}$, then this follows by the same argument as for (a) in the proof of Theorem 23.

If $\mathcal{G}$ is reached in $s^{\mathcal{F}}$, then this follows by the same argument as for (a) in the proof of Theorem 24, replacing Lemma 10 with the variant of Lemma 11 for fork-goal-price frontier sets.

Given (a), let a_k be the first shared action, i. e., say that $a_k \in A_s^{\mathcal{F}}$ and $\{a_t, \dots, a_{k-1}\} \cap A_s^{\mathcal{F}} = \emptyset$.

(b) a_k is reached in $s^{\mathcal{F}}$.

By the same argument as for (b) in the proof of Theorem 23.

(c) a_k is a center action, $a_k \in \mathcal{A}^C$.

Assume for contradiction that a_k is a leaf action, $a_k \in \mathcal{A}^L$. First, with (b) and due to Definition 36, we then know that $A_s^{\mathcal{F}}$ contains a fork-price frontier set A for $p := \mathsf{pre}(a_k)[L]$ in $s^{\mathcal{F}}$. We show that π^{future} contains an action from A in front of a_k, in contradiction to a_k being the first shared action.

Denote by $\langle a_1^L, \dots, a_i^L\rangle$ and $s_0^L, \dots, s_i^L$ the L-actions, respectively states, in π prior to a_k. We know that $k \geq t$ because a_k is on π^{future}. In fact, as a_t is a center action, given the fork topology we know that $a_k \neq a_t$ and hence $k > t$. Given this, as π is fork-past-maximal and a_k is reached in $s^{\mathcal{F}}$ by (b), we have that $\mathsf{cost}(\langle a_1^L, \dots, a_i^L\rangle) < \mathsf{prices}(s^{\mathcal{F}})[s_i^L]$. Furthermore, of course $s_i^L \models p = \mathsf{pre}(a_k)[L]$. We can therefore apply Lemma 11 and get that $\{a_t, \dots, a_{k-1}\} \cap A \neq \emptyset$, as we needed to show.

(d) a_k does not interfere with any of the actions a_i, $t \leq i \leq k-1$, where $\mathsf{pre}(a) \parallel \mathsf{pre}(a_i)$.

With (b) and (c), and as Definition 36 (iii) includes interfering actions for reached center actions in $A_s^{\mathcal{F}}$, this follows by the same argument as for (c) in the proof of Theorem 23.

(e) a_k can be moved to the start of π^{future}. Precisely, $\pi' := \pi^{past} \circ \langle a_k, a_t, \dots, a_{k-1}, a_{k+1}, \dots, a_n\rangle$ is a plan for Π.

By the same argument as for (e) in the proof of Theorem 23, except that we do not need to take care of leaf preconditions as, in a fork structure, the center action a_k cannot have any such preconditions.

The claim now follows with the same concluding argument as in the proof of Theorem 23. □

Definition 68 (DSSS: Enhanced)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a star factoring. Denote by $\mathcal{F}^{FL}$ the set of fork leaves in $\mathcal{F}$, denote by $\mathcal{F}^{IFL}$ the set of inverted-fork leaves in $\mathcal{F}$, and denote $\mathcal{F}^{GL} := \mathcal{L} \setminus (\mathcal{F}^{FL} \cup \mathcal{F}^{IFL})$. Let $s^{\mathcal{F}}$ be a decoupled state where $\mathcal{G}[\bigcup_{L \in \mathcal{F}^{IFL}} L]$ is reached in $s^{\mathcal{F}}$.*

An action set $A_s^{\mathcal{F}}$ is an enhanced decoupled strong stubborn set (EDSSS) *for $s^{\mathcal{F}}$ if all of the following conditions hold:*

(i) If $\mathcal{G}$ is not reached in $s^{\mathcal{F}}$, then $A_s^{\mathcal{F}}$ contains a decoupled necessary enabling set for $\mathcal{G}$ in $s^{\mathcal{F}}$, discarding actions a' where $\mathsf{pre}(a')[\bigcup_{L \in \mathcal{F}^{IFL}} L]$ is not reached in $s^{\mathcal{F}}$.

If $\mathcal{G}$ is reached in $s^{\mathcal{F}}$, then for every $L \in \mathcal{F}^{GL}$ where $\mathcal{G}[L] \neq \emptyset$, $A_s^{\mathcal{F}}$ contains a goal-price frontier set for L in $s_G^{\mathcal{F}}$, discarding actions a' where $\mathsf{pre}(a')[\bigcup_{L \in \mathcal{F}^{IFL}} L]$ is not reached in $s^{\mathcal{F}}$; and for every $L \in \mathcal{F}^{FL}$ where $\mathcal{G}[L] \neq \emptyset$, $A_s^{\mathcal{F}}$ contains a fork-goal-price frontier set for L in $s_G^{\mathcal{F}}$.

(ii) For all actions $a \in A_s^{\mathcal{F}}$ not reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains a decoupled necessary enabling set for $\mathsf{pre}(a)$ in $s^{\mathcal{F}}$, discarding actions a' where $\mathsf{pre}(a')[\bigcup_{L \in \mathcal{F}^{IFL}} L]$ is not reached in $s^{\mathcal{F}}$.

(iii) For all actions $a \in A_s^{\mathcal{F}} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains all actions a' interfering with a where $\mathsf{pre}(a) \parallel \mathsf{pre}(a')$ and $\mathsf{pre}(a')[\bigcup_{L \in \mathcal{F}^{IFL}} L]$ is reached in $s^{\mathcal{F}}$.

(iv) For all actions $a \in A_s^{\mathcal{F}} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ reached in $s^{\mathcal{F}}$, and for all L where $\mathsf{pre}(a)[L] \neq \emptyset$, $A_s^{\mathcal{F}}$ contains a reached-enabling set for $\mathsf{pre}(a)[L]$ in $s^{\mathcal{F}}$; and for all actions $a^L \in A_s^{\mathcal{F}} \cap \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains a fork-price frontier set for $\mathsf{pre}(a^L)[L]$ in $s^{\mathcal{F}}$

Note that, in (i), actions in a fork-goal-price frontier set cannot have a precondition on an inverted-fork leaf, so we do not need to exclude actions with unreached inverted-fork leaf preconditions. In (iv), for non-fork-leaf actions a (first case in the item), a cannot have a precondition on any such leaf. Reached-enabling sets are thus collected only for non-fork leaves.

Theorem 46. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\mathcal{F}$ a star factoring. Denote $\mathcal{F}^{FL}$, $\mathcal{F}^{IFL}$, and $\mathcal{F}^{GL}$ as in Definition 68. Let $s^{\mathcal{F}}$ be a solvable decoupled state (in particular, $\mathcal{G}[\bigcup_{L \in \mathcal{F}^{IFL}} L]$ is reached in $s^{\mathcal{F}}$), and let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$. Let $A_s^{\mathcal{F}}$ be an EDSSS for $s^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ contains a center action starting a permutation of $\pi^{\mathcal{F}}$.*

Proof. Let π be a global plan for $\pi^{\mathcal{F}}$. Observe first that the notions of past-maximality and fork-past-maximality affect individual leaves only, i. e., the actions a_k in question affect exactly one leaf factor. So we can transform π as per Lemma 8 to be past-maximal for non-fork leaves; and we can transform it as per Lemma 25 to be fork-past-maximal for fork leaves. We can hence assume that π has these properties, without loss of generality.

Denote $\pi = \langle a_1, \dots, a_n \rangle$. As before, let a_t be the starting action of $\pi^C(\pi^{\mathcal{F}})$ in π, and denote $\pi^{past} := \langle a_1, \dots, a_{t-1} \rangle$ and $\pi^{future} := \langle a_t, \dots, a_n \rangle$. Then the following properties hold:

(a) There must be an action a shared between $A_s^{\mathcal{F}}$ and π^{future}, i. e., $a \in A_s^{\mathcal{F}} \cap \{a_t, \dots, a_n\}$.

If $\mathcal{G}$ is not reached in $s^{\mathcal{F}}$, this holds by the same argument as for (a) in the proof of Theorem 23.

If $\mathcal{G}$ is reached in $s^{\mathcal{F}}$, then for non-fork leaves this holds by the same argument as for (a) in the proof of Theorem 24, and for fork leaves this holds by the same argument as for (a) in the proof of Theorem 25.

In all but the last case, we invoke Proposition 9 to show that actions with unreached inverted-leaf preconditions can be discarded.

Given (a), let a_k be the first shared action, i. e., say that $a_k \in A_s^{\mathcal{F}}$ and $\{a_t, \dots, a_{k-1}\} \cap A_s^{\mathcal{F}} = \emptyset$.

(b) a_k is reached in $s^{\mathcal{F}}$.

By the same argument as for (b) in the proof of Theorem 23, invoking Proposition 9 to show that actions with unreached inverted-leaf preconditions can be discarded.

(c) If $a_k \in \mathcal{A} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$, then a_k does not interfere with any of the actions a_i, $t \leq i \leq k-1$, where $\mathsf{pre}(a) \parallel \mathsf{pre}(a_i)$.

By the same argument as for (c) in the proof of Theorem 23, applied to $a_k \in \mathcal{A} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ with Definition 68 (iii), and invoking Proposition 9 to show that actions with unreached inverted-leaf preconditions can be discarded.

(d) $a_k \in \mathcal{A} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$.

Assume for contradiction that a_k is a fork-leaf action, $a_k \in \mathcal{A}^L$ for a fork leaf L. With (b) and due to Definition 68, $A_s^{\mathcal{F}}$ contains a fork-price frontier set A for $p := \mathsf{pre}(a_k)[L]$ in $s^{\mathcal{F}}$. From here, the argument is the same as for (c) in the proof of Theorem 25, invoking fork-past-maximality for L.

(e) The leaf precondition of a_k is true at the end of π^{past}, i. e., in $\mathcal{I}[\![\pi^{past}]\!]$.

By the same argument as for (d) in the proof of Theorem 23, applied to $a_k \in \mathcal{A} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ with (d) and Definition 68 (iv).

(f) a_k can be moved to the start of π^{future}. Precisely, $\pi' := \pi^{past} \circ \langle a_k, a_t, \dots, a_{k-1}, a_{k+1}, \dots, a_n\rangle$ is a plan for Π.

By the same argument as for (e) in the proof of Theorem 23, applied to $a_k \in \mathcal{A} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ with (d).

(g) a_k is a center action.

By the same argument as for (e) in the proof of Theorem 23, applied to $a_k \in \mathcal{A} \setminus \bigcup_{L \in \mathcal{F}^{FL}} \mathcal{A}^L$ with (d), invoking past-maximality for the non-fork leaf L.

The claim now follows with the same concluding argument as in the proof of Theorem 23. □

Definition 69 (DSSS: Inverted Forks with Single Variables)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task and $\mathcal{F}$ an inverted-fork factoring where every leaf consists of a single variable, $\forall L \in \mathcal{L} : |L| = 1$. Let $s^{\mathcal{F}}$ be a non-goal decoupled state where $\mathcal{G}[\mathcal{V} \setminus C]$ is reached in $s^{\mathcal{F}}$.*

An action set $A_s^{\mathcal{F}}$ is a single-variable-inverted-fork decoupled strong stubborn set (SVIFDSSS) *for $s^{\mathcal{F}}$ if all of the following conditions hold:*

(i) $A_s^{\mathcal{F}}$ contains a center-necessary enabling set for $\mathcal{G}[C]$ in $s^{\mathcal{F}}$.

(ii) For all center actions $a \in A_s^{\mathcal{F}} \cap \mathcal{A}^C$ not reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains a center-necessary enabling set for $\mathsf{pre}(a)[C]$ in $s^{\mathcal{F}}$.

(iii) For all center actions $a \in A_s^{\mathcal{F}} \cap \mathcal{A}^C$ reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains all center actions a' interfering with a where $\mathsf{pre}(a) \parallel \mathsf{pre}(a')$ and $\mathsf{pre}(a')[\mathcal{V} \setminus C]$ is reached in $s^{\mathcal{F}}$.

(iv) For all center actions $a \in A_s^{\mathcal{F}} \cap \mathcal{A}^C$ reached in $s^{\mathcal{F}}$, $A_s^{\mathcal{F}}$ contains all center actions a' that have a leaf precondition competing with a, $\mathsf{pre}(a)[\mathcal{V} \setminus C] \nparallel \mathsf{pre}(a')[\mathcal{V} \setminus C]$, and $\mathsf{pre}(a')[\mathcal{V} \setminus C]$ is reached in $s^{\mathcal{F}}$.

Theorem 47. *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathsf{cost}, \mathcal{I}, \mathcal{G}\rangle$ be a planning task and $\mathcal{F}$ an inverted-fork factoring where every leaf consists of a single variable, $\forall L \in \mathcal{L} : |L| = 1$. Let $s^{\mathcal{F}}$ be a solvable non-goal decoupled state, and let $\pi^{\mathcal{F}}$ be a decoupled plan for $s^{\mathcal{F}}$. Let $A_s^{\mathcal{F}}$ be a SVIFDSSS for $s^{\mathcal{F}}$. Then $A_s^{\mathcal{F}}$ contains a center action starting a permutation of $\pi^{\mathcal{F}}$.*

Proof. Let π be a global plan for $\pi^{\mathcal{F}}$, and assume, without loss of generality, by Lemma 8, that π is past-maximal. Denote $\pi = \langle a_1, \dots, a_n\rangle$. As above, let a_t be the starting action of $\pi^C(\pi^{\mathcal{F}})$ in π, and denote $\pi^{past} := \langle a_1, \dots, a_{t-1}\rangle$ and $\pi^{future} := \langle a_t, \dots, a_n\rangle$. Then the following properties hold:

(a) There must be a center action shared between $A_s^{\mathcal{F}}$ and π^{future}, i. e., $a \in A_s^{\mathcal{F}} \cap \{a_t, \dots, a_n\}$. Such an action must exist by the same argument as for (a) in the proof of Theorem 23. It must be a center action due to Proposition 9.

Given (a), let a_k be the first shared action, i. e., say that $a_k \in A_s^{\mathcal{F}}$ and $\{a_t, \dots, a_{k-1}\} \cap A_s^{\mathcal{F}} = \emptyset$.

(b) a_k is reached in $s^{\mathcal{F}}$.

By the same argument as for (b) in the proof of Theorem 46 .

(c) a_k does not interfere with any of the center actions a_i in $\langle a_t, \dots, a_{k-1}\rangle$, where $\mathsf{pre}(a_k) \parallel \mathsf{pre}(a_i)$.

For the center actions in $\langle a_t, \dots, a_{k-1}\rangle$, this holds by the same argument as for (c) in the proof of Theorem 46.

(d) The leaf preconditions of a_k agree with those of all center actions a_i on $\langle a_t, \dots, a_{k-1}\rangle$; formally $\mathsf{pre}(a_k)[\mathcal{V} \setminus C] \parallel \mathsf{pre}(a_i)[\mathcal{V} \setminus C]$, $t \leq i < k$.

Assume for contradiction that there exists an action a_i where $\mathsf{pre}(a_k)[\mathcal{V} \setminus C] \not\parallel \mathsf{pre}(a_i)[\mathcal{V} \setminus C]$ preceding a_k on π^{future}. By (a) and (b) a_k is a reached center action. Thus, with Definition 69 (iv), $A_s^{\mathcal{F}}$ contains all center actions a' where $\mathsf{pre}(a_k)[\mathcal{V} \setminus C] \not\parallel \mathsf{pre}(a')[\mathcal{V} \setminus C]$, invoking Proposition 9 to show that actions with unreached leaf preconditions can be discarded. But then, a_i is such an action and is therefore contained in $A_s^{\mathcal{F}}$, in contradiction to a_k being the first shared action.

(e) No action on $\langle a_t, \dots, a_{k-1}\rangle$ affects a variable in $\mathsf{vars}(\mathsf{pre}(a_k)) \cap \bigcup_{L \in \mathcal{L}} L$.

Observe first that, due to the inverted-fork structure, every action that affects a variable in $\mathsf{vars}(\mathsf{pre}(a_k)) \cap \bigcup_{L \in \mathcal{L}} L$ must be a leaf action $a \in \mathcal{A}^{\mathcal{L}}$. Moreover, all $\mathcal{A}^L$ are disjoint and $\mathsf{vars}(\mathsf{pre}(a)) \cup \mathsf{vars}(\mathsf{eff}(a)) = L$ for all $a \in \mathcal{A}^L$ (because $|L| = 1$, for all $L \in \mathcal{L}$). Further, if $\mathsf{vars}(\mathsf{pre}(a_k)) \cap L \neq \emptyset$ then $L \subseteq \mathsf{vars}(\mathsf{pre}(a_k))$.

To prove the claim, we next show that there exists no action a_i on $\langle a_t, \dots, a_{k-1}\rangle$, with $t \leq i < k$, where $a_i \in \mathcal{A}^L$ and $L \subseteq \mathsf{vars}(\mathsf{pre}(a_k))$. Assume for contradiction that such an a_i exists. From (b), we have that (1) a_k is reached in $s^{\mathcal{F}}$. In particular, its leaf precondition on $\mathsf{vars}(\mathsf{pre}(a_k)) \cap L$ is reached in $s^{\mathcal{F}}$. With (d), we have that (2) no center action on $\langle a_t, \dots, a_{k-1}\rangle$ has a competing leaf precondition, so in particular there exists no center action a_j, with $t \leq j < k$, s.t. $\mathsf{pre}(a_k)[L] \not\parallel \mathsf{pre}(a_j)[L]$.

Denote by $\pi^L_{\to k}$ the L-affecting leaf-action subsequence in π prior to a_k that includes a_i. Because π is a plan, $\pi^L_{\to k}$ is compliant with $\pi^C(s^{\mathcal{F}}) \circ \pi^C(\langle a_t, \dots, a_{k-1}\rangle)$. From (1) and (2), it follows that $\pi^L_{\to k}$ is also compliant with $\pi^C(s^{\mathcal{F}})$. Given that by prerequisite a_i is in $\langle a_t, \dots, a_{k-1}\rangle$ this is in contradiction to the assumption that π is past-maximal.

(f) The leaf precondition of a_k is true at the end of π^{past}, i. e., in $\mathcal{I}[\![\pi^{past}]\!]$.

This follows directly from (b) and (e), and the fact the π is a plan.

(g) a_k can be moved to the start of π^{future}. Precisely, $\pi' := \pi^{past} \circ \langle a_k, a_t, \dots, a_{k-1}, a_{k+1}, \dots, a_n\rangle$ is a plan for Π.

Given (b) and (f), a_k is applicable in $\mathcal{I}[\![\pi^{past}]\!]$. With (c) a_k does not interfere with any center action a_i in $\langle a_t, \dots, a_{k-1}\rangle$ by the same argument as in the proof of Theorem 23. The only leaf actions it can interfere with are those affecting $\mathsf{vars}(\mathsf{pre}(a_k)) \cap \bigcup_{L\in\mathcal{L}} L$, but with (e) there are no such action on π^{future} prior to a_k, concluding the argument.

The claim now follows with the same concluding argument as in the proof of Theorem 23. □

B.1.2 Exponential Separations

Theorem 26. *DSSS is exponentially separated from both, decoupled search and SSS.*

Proof. Consider a variant of our running example with M trucks and N packages on a map with two locations A and B, where each truck T_i is associated with a group of N packages that only T_i can transport (all trucks and packages start at A, all packages must be transported to B).

The number of reachable decoupled states is exponential in M, because all trucks must be in the center factor, and their move combinations are enumerated. For SSS, as soon as A_s contains a load/unload action for one group of N packages, the load/unload actions for all other packages in that group are present as well, due to interference. So the SSS-pruned reachable state space has size exponential in N.

Consider now decoupled search with DSSS pruning. In $\mathcal{I}^{\mathcal{F}}$, all packages can be at A or loaded into their respective truck. The necessary enabling set for $\mathcal{G}$ will select one package, associated with some truck T_i; hence $A_s^{\mathcal{F}}$ includes drive(T_i, A, B). This does not interfere with the drive actions for the other trucks, so it is the *only* applicable center action in $A_s^{\mathcal{F}}$, and we get a single successor state $s^{\mathcal{F}}$. In $s^{\mathcal{F}}$, the packages associated with T_i can all be at B. So the decoupled necessary enabling set for $\mathcal{G}$ for DSSS selects a package associated with another truck $T_j \neq T_i$. The only non-pruned action is drive(T_j, A, B); and so forth. Once all trucks are at B, we have a goal decoupled state $s_G^{\mathcal{F}}$. The goal-price frontier sets for all $L \in \mathcal{L}$ in $s_G^{\mathcal{F}}$ are empty, because the package prices are already the cheapest possible ones. So there are exactly $M + 1$ reachable decoupled states. □

B.2 Dominance Pruning for Fork Topologies

In this section, we will give the full proofs of the claims from Chapter 13.

Theorem 31. *$\preceq_F$ is a decoupled dominance relation.*

Proof. Whenever $s^{\mathcal{F}} \preceq_F t^{\mathcal{F}}$ and a global plan for $s^{\mathcal{F}}$ exists, we can construct a global plan that is at least as good for $t^{\mathcal{F}}$. Consider any global plan for $s^{\mathcal{F}}$ consisting of a center path π^C, and a goal leaf path π^L compliant with $\pi^C(s^{\mathcal{F}}) \circ \pi^C$ starting in $s^L_{\mathcal{I}}$ for each leaf. As $s^{\mathcal{F}} \preceq_F t^{\mathcal{F}}$ implies that $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$, π^C is applicable to $t^{\mathcal{F}}$ as well.

Denote $\pi^L = \langle a_1, \dots, a_n \rangle$ and denote the leaf states it traverses by $s^L_{\mathcal{I}} = s^L_0, \dots, s^L_n = s^L_G$. Let s^L_i be the last state visited by π^L such that $\mathsf{prices}(s^{\mathcal{F}})[s^L_i] = \sum_{j=1}^{i} \mathsf{cost}(a_j)$. Such state must exist because the price of the initial state is always 0. s^L_i is the last state in which its price corresponds to the price obtained by π^L so $\langle a_{i+1}, \dots, a_n \rangle$ must be compliant with π^C.

It follows that $s^L_i \in F(s^{\mathcal{F}})$. This is trivially true if $s^L_i = s^L_n$ since the goal state always belongs to the frontier. If $s^L_i \neq s^L_n$, then we have $\mathsf{prices}(s^{\mathcal{F}})[s^L_i] + \mathsf{cost}(a_{i+1}) < \mathsf{prices}(s^{\mathcal{F}})[s^L_{i+1}]$, which implies $s^L_i \in F(s^{\mathcal{F}})$. Since $s^{\mathcal{F}} \preceq_F t^{\mathcal{F}}$, $\mathsf{prices}(t^{\mathcal{F}})[s^L_i] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L_i]$ follows.

Consider the path π^L_t from $s^L_{\mathcal{I}}$ to s^L_G constructed as the concatenation of: a cheapest $\pi^C(t^{\mathcal{F}})$-compliant path to s^L_i with the postfix of π^L behind s^L_i. Then $\mathsf{cost}(\pi^L_t) = \mathsf{prices}(t^{\mathcal{F}})[s^L_i] + \sum_{j=i+1}^{n} \mathsf{cost}(a_j) \leq \mathsf{cost}(\pi^L)$. Since the first part of the plan up to s^L_i is compliant with $\pi^C(t^{\mathcal{F}})$ and the rest is compliant with π^C, π^L_t is compliant with $\pi^C(t^{\mathcal{F}}) \circ \pi^C$ as desired. □

Theorem 32. *$\preceq_F$ subsumes $\preceq$ and is exponentially separated from it.*

Proof. Subsumption is guaranteed as both relations are based on comparing the same pricing functions, but $\preceq_F$ does so on a subset of leaf states. The exponential separation is shown by our logistics example, in which decoupled search with $\preceq$ incurs an exponential blow-up, as explained before.

Note that the prices of (p, l_1) and (p, T) are always 0 and 1, respectively. Since the only leaf action applicable from (p, l_i) is $\mathsf{load}(T, p, l_i)$ and the price of (p, T) is always lower than that of (p, l_i) for $i > 1$, $(p, l_i) \notin F(s^{\mathcal{F}})$ for $i > 2$ for any reachable decoupled state $s^{\mathcal{F}}$. Hence, when using $\preceq_F$, there are at most 2 different decoupled states for every center state, depending on whether the price of (p, l_2) is 2 or ∞. Therefore, the decoupled state space with $\preceq_F$ has size linear in n instead of exponential. □

Theorem 33. *$\preceq_E$ is a decoupled dominance relation.*

Proof. Whenever $s^{\mathcal{F}} \preceq_E t^{\mathcal{F}}$ and a global plan for $s^{\mathcal{F}}$ exists, we can construct an at least as good global plan for $t^{\mathcal{F}}$. Consider any global plan for $s^{\mathcal{F}}$ consisting of a center path π^C, and a goal leaf path π^L for each leaf starting in $s^L_{\mathcal{I}}$ compliant with $\pi^C(s^{\mathcal{F}}) \circ \pi^C$.

As $s^{\mathcal{F}} \preceq_E t^{\mathcal{F}}$ implies that $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$, π^C is applicable to $t^{\mathcal{F}}$ as well. Let s^L be a leaf state traversed by π^L such that the prefix of π^L up to s^L is compliant with $\pi^C(s^{\mathcal{F}})$ and the suffix of π^L starting in s^L, π^L_s, is compliant with π^C. Denote $\pi^L_s = \langle a_1, \dots, a_n \rangle$ and denote the leaf states it traverses by $s^L = s^L_0, \dots, s^L_n = s^L_G$.

Let s^L_i be the first state visited by π^L_s such that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] = \mathsf{prices}(t^{\mathcal{F}})[s^L_i]$. Such state exists by definition since $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_n] = \mathsf{prices}(t^{\mathcal{F}})[s^L_n]$. Then, for all $j < i$, $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_j] \neq \mathsf{prices}(t^{\mathcal{F}})[s^L_j]$, and thus by the definition of effective prices we have that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_j] \geq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_{j+1}] - \mathsf{cost}(a_{j+1})$. Accumulating these inequalities, we get *(*)* $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_0] \geq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] - \sum_{j=1}^{i} \mathsf{cost}(a_j)$.

Consider now the path π^L_t from $s^L_{\mathcal{I}}$ to s^L_G constructed as the concatenation of: a cheapest $\pi^C(t^{\mathcal{F}})$-compliant path to s^L_i with the postfix of π^L_s behind s^L_i. Then $\mathsf{cost}(\pi^L_t) = \mathsf{prices}(t^{\mathcal{F}})[s^L_i] + \sum_{j=i+1}^{n} \mathsf{cost}(a_j)$. As $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] = \mathsf{prices}(t^{\mathcal{F}})[s^L_i]$, we get $\mathsf{cost}(\pi^L_t) = \mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] + \sum_{j=i+1}^{n} \mathsf{cost}(a_j)$. With (*), we get the desired property that:

$$
\begin{aligned}
& \mathsf{cost}(\pi^L_t) \\
\leq\ & \mathsf{Eprices}(t^{\mathcal{F}})[s^L] + \textstyle\sum_{j=1}^{i} \mathsf{cost}(a_j) + \sum_{j=i+1}^{n} \mathsf{cost}(a_j) \\
=\ & \mathsf{Eprices}(t^{\mathcal{F}})[s^L] + \mathsf{cost}(\pi^L_s) \\
\leq\ & \mathsf{prices}(s^{\mathcal{F}})[s^L] + \mathsf{cost}(\pi^L_s) \\
=\ & \mathsf{cost}(\pi^L)
\end{aligned}
$$

□

Theorem 34. *$\preceq_E$ subsumes $\preceq_F$ and is exponentially separated from it.*

Proof. To prove that $\preceq_E$ subsumes $\preceq_F$, we show that if $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$ for all $s^L \in F(s^{\mathcal{F}})$, then it also holds for all $s^L \notin F(s^{\mathcal{F}})$.

Consider any $s^L \notin F(s^{\mathcal{F}})$. Let $R^L \subseteq F(s^{\mathcal{F}})$ be the set of leaf states in the frontier of $s^{\mathcal{F}}$ reachable from s^L through a path that does not traverse any other state in $F(s^{\mathcal{F}})$. Note that, since $s^L_G \in F(s^{\mathcal{F}})$, any path from s^L to s^L_G necessarily passes through some $r^L \in R^L$. Let $r^L = \mathrm{argmax}_{t^L \in R^L} \mathsf{Eprices}(t^{\mathcal{F}})[t^L] - \mathsf{cost}(s^L, t^L)$, where $\mathsf{cost}(s^L, t^L)$ is the cost of the cheapest L-path from s^L to t^L.

First we show that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq \mathsf{Eprices}(t^{\mathcal{F}})[r^L] - \mathsf{cost}(s^L, r^L)$. By definition, $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq \max_{s^L \xrightarrow{a} t^L} (\mathsf{Eprices}(t^{\mathcal{F}})[t^L] - \mathsf{cost}(a))$. Let s^L_1 be any t^L that maximizes such expression. If $s^L_1 \in R^L$ then $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_1] - \mathsf{cost}(s^L, s^L_1) \leq \mathsf{Eprices}(t^{\mathcal{F}})[r^L] - \mathsf{cost}(s^L, r^L)$ and we are done. If $s^L_1 \notin R^L$ then we can compose the definition to obtain $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq \mathsf{Eprices}(t^{\mathcal{F}})[s^L_1] - \mathsf{cost}(a) \leq \max_{s^L_1 \xrightarrow{a} s^L_2} (\mathsf{Eprices}(t^{\mathcal{F}})[s^L_2] - \mathsf{cost}(a))$. By repeating the same argument we have that for at least one leaf state $u^L \in R^L$. This is clear if there are no 0-cost actions because $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] < \mathsf{Eprices}(t^{\mathcal{F}})[s^L_{i+1}]$ so the effective prices monotonically increase along this chain and there can be no cycles.

If there are 0-cost actions, there may be a 0-cost cycle, so we must prove that even in that case, there exists yet another state u^L outside the cycle such that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L] \leq$

$\mathsf{Eprices}(t^{\mathcal{F}})[u^L] - \mathsf{cost}(s^L, u^L)$. Consider the set of states, S^L_s for which there is a 0-cost cycle such that $S^L_s \cap R^L = \emptyset$ and all the states in S^L_s have the same effective price. Let c be a constant such that $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] = c$ for all $s^L_i \in S^L_s$. Then, there exists $t^L \notin S^L_s$ such that $s^L_i \xrightarrow{a} t^L$, and $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] \leq \mathsf{Eprices}(t^{\mathcal{F}})[t^L] - \mathsf{cost}(a)$ for some $s^L_i \in S^L_s$. Otherwise, $\mathsf{Eprices}(t^{\mathcal{F}})[s^L_i] = c'$ for all $s^L_i \in S^L_i$ and some $c' < c$ would satisfy the equation for which $\mathsf{Eprices}()$ is defined to be the minimum point-wise that satisfies it.

Finally, since $s^L \notin F(s^{\mathcal{F}})$, $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{prices}(s^{\mathcal{F}})[t^L] - \mathsf{cost}(a)$ for all $s^L \xrightarrow{a} t^L$. Since all states in the path from s^L to any $r^L \in R^L$ do not belong to $F(s^{\mathcal{F}})$, we have the inequality $\mathsf{prices}(s^{\mathcal{F}})[s^L] \geq \mathsf{prices}(s^{\mathcal{F}})[r^L] - \mathsf{cost}(s^L, r^L)$. Combining the previous inequalities we get:

$$\begin{aligned} & \mathsf{Eprices}(t^{\mathcal{F}})[s^L] \\ \leq\ & \mathsf{Eprices}(t^{\mathcal{F}})[r^L] - \mathsf{cost}(s^L, r^L) \\ \leq\ & \mathsf{prices}(s^{\mathcal{F}})[r^L] - \mathsf{cost}(s^L, r^L) \\ \leq\ & \mathsf{prices}(s^{\mathcal{F}})[s^L] \end{aligned}$$

For the exponential separation, consider the planning task of our logistics example with an additional teleport device that allows to teleport the package between any two locations that are not the goal. Hence, we have a new center variable $\mathrm{tel\text{-}activated}$ with domain $\{\bot, \top\}$, initially set to $\bot$ and two new actions $\mathrm{activate\text{-}teleport}$ that sets $\mathrm{tel\text{-}activated}$ to $\top$, and $\mathrm{teleport\text{-}package}(l_i, l_j)$ for $i, j \neq 2$ with precondition $\{(\mathrm{tel\text{-}activated}, \top), (p, l_i)\}$ and effect $\{(p, l_j)\}$. The cost of $\mathrm{activate\text{-}teleport}$ is n and all other actions have cost 1.

In this example, decoupled search with $\preceq_F$ expands an exponential number of states with $(\mathrm{tel\text{-}activated}, \bot)$ since for all reached (p, l_i) with $i > 2$, $(p, l_i) \in F(s^{\mathcal{F}})$ as long as any (p, l_i) is still not reached. When using $\preceq_E$, $\mathsf{prices}(\mathcal{I}^{\mathcal{F}})[\{(p, T)\}] = 1$ in every state, so $\mathsf{Eprices}(s^{\mathcal{F}})[\{(p, l_i)\}] = 0$. Therefore, the state space is linear in the number of locations. □

Lemma 14. *Let $\preceq^L$ be a leaf simulation and π^C a center path. If $s^L \preceq^L t^L$ and there exists a path π^L_s from s^L to s^L_G compliant with π^C, then there exists a path π^L_t from t^L to s^L_G compliant with π^C such that* $\mathsf{cost}(\pi^L_t) \leq \mathsf{cost}(\pi^L_s)$.

Proof. Assume without loss of generality that π^L_s is a shortest optimal L-path compliant with π^C. Proof by induction on $|\pi^L_s|$. Base case, $|\pi^L_s| = 0$. Then, $s^L = s^L_G$ and $t^L = s^L_G$ because $s^L_G \not\preceq^L t^L$ for any $t^L \neq s^L_G$. Thus, $\pi^L_s = \pi^L_t = \langle\rangle$.

Inductive case. Let $s^L \xrightarrow{a^L_s} s^L_2$ be the first transition in π^L_s. Since $s^L \preceq^L t^L$, either (i) $s^L_2 \preceq^L t^L$ or (ii) exists $t^L \xrightarrow{a^L_t} t^L_2$ such that a^L_t dominates a^L_s and $s^L_2 \preceq^L t^L_2$. If (i) holds then the claim follows by induction since it must exist π^L_t with $\mathsf{cost}(\pi^L_t) \leq \mathsf{cost}(\pi^L_{s2})$ where π^L_{s2} is a shortest optimal π^C-compliant plan from s^L_2 and necessarily $\mathsf{cost}(\pi^L_{s2}) \leq \mathsf{cost}(\pi^L_s)$ and π^L_{s2} is shorter than π^L_s.

If (ii) holds, by induction we know that the claim holds for s^L_2 and t^L_2, so there exists a plan π^L_{t2}. Thus, $\mathsf{cost}(\pi^L_t) = \mathsf{cost}(\pi^L_{t2}) + \mathsf{cost}(a^L_t) \leq \mathsf{cost}(\pi^L_{s2}) + \mathsf{cost}(a^L_t) \leq \mathsf{cost}(\pi^L_{s2}) + \mathsf{cost}(a^L_s) = \mathsf{cost}(\pi^L_s)$. Moreover, since π^L_s is compliant with π^C and $\mathsf{pre}(a^L_t)[C] \subseteq \mathsf{pre}(a^L_s)[C]$, π^L_t is also compliant with π^C. □

Theorem 35. *$\preceq_S$ is a decoupled dominance relation.*

Proof. Whenever $s^{\mathcal{F}} \preceq_S t^{\mathcal{F}}$ and a global plan for $s^{\mathcal{F}}$ exists, we can construct an at least as good global plan for $t^{\mathcal{F}}$. Consider any global plan for $s^{\mathcal{F}}$ consisting of a center path π^C, and a goal leaf path π^L for each leaf starting in $s^L_{\mathcal{I}}$ compliant with $\pi^C(s^{\mathcal{F}}) \circ \pi^C$. As $s^{\mathcal{F}} \preceq_S t^{\mathcal{F}}$ implies that $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$, π^C is applicable to $t^{\mathcal{F}}$ as well. Let s^L be a leaf state traversed by π^L such that the prefix of π^L up to s^L is compliant with $\pi^C(s^{\mathcal{F}})$ and the suffix of π^L starting in s^L, π^L_s, is compliant with π^C. Denote $\pi^L_s = \langle a_1, \dots, a_n \rangle$ and denote the leaf states it traverses by $s^L = s^L_0, \dots, s^L_n = s^L_G$.

Since $s^{\mathcal{F}} \preceq_S t^{\mathcal{F}}$, $\min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$. Then, there exists t^L such that $s^L \preceq^L t^L$ and $\mathsf{prices}(t^{\mathcal{F}})[t^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]$. By Lemma 14, there exists a π^C-compliant leaf plan π^L_t from t^L to s^L_G such that $\mathsf{cost}(\pi^L_t) \leq \mathsf{cost}(\pi^L_s)$. Consider now the path π^L_2 from $s^L_{\mathcal{I}}$ to s^L_G constructed as the concatenation of: a cheapest $\pi^C(t^{\mathcal{F}})$-compliant path to t^L and π^L_t. By definition, π^L_2 is compliant with $\pi^C(t^{\mathcal{F}}) \circ \pi^C$. Then, $\mathsf{cost}(\pi^L_2) = \mathsf{prices}(t^{\mathcal{F}})[t^L] + \mathsf{cost}(\pi^L_t) \leq \mathsf{prices}(s^{\mathcal{F}})[s^L] + \mathsf{cost}(\pi^L_s) = \mathsf{cost}(\pi^L)$. □

Theorem 36. *$\preceq_S$ subsumes $\preceq$ and is exponentially separated from it.*

Proof. The subsumption holds because $\preceq^L$ is reflexive, so $\min_{s^L \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L] \leq \mathsf{prices}(t^{\mathcal{F}})[s^L]$.

To show the exponential separation, we use again our running example. Leaf simulation captures that $\{(p, l_i)\} \preceq^L \{(p, T)\}$ for all $i \neq 2$. As $\mathsf{prices}(\mathcal{I}^{\mathcal{F}})[\{(p, T)\}] = 1$, $\mathsf{prices}(s^{\mathcal{F}})[\{(p, l_i)\}] = 1$ for $i > 2$ in any decoupled state. Therefore, the size of the decoupled state space when using $\preceq_S$ is linear in the number of locations. □

Theorem 37. *$\preceq_{ES}$ is a decoupled dominance relation.*

Proof. Whenever $s^{\mathcal{F}} \preceq_{ES} t^{\mathcal{F}}$ and a global plan for $s^{\mathcal{F}}$ exists, we can construct an at least as good global plan for $t^{\mathcal{F}}$. Consider any global plan for $s^{\mathcal{F}}$ consisting of a center path π^C, and a goal leaf path π^L for each leaf starting in $s^L_{\mathcal{I}}$ compliant with $\pi^C(s^{\mathcal{F}}) \circ \pi^C$. As $s^{\mathcal{F}} \preceq_{ES} t^{\mathcal{F}}$ implies that $\mathsf{center}(s^{\mathcal{F}}) = \mathsf{center}(t^{\mathcal{F}})$, π^C is applicable to $t^{\mathcal{F}}$ as well. Let s^L be a leaf state traversed by π^L such that the prefix of π^L up to s^L is compliant with $\pi^C(s^{\mathcal{F}})$ and the suffix of π^L starting in s^L, π^L_s, is compliant with π^C. Denote $\pi^L_s = \langle a_1, \dots, a_n \rangle$ and denote the leaf states it traverses by $s^L = s^L_0, \dots, s^L_n = s^L_G$.

By the same arguments as in the proof of Theorem 33, there exists i such that (a) $\mathsf{ESprices}(t^{\mathcal{F}})[s^L_0] \geq \mathsf{ESprices}(t^{\mathcal{F}})[s^L_i] - \sum_{j=1}^{i} \mathsf{cost}(a_j)$, and (b) $\mathsf{ESprices}(t^{\mathcal{F}})[s^L_i] = \min_{s^L_i \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L]$. Thus, there exists t^L such that $\mathsf{prices}(t^{\mathcal{F}})[t^L] \leq \mathsf{prices}(s^{\mathcal{F}})[s^L]+$

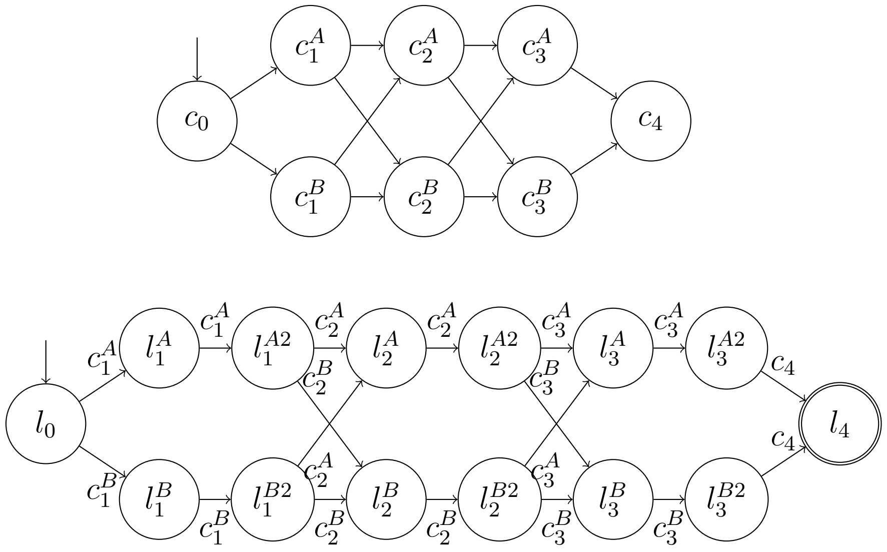


Figure B.1: Illustrative example in which $\preceq_{ES}$ is exponentially better than $\preceq_S$ and $\preceq_E$ for $n = 3$. The center factor is depicted at the top, the leaf at the bottom.

$\sum_{j=1}^{i} \mathsf{cost}(a_j)$. Also, by Lemma 14, there exists a π^C-compliant leaf path π_t^L, from t^L to s_G^L such that $\mathsf{cost}(\pi_t^L) \leq \sum_{j=i+1}^{n} \mathsf{cost}(a_j)$.

We construct our desired path π_2^L from $s_\mathcal{I}^L$ to s_G^L by a cheapest $\pi^C(t^\mathcal{F})$-compliant path to t^L, concatenated with π_t^L. In summary:

$$\begin{aligned}
& \mathsf{cost}(\pi_2^L) \\
= \; & \mathsf{prices}(t^\mathcal{F})[t^L] + \mathsf{cost}(\pi_t^L) \\
\leq \; & \mathsf{prices}(s^\mathcal{F})[s_i^L] + \mathsf{cost}(\pi_t^L) \\
\leq \; & \mathsf{prices}(s^\mathcal{F})[s^L] + \textstyle\sum_{j=1}^{i} \mathsf{cost}(a_j) + \mathsf{cost}(\pi_t^L) \\
\leq \; & \mathsf{prices}(s^\mathcal{F})[s^L] + \mathsf{cost}(\pi_s^L) \\
= \; & \mathsf{cost}(\pi^L)
\end{aligned}$$

□

Theorem 38. *$\preceq_{ES}$ subsumes $\preceq_E$ and $\preceq_S$, and is exponentially separated from each of them.*

Proof. By definition, we have $\mathsf{ESprices}(s^\mathcal{F})[s^L] \leq \mathsf{Eprices}(s^\mathcal{F})[s^L]$ and furthermore $\mathsf{ESprices}(s^\mathcal{F})[s^L] \leq \min_{s^L \preceq^L t^L} \mathsf{prices}(t^\mathcal{F})[t^L]$, so subsumption follows.

To prove the exponential separation, consider the following planning task, in which we have a single center variable c (see top of Figure B.1) with domain $\{c_0, c_{n+1}\} \cup \{c_i^A, c_i^B \mid i \in [1, \dots, n]\}$, arranged such that we can move from c_0 to c_1^X, from c_n^X to c_{n+1}, and from each c_i^X to each c_{i+1}^Y ($X, Y \in \{A, B\}$). The initial value is c_0. We have a single leaf variable l with domain $\{l_0, l_{n+1}\} \cup \{l_i^A, l_i^B, l_i^{A2}, l_i^{B2} \mid i \in [1, \ldots, n]\}$ with initial value l_0 and goal value l_{n+1}. The transitions of the leaf l are depicted in Figure B.1 (bottom). l has transitions from l_0 to l_1^X under center precondition c_1^X, from l_n^{X2} to l_{n+1} under center precondition c_{n+1}, from each l_i^X to l_i^{X2} under center precondition c_i^X, and from each l_i^{X2} to each l_{i+1}^X and l_{i+1}^Y under center precondition c_i^X and c_i^Y, respectively $(X, Y \in \{A, B\}, X \neq Y)$.

Here, each of the individual methods expands a number of states exponential in n:

- With $\preceq$, at each step in the center path, one of the two paths, A or B is chosen. Hence, the pricing function "remembers" which one of the two paths were taken and there is an exponential number of combinations.

- $\preceq_F$ and $\preceq_E$ ignore the prices of l_i^X because they do not belong to the frontier. However, they still remember whether l_i^{A2} or l_i^{B2} has been reached. $\preceq_E$ helps whenever l_n is reached but those states have the largest g-value (equal to $n+1$) so cannot prune any other state.

- $\preceq_S$ finds the following leaf-simulation relation:

$$\begin{aligned}\preceq^L = &\{(l_i^X, l_j^{Y2}), (l_i^{X2}, l_j^{Y2}) \mid i \leq j, X, Y \in A, B\}\\ &\cup \{(x, x), (x, l_n) \mid x \in S^L\}\end{aligned}$$

  In summary, according to $\preceq^L$ states of the form l_i^{A2}, l_i^{B2} are equivalent and they simulate all previous states. However, nothing is found for states l_i^A and l_i^B.

  Therefore, decoupled search with $\preceq_S$ expands the same states than with $\preceq$. The main reason is that, even though it finds the equivalence between l_i^{A2} and l_i^{B2}, depending which path was used, either l_i^A or l_i^B will have a lower price.

  For example, consider the states $s^{\mathcal{F}}$ and $t^{\mathcal{F}}$ reached from the initial decoupled state by following the center transitions $c_0 \rightarrow c_1^A$ and $c_0 \rightarrow c_1^B$, respectively. Then, $\mathsf{prices}(s^{\mathcal{F}})[l_1^A] = 1$, $\mathsf{prices}(s^{\mathcal{F}})[l_1^{A2}] = 2$, $\mathsf{prices}(t^{\mathcal{F}})[l_1^B] = 1$, and $\mathsf{prices}(t^{\mathcal{F}})[l_1^{B2}] = 2$. The initial state has price 0 and all other leaf states ∞. In this case, $s^{\mathcal{F}} \not\preceq_S t^{\mathcal{F}}$ (nor $t^{\mathcal{F}} \not\preceq_S s^{\mathcal{F}}$) because $\mathsf{prices}(s^{\mathcal{F}})[l_1^A] = 1 < \min_{l_1^A \preceq^L t^L} \mathsf{prices}(t^{\mathcal{F}})[t^L] = 2$.

Finally, by using $\preceq_{ES}$, the decoupled state space is polynomial in n. Since $l_i^{A2} \preceq^L l_i^{B2}$ and $l_i^{B2} \preceq^L l_i^{A2}$, $\mathsf{ESprices}(t^{\mathcal{F}})[l_i^{A2}] = \mathsf{ESprices}(t^{\mathcal{F}})[l_i^{B2}]$. Hence, since the only transitions from l_i^A and l_i^B go to l_i^{A2} and l_i^{B2}, $\mathsf{ESprices}(t^{\mathcal{F}})[l_i^A] = \mathsf{ESprices}(t^{\mathcal{F}})[l_i^B] = \mathsf{ESprices}(t^{\mathcal{F}})[l_i^{A2}] - 1$. Therefore, the number of different states according to $\preceq_{ES}$ is polynomial in n. □

Theorem 39. *$\preceq_S$ is exponentially separated from $\preceq_E$, and therefore also from $\preceq_F$. $\preceq_F$, and therefore also $\preceq_E$, is exponentially separated from $\preceq_S$.*

Proof. First, we prove that $\preceq_F$ can be exponentially better than $\preceq_S$. Consider the following planning task, in which we have a single center variable c (see top of Figure B.2) with domain $\{c_1, \ldots, c_n\} \cup \{c_i^A, c_i^B \mid i \in [1, \ldots, n-1]\}$, arranged such that we can move from c_i to c_i^X, and from c_i^X to c_{i+1} ($X \in \{A, B\}$). The initial value is c_1.

We have a single leaf variable l with values $\{l_0, \ldots, l_n\} \cup \{l_i^A l_i^B, l_i^{A2}, l_i^{B2} \mid i \in [1, \ldots, n-1]\}$ with initial value l_0 and goal value l_n. The transitions are depicted in Figure B.2 (middle):

- From l_0 to l_1^X under center precondition c_1^X for $X \in \{A, B\}$.
- From l_{n-1}^{X2} to l_n under center precondition c_n^X for $X \in \{A, B\}$.
- From each l_i^X to l_i^{Y2} under center precondition c_i^X for $i \in [1, \ldots, n-1]$, and $X, Y \in \{A, B\}$.
- From each l_i^{X2} to l_{i+1}^X under center precondition c_{i+1}^X (for $i \in [1, \ldots, n-1]$ and $X \in \{A, B\}$).

In this example, the coarsest simulation relation consist only of $s^L \preceq^L s_G^L$ for all s^L. $a_i \not\preceq^L b_i$ because the center preconditions of their transitions are different. Note that $l_i^A \not\preceq^L l_i^{A2}$ because $l_i^A \xrightarrow{c_i^A} l_{i+1}^{B2}$ and l_i^{A2} does not have any outgoing transition labeled with c_i^A. l_i^{A2} and l_i^{B2} are always reached at the same time, either via l_i^A or l_i^B. The pricing function remembers whether l_i^A or l_i^B was used, so there is an exponential number of decoupled states with the same center state, even under $\preceq_S$ pruning.

However, every time l_i^A or l_i^B are reached, the center precondition c_i^A or c_i^B must be satisfied. Both l_i^{A2} and l_i^{B2} have always a price of $\mathsf{prices}(s^{\mathcal{F}})[l_i^{A2}] = \mathsf{prices}(s^{\mathcal{F}})[l_i^{B2}] \leq \mathsf{prices}(s^{\mathcal{F}})[l_i^X] + 1$ for $X \in \{A, B\}$. Therefore, $l_i^A, l_i^B \notin F(s^{\mathcal{F}})$ for all $s^{\mathcal{F}}$ and i. Since $\mathsf{prices}(s^{\mathcal{F}})[l_i^{A2}] = \mathsf{prices}(s^{\mathcal{F}})[l_i^{B2}]$, there is only a polynomial number of decoupled states when using dominance pruning with $\preceq_F$.

Next, we show that decoupled search with $\preceq_S$ can be exponentially better than decoupled search with $\preceq_E$. Consider a planning task with the same center variable as our previous example. We have a single leaf variable l with values $l_0, \ldots, l_n$, with initial value l_0 and goal value l_n. The transitions of the leaf l are depicted in Figure B.2 (bottom). l has transitions from l_0 to l_1 without any center precondition, from l_0 to l_i under center precondition c_i^A for $i \in [2, n-1]$ and from l_i to l_n under center precondition c_n for $i \in [1, n-1]$.

In this example, decoupled search with $\preceq$ expands an exponential number of states with the same center state because the pricing function remembers whether c_i^A was traversed or not for each $i \in [2, \ldots, n-1]$. Both $\preceq_F$ and $\preceq_E$ cannot reduce the size of the decoupled state space in this example, because l_n is only reached under center precondition c_n, so $l_i \in F(s^{\mathcal{F}})$ for all $0 < i < n$ and $\mathsf{center}(s^{\mathcal{F}}) \neq \{c = c_n\}$.

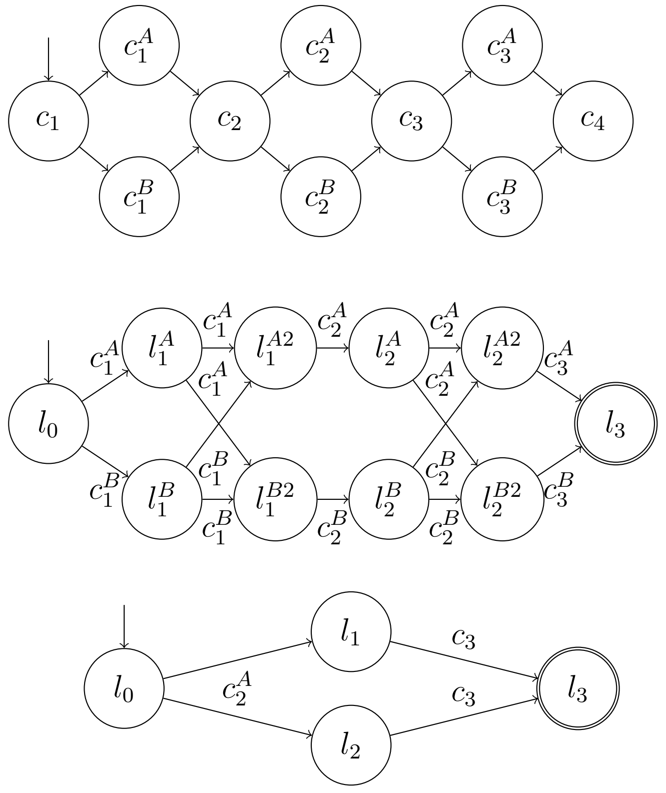


Figure B.2: Illustration of the example of the exponential separations used in the proof of Theorem 39, for $n = 3$. The center factor is depicted in the top, the leaf factor where $\preceq_F$ is exponentially better than $\preceq_S$ in the middle, and Leaf factor where $\preceq_S$ is exponentially better than $\preceq_E$ in the bottom.

However, since all l_i for $i \in [1, n-1]$ have the same outgoing transitions, $l_i \preceq^L l_1$ for all $i \in [0, n-1]$. As the transition from l_0 to l_1 has no center precondition, $\mathsf{prices}(s^{\mathcal{F}})[l_1] = 1$ for any $s^{\mathcal{F}}$. Therefore, $\min_{l_i \preceq^L t^L} \mathsf{prices}(s^{\mathcal{F}})[t^L] = 1$ for $i \in [1, n-1]$. Hence, the number of different decoupled states with $\preceq_S$ is polynomial in n. $\square$

Bibliography

Aghighi, M., Jonsson, P., and Ståhlberg, S. (2015). Tractable cost-optimal planning over restricted polytree causal graphs. In Bonet, B. and Koenig, S., editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3225–3231. AAAI Press.

Alcázar, V. and Torralba, Á. (2015). A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R., Domshlak, C., Haslum, P., and Zilberstein, S., editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 2–6. AAAI Press.

Alkhazraji, Y., Wehrle, M., Mattmüller, R., and Helmert, M. (2012). A stubborn set algorithm for optimal planning. In Raedt, L. D., editor, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pages 891–892, Montpellier, France. IOS Press.

Alrahman, Y. A., Andric, M., Beggiato, A., and Lluch-Lafuente, A. (2014). Can we efficiently check concurrent programs under relaxed memory models in maude? In *Revised Selected Papers of the 10th International Workshop on Rewriting Logic and Its Applications (WRLA'14)*, pages 21–41.

Amir, E. and Engelhardt, B. (2003). Factored planning. In Gottlob, G., editor, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 929–935, Acapulco, Mexico. Morgan Kaufmann.

Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Kretínský, J., Müller, D., Parker, D., and Strejcek, J. (2015). The hanoi omega-automata format. volume 9206 of *Lecture Notes in Computer Science*, pages 479–486. Springer.

Bacchus, F. and Yang, Q. (1994). Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100.

Bäckström, C. and Nebel, B. (1995). Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655.

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., and Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206.

Baldan, P., Bruni, A., Corradini, A., König, B., Rodríguez, C., and Schwoon, S. (2012). Efficient unfolding of contextual Petri nets. *Theoretical Computer Science*, 449:2–22.

Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33.

Bonet, B., Haslum, P., Hickmott, S. L., and Thiébaux, S. (2008). Directed unfolding of petri nets. *Transactions on Petri Nets and Other Models of Concurrency*, 1:172–198.

Bonet, B., Haslum, P., Khomenko, V., Thiébaux, S., and Vogler, W. (2014). Recent advances in unfolding technique. *Theoretical Computer Science*, 551:84–101.

Brafman, R. and Domshlak, C. (2003). Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, 18:315–349.

Brafman, R. and Domshlak, C. (2006). Factored planning: How, when, and when not. In Gil, Y. and Mooney, R. J., editors, *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI'06)*, pages 809–814, Boston, Massachusetts, USA. AAAI Press.

Brafman, R. and Domshlak, C. (2013). On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71.

Brafman, R. I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J., Nebel, B., Beck, J. C., and Hansen, E., editors, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 28–35. AAAI Press.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204.

Ciardo, G. and Siminiceanu, R. (2002). Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Formal Methods in Computer-Aided Design, 4th International Conference, FMCAD 2002, Portland, OR, USA, November 6-8, 2002, Proceedings*, pages 256–273.

Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An opensource tool for symbolic model checking. In Brinksma, E. and Larsen, K. G., editors, *Computer Aided Verification, 14th International Conference, CAV 2002, Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer.

Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1–2):35–84.

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2003). Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the Association for Computing Machinery*, 50(5):752–794.

Clarke, E., Grumberg, O., and Peled, D. (2001). *Model Checking*. MIT Press.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263.

Clarke, E. M., Filkorn, T., and Jha, S. (1993). Exploiting symmetry in temporal logic model checking. In Courcoubetis, C., editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 450–462. Springer.

Clarke, E. M., Long, D. E., and McMillan, K. L. (1989). Compositional model checking. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science (LICS '89)*, pages 353–362.

Cobleigh, J. M., Giannakopoulou, D., and Pasareanu, C. (2003). Learning assumptions for compositional verification. In Garavel, H. and Hatcliff, J., editors, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, pages 331–346. Springer-Verlag.

Courcoubetis, C., Vardi, M. Y., Wolper, P., and Yannakakis, M. (1992). Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2/3):275–288.

Couvreur, J. (1999). On-the-fly verification of linear temporal logic. In Wing, J. M., Woodcock, J., and Davies, J., editors, *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20-24, 1999, Proceedings, Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 253–271. Springer.

Crosby, M., Rovatsos, M., and Petrick, R. P. A. (2013). Automated agent decomposition for classical planning. In Borrajo, D., Fratini, S., Kambhampati, S., and Oddi, A., editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy. AAAI Press.

Domshlak, C. and Dinitz, Y. (2001). Multi-agent offline coordination: Structure and complexity. In Cesta, A. and Borrajo, D., editors, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 34–43. Springer-Verlag.

Domshlak, C., Hoffmann, J., and Katz, M. (2015a). Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, 221:73–114.

Domshlak, C., Katz, M., and Shleyfman, A. (2012). Enhanced symmetry breaking in cost-optimal planning as forward search. In Bonet, B., McCluskey, L., Silva, J. R., and Williams, B., editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Domshlak, C., Katz, M., and Shleyfman, A. (2013). Symmetry breaking: Satisficing planning and landmark heuristics. In Borrajo, D., Fratini, S., Kambhampati, S., and Oddi, A., editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy. AAAI Press.

Domshlak, C., Katz, M., and Shleyfman, A. (2015b). Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. *Technical Report IS/IE-2015-02*.

Doran, J. E. and Michie, D. (1966). Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437):235–259.

Dräger, K., Finkbeiner, B., and Podelski, A. (2006). Directed model checking with distance-preserving abstractions. In Valmari, A., editor, *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, pages 19–34. Springer-Verlag.

Edelkamp, S. (2003a). Promela planning. In Ball, T. and Rajamani, S., editors, *Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN-03)*, pages 197–212, Portland, OR. Springer-Verlag.

Edelkamp, S. (2003b). Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, 20:195–238.

Edelkamp, S. and Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S. and Fox, M., editors, *Proceedings* *of the 5th European Conference on Planning (ECP'99)*, pages 135–147. Springer-Verlag.

Edelkamp, S. and Kissmann, P. (2008). Limits and possibilities of bdds in state space search. In Fox, D. and Gomes, C., editors, *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, pages 1452–1453, Chicago, Illinois, USA. AAAI Press.

Edelkamp, S. and Kissmann, P. (2011). On the complexity of BDDs for state space search: A case study in connect four. In Burgard, W. and Roth, D., editors, *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, San Francisco, CA, USA. AAAI Press.

Edelkamp, S., Leue, S., and Lluch-Lafuente, A. (2004a). Partial-order reduction and trail improvement in directed model checking. *International Journal on Software Tools for Technology Transfer*, 6(4):277–301.

Edelkamp, S., Lluch-Lafuente, A., and Leue, S. (2001). Directed explicit model checking with hsf-spin. In Vardi, M. Y., Dwyer, M. B., and Chechik, M., editors, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN-01)*, pages 57–79, Toronto, Canada. Springer-Verlag.

Edelkamp, S., Lluch-Lafuente, A., and Leue, S. (2004b). Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology Transfer*, 5(2-3):247–267.

Emerson, E. A. and Sistla, A. P. (1993). Symmetry and model checking. In Courcoubetis, C., editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 463–478. Springer.

Emerson, E. A. and Sistla, A. P. (1996). Symmetry and model-checking. *Formal Methods in System Design*, 9(1/2):105–131.

Ernst, G. W. and Newell, A. (1969). *GPS: A Case Study in Generality and Problem-Solving*. Academic Press, New York, NY.

Esparza, J. and Heljanko, K. (2000). A new unfolding approach to LTL model checking. volume 1853 of *Lecture Notes in Computer Science*, pages 475–486. Springer.

Esparza, J. and Heljanko, K. (2001). Implementing LTL model checking with net unfoldings. In Vardi, M. Y., Dwyer, M. B., and Chechik, M., editors, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN-01)*, pages 37–56, Toronto, Canada. Springer-Verlag.

Esparza, J., Römer, S., and Vogler, W. (2002). An improvement of mcmillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310.

Fabre, E., Jezequel, L., Haslum, P., and Thiébaux, S. (2010). Cost-optimal factored planning: Promises and pitfalls. In Brafman, R. I., Geffner, H., Hoffmann, J., and Kautz, H. A., editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 65–72. AAAI Press.

Fickert, M., Gnad, D., and Hoffmann, J. (2018a). Unchaining the power of partial delete relaxation, part II: finding plans with red-black state space search. In Lang, J., editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 4750–4756.

Fickert, M., Gnad, D., Speicher, P., and Hoffmann, J. (2018b). Saarplan: Combining Saarland's greatest planning techniques. In *IPC 2018 planner abstracts*.

Fikes, R. E. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Fišer, D., Gnad, D., Katz, M., and Hoffmann, J. (2021). Custom-design of FDR encodings: The case of red-black planning. In Zhou, Z.-H., editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 4054–4061.

Fortune, S., Hopcroft, J., and Wyllie, J. (1980). The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121.

Fox, M. and Long, D. (1999). The detection and exploitation of symmetry in planning problems. In Pollack, M., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 956–961, Stockholm, Sweden. Morgan Kaufmann.

Francès, G., Lipovetzky, N., Geffner, H., and Ramírez, M. (2018). Best-first width search in the IPC 2018: Complete, simulated, and polynomial variants. In *IPC 2018 planner abstracts*.

Franco, S., Lelis, L. H., and Barley, M. (2018). The complementary2 planner in the IPC 2018. In *IPC 2018 planner abstracts*.

Franco, S., Torralba, A., Lelis, L. H., and Barley, M. (2017). On creating complementary pattern databases. In Sierra, C., editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 4302–4309. AAAI Press/IJCAI.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

Geldenhuys, J. and Valmari, A. (2004). Tarjan's algorithm makes on-the-fly LTL verification more efficient. volume 2988 of *Lecture Notes in Computer Science*, pages 205–219. Springer.

Gentilini, R., Piazza, C., and Policriti, A. (2003). From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning*, 31(1):73–103.

Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, 20:239–290.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Giménez, O. and Jonsson, A. (2008). The complexity of planning problems with simple causal graphs. *Journal of Artificial Intelligence Research*, 31:319–351.

Giménez, O. and Jonsson, A. (2009). Planning over chain causal graphs for variables with domains of size 5 is NP-hard. *Journal of Artificial Intelligence Research*, 34:675–706.

Giménez, O. and Jonsson, A. (2012). The influence of k-dependence on the complexity of planning. *Artificial Intelligence*, 177-179:25–45.

Gnad, D. (2021a). Code and Evaluation Results of the Work "Star-Topology Decoupled State-Space Search for AI Planning and Model Checking". `https://zenodo.org/record/5230126`.

Gnad, D. (2021b). Revisiting dominance pruning in decoupled search. In Leyton-Brown, K. and Mausam, editors, *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21)*, pages 11809–11817. AAAI Press.

Gnad, D., Dubbert, P., Lluch-Lafuente, A., and Hoffmann, J. (2018a). Star-topology decoupling in SPIN. In del Mar Gallardo, M. and Merino, P., editors, *Proceedings of the 25th International Symposium on Model Checking of Software (SPIN'18)*, Lecture Notes in Computer Science. Springer.

Gnad, D., Dubbert, P., Lluch-Lafuente, A., and Hoffmann, J. (2021a). Code and Benchmark Models of the SPIN 2018 paper "Star-Topology Decoupling in SPIN". `https://zenodo.org/record/5229954`.

Gnad, D., Eisenhut, J., Lluch Lafuente, A., and Hoffmann, J. (2021b). Code and Benchmark Models of the CAV'21 paper "Model Checking ω-Regular Properties with Decoupled Search". `https://zenodo.org/record/4501646`.

Gnad, D., Eisenhut, J., Lluch-Lafuente, A., and Hoffmann, J. (2021c). Model checking ω-regular properties with decoupled search. In Silva, A. and Leino, K. R. M., editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*, volume 12760 of *Lecture Notes in Computer Science*, pages 411–434. Springer.

Gnad, D. and Hoffmann, J. (2015a). Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Brafman, R., Domshlak, C., Haslum, P., and Zilberstein, S., editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 88–96. AAAI Press.

Gnad, D. and Hoffmann, J. (2015b). Red-black planning: A new tractability analysis and heuristic function. In Lelis, L. and Stern, R., editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*. AAAI Press.

Gnad, D. and Hoffmann, J. (2018). Star-topology decoupled state space search. *Artificial Intelligence*, 257:24 – 60.

Gnad, D. and Hoffmann, J. (2019). On the relation between star-topology decoupling and petri net unfolding. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*, pages 172–180. AAAI Press.

Gnad, D., Hoffmann, J., and Domshlak, C. (2015). From fork decoupling to star-topology decoupling. In Lelis, L. and Stern, R., editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, pages 53–61. AAAI Press.

Gnad, D., Hoffmann, J., and Wehrle, M. (2019a). Strong stubborn set pruning for star-topology decoupled state space search. *Journal of Artificial Intelligence Research*, 65:343–392.

Gnad, D., Poser, V., and Hoffmann, J. (2017a). Beyond forks: Finding and ranking star factorings for decoupled search. In Sierra, C., editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 4310–4316. AAAI Press/IJCAI.

Gnad, D., Shleyfman, A., and Hoffmann, J. (2018b). DecStar - star-topology decoupled search at its best. In *IPC 2018 planner abstracts*.

Gnad, D., Steinmetz, M., and Hoffmann, J. (2016a). Django: Unchaining the power of red-black planning. In *UIPC 2016 planner abstracts*, pages 19–23.

Gnad, D., Steinmetz, M., Jany, M., Hoffmann, J., Serina, I., and Gerevini, A. (2016b). Partial delete relaxation, unchained: On intractable red-black planning and its applications. In Baier, J. and Botea, A., editors, *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS'16)*. AAAI Press.

Gnad, D., Torralba, Á., Domínguez, M., Areces, C., and Bustos, F. (2019b). IPA LAMA: Planner abstract. In *Sparkle Planning Challenge 2019*.

Gnad, D., Torralba, Á., Domínguez, M., Areces, C., and Bustos, F. (2019c). Learning how to ground a plan – partial grounding in classical planning. In Hentenryck, P. V. and Zhou, Z.-H., editors, *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, pages 7602–7609. AAAI Press.

Gnad, D., Torralba, Á., and Hoffmann, J. (2017b). Symbolic leaf representation in decoupled search. In Fukunaga, A. and Kishimoto, A., editors, *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*. AAAI Press.

Gnad, D., Torralba, Á., Hoffmann, J., and Wehrle, M. (2016c). Decoupled search for proving unsolvability. In *UIPC 2016 planner abstracts*, pages 16–18.

Gnad, D., Torralba, Á., Shleyfman, A., and Hoffmann, J. (2017c). Symmetry breaking in star-topology decoupled search. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, pages 125–134. AAAI Press.

Gnad, D., Wehrle, M., and Hoffmann, J. (2016d). Decoupled strong stubborn sets. In Kambhampati, S., editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 3110–3116. AAAI Press/IJCAI.

Godefroid, P. (1996). *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer.

Godefroid, P. and Wolper, P. (1991). Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*, pages 332–342.

Groote, J. F. and Sellink, M. P. A. (1995). Confluence for process verification. In Lee, I. and Smolka, S. A., editors, *CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings*, volume 962 of *Lecture Notes in Computer Science*, pages 204–218. Springer.

Groote, J. F. and van de Pol, J. (2000). State space reduction using partial tau-confluence. In Nielsen, M. and Rovan, B., editors, *Mathematical Foundations of*

Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings, volume 1893 of *Lecture Notes in Computer Science*, pages 383–393. Springer.

Hall, D., Cohen, A., Burkett, D., and Klein, D. (2013). Faster optimal planning with partial-order pruning. In Borrajo, D., Fratini, S., Kambhampati, S., and Oddi, A., editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy. AAAI Press.

Hansen, E. A., Zhou, R., and Feng, Z. (2002). Symbolic heuristic search using decision diagrams. In Koenig, S. and Holte, R. C., editors, *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, volume 2371 of *Lecture Notes in Computer Science*, pages 83–98. Springer.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Haslum, P. (2007). Reducing accidental complexity in planning problems. In Veloso, M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1898–1903, Hyderabad, India. Morgan Kaufmann.

Haslum, P., Botea, A., Helmert, M., Bonet, B., and Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In Howe, A. and Holte, R. C., editors, *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, pages 1007–1012, Vancouver, BC, Canada. AAAI Press.

Haslum, P. and Geffner, H. (2000). Admissible heuristics for optimal planning. In Chien, S., Kambhampati, R., and Knoblock, C., editors, *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 140–149, Breckenridge, CO. AAAI Press, Menlo Park.

Haslum, P., Lipovetzky, N., Magazzeni, D., and Muise, C. (2019). *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Koenig, S., Zilberstein, S., and Koehler, J., editors, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 161–170, Whistler, Canada. AAAI Press.

Helmert, M. (2006a). Fast (diagonally) downward. In *IPC 2006 planner abstracts*.

Helmert, M. (2006b). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535.

Helmert, M. and Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A., Howe, A., Cesta, A., and Refanidis, I., editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press.

Helmert, M., Haslum, P., Hoffmann, J., and Nissim, R. (2014). Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, 61(3):16:1–16:63.

Henzinger, M. R., Henzinger, T. A., and Kopke, P. W. (1995). Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 453–462. IEEE Computer Society.

Hickmott, S. L., Rintanen, J., Thiébaux, S., and White, L. B. (2007). Planning via petri net unfolding. In Veloso, M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1904–1911, Hyderabad, India. Morgan Kaufmann.

Hoffmann, J., Kissmann, P., and Torralba, Á. (2014). "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T., editor, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, pages 441–446, Prague, Czech Republic. IOS Press.

Hoffmann, J. and Kupferschmid, S. (2005). A covering problem for hypercubes. In Kaelbling, L. P., editor, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 1523–1524, Edinburgh, UK. Morgan Kaufmann.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Hoffmann, J., Porteous, J., and Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278.

Holzmann, G. (2004). *The Spin Model Checker - Primer and Reference Manual*. Addison-Wesley.

Holzmann, G. J. and Peled, D. A. (1994). An improvement in formal verification. In Hogrefe, D. and Leue, S., editors, *Formal Description Techniques VII, Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques, Berne, Switzerland, 1994*, volume 6 of *IFIP Conference Proceedings*, pages 197–211. Chapman & Hall.

Holzmann, G. J., Peled, D. A., and Yannakakis, M. (1996). On nested depth first search. In Grégoire, J., Holzmann, G. J., and Peled, D. A., editors, *The Spin Verification System, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, August, 1996*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 23–31. DIMACS/AMS.

Ip, C. N. and Dill, D. L. (1996). Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–75.

Jensen, R. M., Veloso, M. M., and Bryant, R. E. (2008). State-set branching: Leveraging BDDs for heuristic search. *Artificial Intelligence*, 172(2-3):103–139.

Jonsson, B. (2008). State-space exploration for concurrent algorithms under weak memory orderings. *SIGARCH Computer Architecture News*, 36(5):65–71.

Jonsson, P. and Bäckström, C. (1995). Incremental planning. In *European Workshop on Planning*.

Jøsang, A. (1995). Security protocol verication using spin. In *The First SPIN Workshop, Montreal, Quebec, Canada*.

Junttila, T. and Kaski, P. (2007). Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007)*, pages 135–149. SIAM.

Karpas, E. and Domshlak, C. (2009). Cost-optimal planning with landmarks. In Boutilier, C., editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1728–1733, Pasadena, California, USA. Morgan Kaufmann.

Katz, M. and Domshlak, C. (2008). New islands of tractability of cost-optimal planning. *Journal of Artificial Intelligence Research*, 32:203–288.

Katz, M. and Hoffmann, J. (2014). Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, pages 43–47.

Katz, M. and Keyder, E. (2012). Structural patterns beyond forks: Extending the complexity boundaries of classical planning. In Hoffmann, J. and Selman, B., editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 1779–1785, Toronto, ON, Canada. AAAI Press.

Katz, M., Lipovetzky, N., Moshkovich, D., and Tuisov, A. (2018). MERWIN planner: Mercury enchanced with novelty heuristic. In *IPC 2018 planner abstracts*, pages 53–56.

Kelareva, E., Buffet, O., Huang, J., and Thiébaux, S. (2007). Factored planning using decomposition trees. In Veloso, M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1942–1947, Hyderabad, India. Morgan Kaufmann.

Kissmann, P. and Edelkamp, S. (2011). Improving cost-optimal domain-independent symbolic planning. In Burgard, W. and Roth, D., editors, *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, pages 992–997, San Francisco, CA, USA. AAAI Press.

Knoblock, C. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302.

Kronegger, M., Ordyniak, S., and Pfandler, A. (2014). Backdoors to planning. In Brodley, C. E. and Stone, P., editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, pages 2300–2307, Austin, Texas, USA. AAAI Press.

Kronegger, M., Ordyniak, S., and Pfandler, A. (2015). Variable-deletion backdoors to planning. In Bonet, B. and Koenig, S., editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3305–3312. AAAI Press.

Kupferman, O., Vardi, M. Y., and Wolper, P. (2000). An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360.

Kupferschmid, S., Hoffmann, J., Dierks, H., and Behrmann, G. (2006). Adapting an AI planning heuristic for directed model checking. In Valmari, A., editor, *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, pages 35–52. Springer-Verlag.

Kupferschmid, S., Hoffmann, J., Dräger, K., Finkbeiner, B., Dierks, H., Podelski, A., and Behrmann, G. (2007). Uppaal/DMC – abstraction-based heuristics for directed model checking. In Grumberg, O. and Huth, M., editors, *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*. Springer-Verlag.

Kupferschmid, S., Hoffmann, J., and Larsen, K. G. (2008). Fast directed model checking via Russian doll abstraction. In Ramakrishnan, C. R. and Rehof, J., editors, *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, pages 203–217. Springer-Verlag.

Kwiatkowska, M. Z., Norman, G., and Parker, D. (2011). Prism 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G. and Qadeer, S., editors, *Proceedings of the 23rd International on Conference Computer Aided Verification (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer.

Laarman, A., Olesen, M. C., Dalsgaard, A. E., Larsen, K. G., and van de Pol, J. (2013). Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 968–983. Springer.

Lansky, A. L. and Getoor, L. (1995). Scope and abstraction: Two criteria for localized planning. In Mellish, S., editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1612–1619, Montreal, Canada. Morgan Kaufmann.

Lichtenstein, O. and Pnueli, A. (1985). Checking that finite state concurrent programs satisfy their linear specification. pages 97–107. ACM Press.

Linden, A. and Wolper, P. (2013). A verification-based approach to memory fence insertion in PSO memory systems. In Piterman, N. and Smolka, S. A., editors, *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, pages 339–353. Springer-Verlag.

Lipovetzky, N. and Geffner, H. (2012). Width and serialization of classical planning problems. In Raedt, L. D., editor, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pages 540–545, Montpellier, France. IOS Press.

Lipovetzky, N. and Geffner, H. (2017). Best-first width search: Exploration and exploitation in classical planning. In Singh, S. and Markovitch, S., editors, *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 3590–3596. AAAI Press.

Liu, M. T. (1989). Protocol engineering. In *Advances in computers*, volume 29, pages 79–195. Elsevier.

Long, D. and Fox, M. (1999). Efficient implementation of the plan graph in stan. *Journal of Artificial Intelligence Research*, 10:87–115.

Luks, E. M. (1993). Permutation groups and polynomial-time computation. In *Groups and Computation, DIMACS Series in Disc. Math. and Th. Comp. Sci.*, volume 11, pages 139–175.

Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann.

McCarthy, J. (1959). Programs with common sense. In *Proceedings of the Symposium on Mechanisation of the Thought Process I.*

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee.

McDermott, D. V. (1999). Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1–2):111–159.

McMillan, K. L. (1992). Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In von Bochmann, G. and Probst, D. K., editors, *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV'92)*, Lecture Notes in Computer Science, pages 164–177. Springer.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers.

Milner, R. (1971). An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71)*, pages 481–489, London, UK. William Kaufmann.

Moura, L. D. and Bjørner, N. (2008). Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, pages 337–340.

Newell, A. and Simon, H. (1963). GPS, a program that simulates human thought. In Feigenbaum, E. and Feldman, J., editors, *Computers and Thought*, pages 279–293. McGraw-Hill.

Nissim, R. and Brafman, R. (2014). Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research*, 51:293–332.

Nissim, R., Brafman, R. I., and Domshlak, C. (2010). A general, fully distributed multi-agent planning algorithm. In van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., and Sen, S., editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1323–1330. IFAAMAS.

Pearl, J. (1984). *Heuristics*. Morgan Kaufmann.

Peled, D. (1993). All from one, one for all: on model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV'93)*, pages 409–423.

Peled, D. A. (1996). Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS'77)*.

Pnueli, A. (1985). In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, volume 13, pages 123–144.

Pochter, N., Zohar, A., and Rosenschein, J. S. (2011). Exploiting problem symmetries in state-based planners. In Burgard, W. and Roth, D., editors, *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, San Francisco, CA, USA. AAAI Press.

Podkopaev, A., Lahav, O., and Vafeiadis, V. (2019). Bridging the gap between programming languages and hardware weak memory models. *Proc. ACM Program. Lang.*, 3(POPL):69:1–69:31.

Pommerening, F. and Helmert, M. (2015). A normal form for classical planning tasks. In Brafman, R., Domshlak, C., Haslum, P., and Zilberstein, S., editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 188–192. AAAI Press.

Pommerening, F., Röger, G., and Helmert, M. (2013). Getting the most out of pattern databases for classical planning. In Rossi, F., editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. AAAI Press/IJCAI.

PromelaManual (2020). Promela manual pages. `http://spinroot.com/spin/Man/promela.html`.

Richter, S. and Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A., Howe, A., Cesta, A., and Refanidis, I., editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 273–280. AAAI Press.

Richter, S., Helmert, M., and Westphal, M. (2008). Landmarks revisited. In Fox, D. and Gomes, C., editors, *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, pages 975–982, Chicago, Illinois, USA. AAAI Press.

Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.

Richter, S., Westphal, M., and Helmert, M. (2011). LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, pages 50–54.

Rintanen, J. (2003). Symmetry reduction for SAT representations of transition systems. In Giunchiglia, E., Muscettola, N., and Nau, D., editors, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 32–41, Trento, Italy. AAAI Press.

Rodríguez, C. and Schwoon, S. (2013). Cunf: A tool for unfolding and verifying petri nets with read arcs. In *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, pages 492–495.

Röger, G., Helmert, M., Seipp, J., and Sievers, S. (2020). An atom-centric perspective on stubborn sets. In Harabor, D. and Vallati, M., editors, *Proceedings of the Thirteenth International Symposium on Combinatorial Search, SOCS 2020, Online Conference [Vienna, Austria], 26-28 May 2020*, pages 57–65. AAAI Press.

Roggenbach, M. (2001). Determinization of büchi-automata. In Grädel, E., Thomas, W., and Wilke, T., editors, *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, pages 43–60. Springer.

Rudin, H. (1987). Network protocols and tools to help produce them. *Annual Review of Computer Science*, 2(1):291–316.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135.

Schmitt, F. (2018). Finding star factorings using integer linear programming. *Bachelors Thesis at Saarland University*.

Schmitt, F., Gnad, D., and Hoffmann, J. (2019). Advanced factoring strategies for decoupled search using linear programming. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*. AAAI Press.

Schwoon, S. and Esparza, J. (2005). A note on on-the-fly verification algorithms. volume 3440 of *Lecture Notes in Computer Science*, pages 174–190. Springer.

Seipp, J. and Helmert, M. (2013). Counterexample-guided Cartesian abstraction refinement. In Borrajo, D., Fratini, S., Kambhampati, S., and Oddi, A., editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pages 347–351, Rome, Italy. AAAI Press.

Seipp, J., Pommerening, F., Sievers, S., and Helmert, M. (2017). Downward Lab. `https://doi.org/10.5281/zenodo.790461`.

Shleyfman, A., Katz, M., Helmert, M., Sievers, S., and Wehrle, M. (2015). Heuristics and symmetries in classical planning. In Bonet, B. and Koenig, S., editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3371–3377. AAAI Press.

Sievers, S. and Wehrle, M. (2021). On weak stubborn sets in classical planning. In Zhou, Z.-H., editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 4167–4174.

Sievers, S., Wehrle, M., Helmert, M., Shleyfman, A., and Katz, M. (2015). Factored symmetries for merge-and-shrink abstractions. In Bonet, B. and Koenig, S., editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3378–3385. AAAI Press.

Sistla, A. P., Miliades, L., and Gyuris, V. (1997). SMC: A symmetry based model checker for verification of liveness properties. In Grumberg, O., editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, Lecture Notes in Computer Science, pages 464–467. Springer.

Somenzi, F. (2021). CUDD: CU decision diagram package release 3.0.0. At `https://github.com/ivmai/cudd`.

Speck, D., Geißer, F., and Mattmüller, R. (2018). Symbolic planning with edge-valued multi-valued decision diagrams. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, pages 250–258. AAAI Press.

Speicher, P., Steinmetz, M., Gnad, D., Hoffmann, J., and Gerevini, A. (2017). Beyond red-black planning: Limited-memory state variables. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, pages 269–273. AAAI Press.

Starke, P. (1991). Reachability analysis of petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in Systems Analysis*, 8(4/5):293–304.

Tauriainen, H. (2006). Nested emptiness search for generalized büchi automata. *Fundamenta Informaticae*, 70(1-2):127–154.

Torralba, Á. (2016). Sympa: Symbolic perimeter abstractions for proving unsolvability. In *UIPC 2016 planner abstracts*, pages 8–11.

Torralba, Á. (2017). From qualitative to quantitative dominance pruning for optimal planning. In Sierra, C., editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 4426–4432. AAAI Press/IJCAI.

Torralba, Á. (2018). Completeness-preserving dominance techniques for satisficing planning. In Lang, J., editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 4844–4851.

Torralba, Á., Alcázar, V., Borrajo, D., Kissmann, P., and Edelkamp, S. (2014). SymBA*: A symbolic bidirectional A* planner. In *IPC 2014 planner abstracts*, pages 105–109.

Torralba, Á., Alcázar, V., Kissmann, P., and Edelkamp, S. (2017). Efficient symbolic search for cost-optimal planning. *Artificial Intelligence*, 242:52–79.

Torralba, Á., Gnad, D., Dubbert, P., and Hoffmann, J. (2016). On state-dominance criteria in fork-decoupled search. In Kambhampati, S., editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 3265–3271. AAAI Press/IJCAI.

Torralba, Á. and Hoffmann, J. (2015). Simulation-based admissible dominance pruning. In Yang, Q., editor, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1689–1695. AAAI Press/IJCAI.

Torralba, Á. and Kissmann, P. (2015). Focusing on what really matters: Irrelevance pruning in merge-and-shrink. In Lelis, L. and Stern, R., editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, pages 122–130. AAAI Press.

Tozicka, J., Jakubuv, J., Svatos, M., and Komenda, A. (2016). Recursive polynomial reductions for classical planning. In Coles, A., Coles, A., Edelkamp, S., Magazzeni, D., and Sanner, S., editors, *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*, pages 317–325. AAAI Press.

Travkin, O., Mütze, A., and Wehrheim, H. (2013). SPIN as a linearizability checker under weak memory models. In *Proceedings of the 9th International Haifa Verification Conference (HVC'13)*, pages 311–326.

Valmari, A. (1989). Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, pages 491–515.

Valmari, A. (1992). A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322.

Wang, D. and Williams, B. C. (2015). tburton: A divide and conquer temporal planner. In Bonet, B. and Koenig, S., editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3409–3417. AAAI Press.

Wehrle, M. and Helmert, M. (2012). About partial order reduction in planning and computer aided verification. In Bonet, B., McCluskey, L., Silva, J. R., and Williams, B., editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Wehrle, M. and Helmert, M. (2014). Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S., Do, M., Fern, A., and Ruml, W., editors, *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

Wehrle, M., Helmert, M., Alkhazraji, Y., and Mattmüller, R. (2013). The relative pruning power of strong stubborn sets and expansion core. In Borrajo, D., Fratini, S., Kambhampati, S., and Oddi, A., editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy. AAAI Press.

Wehrle, M., Helmert, M., Shleyfman, A., and Katz, M. (2015). Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In Yang, Q., editor, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press/IJCAI.

West, C. H. (1978). General technique for communications protocol validation. *IBM Journal of Research and Development*, 22(4):393–404.

Winterer, D., Alkhazraji, Y., Katz, M., and Wehrle, M. (2017). Stubborn sets for fully observable nondeterministic planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.

Zhu, L. and Givan, R. (2003). Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, pages 156–160.