Linköping Studies in Science and Technology
Dissertation No. 2504

Computing Perfect Cost Partitioning Heuristics for Classical Planning

Paul Höft

LINKÖPING UNIVERSITY

Linköping Studies in Science and Technology
Dissertations, No. 2504

Computing Perfect Cost Partitioning Heuristics for Classical Planning

Paul Höft

Linköping University
Department of Computer and Information Science
Division of Artificial Intelligence and Integrated Computer Systems
SE-581 83 Linköping, Sweden

Linköping 2026

Typeset using XeTeX

Printed by LiU-Tryck, Linköping 2026

Edition 1:1

ISBN 978–91–8118–452–5 (print)
ISBN 978–91–8118–453–2 (PDF)
https://doi.org/10.3384/9789181184532
ISSN 0345-7524

POPULÄRVETENSKAPLIG SAMMANFATTNING

Föreställ dig en rymdströvare som utforskar ytan på Mars. För varje beslut den tar, förbrukas dyrbara resurser och ett suboptimalt beteende kan hindra den från att uppnå sina vetenskapliga mål. I scenarier med höga insatser, som i detta fall, är det inte nog att uppfylla målen ”tillräckligt bra”, utan vi behöver lösningar som bevisligen är de bästa. Detta är en utav huvudtillämpningarna av en gren inom artificiell intelligens som heter ”optimal klassisk planering”. Given en formell problembeskrivning beräknar vi en ”plan”, vilket är en sekvens av åtgärder som uppnår ett specifikt mål. Denna plan garanteras vara optimal med hänsyn till resursförbrukning, exempelvis energiförbrukning eller komponentslitage. Till skillnad från approximativa metoder som introducerar en osäkerhet genom att nöja sig med suboptimala lösningar, ger optimal planering säkerhet - ett kritiskt krav när resurserna är knappa och misstag är kostsamma. Att hitta optimala lösningar är dock svårt. Planeringsalgoritmer använder sig av heuristiker som uppskattar det optimala avståndet till målet. Om heuristiken aldrig överskattar det optimala avståndet, det vill säga är en ”tillåtande” heuristik, garanteras det att optimala lösningar kan upptäckas. Tillåtande heuristiker är inte unika, därför är det ofta fördelaktigt att kombinera flera heuristiker av sådan kategori. Tyvärr är det inte trivialt att kombinera heuristiker på ett sådan sätt att kombinationen i sig är tillåtande. Om flera heuristiker beaktar samma åtgärd i sina individuella uppskattningar kommer denna åtgärd att räknas flera gånger, vilket ökar risken för överskattning av det optimala avståndet. En lösning är kostnadsfördelning, där åtgärdskostnaderna fördelas mellan flera heuristiker sådan att de alla förblir informativa samtidigt som deras uppskattningar kan summeras utan att riskera överskattning. Detta arbete utforskar flertalet förbättringar av framgångsrika kostnadsfördelningsalgoritmer, vilket möjliggör en snabbare beräkning av tillåtande heuristiker, och som fördjupar vår förståelse av dem. Detta kommer i förlängningen att förbättra beslutsfattningen och planeringen i situationer där optimalitet är avgörande.

ABSTRACT

Optimal classical planning aims to find minimal-cost action sequences in deterministic, fully observable problems. Strong domain-independent admissible heuristics are crucial for optimal A* searches, and cost partitioning is one of the most powerful techniques for generating them. However, computing optimal cost partitions is challenging because it requires solving a linear program for each encountered state. There exist two strategies to make cost partitioning more practical. First there exist non-optimal cost partitioning strategies that offer weaker guidance but are easier to compute. Second, cost partitioning heuristics are often approximated by not computing one for every encountered state.

This thesis investigates how efficiently the non-approximative, say perfect, versions of non-optimal cost partitioning strategies can be computed, which helps to understand their true practical complexity and limits. We show that the perfect variant of two common cost partitioning strategies, saturated cost partitioning (SCP) and saturated post-hoc optimization (SPhO), can be computed much more efficiently than the naive strategy.

SPhO computes cost partitions by solving a linear program for each state encountered during the search, which is prohibitively expensive in practice. We introduce cover rules based on the sensitivity analysis of linear programs that enable the reuse of cost partitions across states without sacrificing heuristic quality. This drastically reduces the number of linear program solver calls while preserving optimality. We also analyze the structure of the saturated post-hoc optimization linear program, including degeneracy and non-uniqueness, and propose an algorithm to maximize cost partition reusability.

SCP does not require a linear program to be solved and is cheaper to compute. However, its quality depends heavily on the ordering of component heuristics. The best SCP heuristic would maximize over all possible orders, but this is infeasible due to factorial growth. We address this issue by representing SCP collections as term graphs, which allows for the identification and elimination of trivial redundancies. By formalizing conditions for equivalent orders, we can eliminate non-trivial redundancies and reduce the graph size by several orders of magnitude. This enables the first computation of the perfect SCP heuristic and the first comparison of existing SCP approaches with their theoretical limit.

Acknowledgments

Travelling to Sweden to pursue my doctoral studies was an incredible journey and I want to thank everyone who helped make it possible.

First, I would like to thank my main supervisor, Jendrik Seipp. During my doctoral studies, I went from being his only doctoral student to being one of many. Yet, his support, advice, and patience never wavered despite the MR Lab's rapid growth. He provided the feedback and guidance I needed to grow, while giving me the space to pursue my own ideas. Most importantly, he allowed me to make my own mistakes and learn from them. His exceptional writing skills motivated me to produce precise yet understandable scientific publications.

Next, I would like to thank my second supervisor, David Speck, whose expertise in different planning approaches was invaluable. Thank you for always being available to answer my many questions.

I would also like to thank Florian Pommerening and Daniel Gnad for their excellent help and collaborations throughout my doctoral studies.

Special thanks go foremost to Anne Moe and Karin Baardsen, as well as the other administrative staff at IDA, for their support. Their expertise saved me countless hours.

I would also like to thank all my wonderful colleagues in the MR Lab for being more than just coworkers. I greatly appreciated all the insightful discussions, proofreading assistance, and general support. At the same time, conducting good research requires getting your mind off work regularly. Thank you to all my climbing buddies, long-distance skiing relay partners, hiking friends, and movie night companions.

A big thanks to Chorus Lin for many rehearsals and concerts that provided a meaningful counterpoint to my research. I found much-needed distraction and balance in singing with them.

I am grateful for my friendships with Windy, Arnaud, and Markus. You supported me during the hardest time of my life. This thesis would not have been possible without you.

I want to thank my family for making me feel at home every time I found the time to return to Germany. Your love and distraction were always greatly appreciated.

My greatest thanks go to my fiancée, Hellena Herrmann. You supported my decision to pursue this PhD even though we had to live more than 1300 km apart. I will forever be grateful that I was able to both pursue this degree and have you in my life. Your love and support are my greatest treasures.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

Paul Höft, Linköping, February 2026

Contents

1 Introduction

Building a rationally acting agent is widely recognized as the overall goal of artificial intelligence research (Russell and Norvig 2003). Automated planning, the task of finding the actions that will achieve a goal, is a quintessential component of building such an agent. It enables an agent to reason about alternatives, constraints, and long-term consequences instead of reacting to its environment. Without planning, AI systems would struggle to operate autonomously in complex domains such as robotics, logistics, and decision support, where foresight, coordination, and long-term reasoning are necessary.

This thesis focuses on classical planning, which restricts itself to the study of planning problems that are fully observable with deterministic effects. Despite being very restrictive, classical planning still models many interesting and challenging problems, ranging from transporting goods to elevator control and games like FreeCell and Sokoban. A planning system can approach all these problems by planning on a domain-independent representation such as boolean logic or transition systems. In *optimal* classical planning, we are only interested in finding provably optimal solutions with respect to a single objective, usually plan cost. Current optimal planners typically approach domain-independent planning as heuristic search (Bonet and Geffner

2001; Taitler et al. 2024). In this approach, the planning problem is converted into a transition system that is then systematically solved using a search algorithm. This approach is preferred because the A* search algorithm with an admissible heuristic (Pearl 1984) (goal-distance estimator) produces provably optimal solutions while exploring as little of the state space as possible given the heuristic (Hart et al. 1968). Automatically deriving strong admissible heuristics from a domain-independent representation is therefore a core research field of optimal classical planning.

Heuristics based on abstractions and cost partitioning have shown great promise in the recent International Planning Competition (IPC) (Taitler et al. 2024). The linear programming-based cost partitioning heuristic named *optimal cost partitioning* (Katz and Domshlak 2010; Pommerening et al. 2015), computes the best possible cost partition. Although it is the most informative cost partitioning heuristic, it is too expensive to compute. Non-optimal cost partitioning approaches, such as saturated post-hoc optimization (Pommerening et al. 2013; Seipp et al. 2021), saturated cost partitioning (Seipp et al. 2020), and the cost-partitioning-related potential heuristics (Pommerening et al. 2015; Seipp et al. 2015; Pommerening 2017; Fišer et al. 2020) perform much better despite computing a less informative heuristic. In addition to using a non-optimal cost partitioning strategy, cost partitioning heuristics can easily be approximated, since an admissible cost partitioning computed for one state is always admissible for all other states as well. Therefore, cost partitioning heuristics are often approximated, e.g., Karpas et al. (2011), Seipp et al. (2015), Seipp, Keller, et al. (2017b), and Seipp (2021). This thesis instead investigates the non-approximative, meaning the perfect, versions of two state-of-the-art cost partitioning strategies: the saturated post-hoc optimization heuristic and the saturated cost partitioning heuristic. We demonstrate that the perfect versions of these two heuristics can be computed much more effectively than with a naive approach.

1.1 Outline

The thesis is divided into two parts, one for each heuristic. The first part shows that computing the saturated post-hoc optimization (SPhO) linear program (LP) for each encountered state typically results in re-

dundant work. Many SPhO cost partitions can be reused for different states without approximating the heuristic, because the cost partitions are frequently optimal for states other than the one for which they were optimized. We demonstrate that translating the concept of sensitivity analysis from linear programming to classical planning makes it possible to predict whether a cost partition will remain a perfect heuristic when reused for another state. We define four such predictors that increase in strength but also in computational effort. Each predictor is verified to be safe, both theoretically and practically, meaning that the resulting heuristic is equivalent to the original SPhO heuristic.

We further analyze the LP solutions for degeneracy and non-uniqueness, conditions that are known to impair the performance of sensitivity analysis. Our experiments show that most saturated post-hoc optimization LPs have non-unique or degenerate solutions. As a countermeasure, we explore reformulations of the saturated post-hoc optimization LP that reduce the amount of redundant information. Finally, we explore cost partition tiebreaking. We propose two strategies for computing more versatile cost partitions, i.e., cost partitions that are reusable for more states.

In the second part, we study another state-of-the-art non-optimal cost partitioning algorithm: saturated cost partitioning. Unlike SPhO, its perfect heuristic is not just slow to compute. It is impossible to compute for most problems within the usual resource limits.

Saturated cost partitioning greedily computes a cost partition following a given order over component heuristics. The resulting heuristic quality depends heavily on this order, and the optimal order may differ for each state. Therefore, the perfect saturated cost partitioning heuristic would need to maximize over all possible orders. This grows factorially with the number of component heuristics and quickly becomes infeasible. We show that optimizing the heuristic representation with computational graphs can automatically remove many redundancies from the heuristic computation.

Combined with a theoretical analysis of which orders can be excluded without changing the heuristic value, this allows us to compute the perfect saturated cost partitioning heuristic for much larger abstraction sets than previously possible. We compare our approach with an existing sampling-based saturated cost partitioning algorithm and show that the existing approach is very close to computing the perfect heuristic version. This in turn confirms the large gap in heuris-

tic strength between saturated cost partitioning and optimal cost partitioning.

1.2 Experimental Setup

All algorithms in this thesis were implemented in Scorpion (Seipp et al. 2020), a planner that is based on the Fast Downward planning system (Helmert 2006). The planner uses the CPLEX LP solver version 20.1 to solve the linear programs. We conducted our experiments using the Downward Lab toolkit (Seipp, Pommerening, et al. 2017) with a benchmark set of all 1907 tasks from the international planning competitions from 1998-2023 that contain no conditional effects. As usual in the IPC, we limit time to 30 minutes and memory to 8 GiB. Of the 1907 tasks, 1884 can be grounded, a necessary preprocessing step in Scorpion, without running out of memory, making this the effective benchmark size. The experiments were run on a cluster of Intel Xeon Gold 6130 processors. All of our experimental data, code, and results are published online[1].

1.3 Contributions

Part one of this thesis unifies and builds upon the following two publications:

- Paul Höft et al. (2023b). "Sensitivity Analysis for Saturated Post-hoc Optimization in Classical Planning." In: *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*. Ed. by Kobi Gal et al. IOS Press, pp. 1044–1051
- Paul Höft et al. (2024). "Versatile Cost Partitioning with Exact Sensitivity Analysis." In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, pp. 276–280.

The second part is based on the publication:

[1] Paul Höft (2026). *Experiment data for the PhD Thesis "Computing Perfect Cost Partitioning Heuristics for Classical Planning"*. `https://doi.org/10.5281/zenodo.18376875`.

- Paul Höft et al. (2025). "Representing Perfect Saturated Cost Partitioning Heuristics in Classical Planning." In: *Proceedings of the Twenty-Second International Conference on Principles of Knowledge Representation and Reasoning (KR 2025)*. Ed. by Magdalena Ortiz et al. IJCAI Organization, pp. 821–831.

The following planner abstracts from the IPC 2023 are related to this thesis:

- Paul Höft et al. (2023a). "Dofri." In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*
- Dominik Drexler et al. (2023). "Ragnarok." In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

The following publications are not covered by this thesis:

- David Speck et al. (2023). "Finding Matrix Multiplication Algorithms with Classical Planning." In: *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. Ed. by Sven Koenig et al. AAAI Press, pp. 411–416
- Elliot Gestrin et al. (2025). "Explainable Planning via Counterfactual Task Analysis for the Beluga Challenge and Beyond." In: *ICAPS 2025 Workshop on Human-Aware and Explainable Planning (HAXP)*.

2 Background

In this chapter, we define the core concepts that will be referenced throughout the thesis. We introduce linear programming with its relevant definitions of bases, sensitivity analysis, and degeneracy. Afterward, we describe the most important classical planning background. This includes its general definition, heuristics, and cost partitioning.

2.1 Linear Programming

The first part of this work focuses on the saturated post-hoc optimization heuristic, which is based on linear programming. It will also make use of the sensitivity analysis of linear program solutions. Given that linear programming is not typically assumed as background knowledge in the planning literature, this section presents a more detailed introduction to the relevant LP concepts. Most notation and definitions in this section follow Vanderbei (2013) or Sierksma and Zwols (2015).

Linear programming (Dantzig 2002) is a subfield of Operations Research investigating systems of linear inequalities called *linear programs*.

Definition 2.1 (Canonical Linear Program)
A linear program (LP) in *canonical* form is defined as:

$$\begin{aligned} &\max_{x \in \mathbb{R}^n} c^T x \\ &\text{subject to } Ax \leq b \\ &\qquad x \geq 0 \\ &\text{with } A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n \end{aligned} \tag{2.1}$$

In a canonical LP, we look for the *decision variables* x that maximize the *objective* $z = c^\top x$ subject to linear constraints $Ax \leq b$ formed by the *constraint matrix* A and right-hand side bounds (RHS) b, along with nonnegativity constraints $x \geq 0$. Every primal LP has a corresponding dual LP.

Definition 2.2 (Dual Linear Program)
The dual linear program of a canonical maximization linear program is constructed as:

$$\begin{aligned} &\min_{y \in \mathbb{R}^m} b^\top y \\ &\text{subject to } A^\top y \geq c \\ &\qquad y \geq 0 \end{aligned}$$

with objective $w = b^\top y$.

The primal and dual LP have the same optimal objective value if such a solution exists. This is known as the *strong duality theorem* (Sierksma and Zwols 2015). In the following, we assume that all LPs or their duals are in canonical form.

Example 2.1
We introduce a running example with the following canonical linear program:

$$\begin{aligned} &\max_{x} 2x_1 + 1x_2 \\ &\text{subject to } x_1 + x_2 \leq 1 \\ &\qquad x_1 \leq 2 \\ &\qquad x_1, x_2 \geq 0 \end{aligned}$$

In matrix notation: $A = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, $b = \left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right]$ $c = \left[\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right]$, so the corresponding dual is:

$$\begin{aligned} \min_{y} \; & 1y_1 + 2y_2 \\ \text{subject to } & y_1 + y_2 \geq 2 \\ & y_1 \geq 1 \\ & y_1, y_2 \geq 0 \end{aligned}$$

Linear programming is a versatile tool due to its broad applicability and polynomial complexity (Spielman and Teng 2004; Kelner and Spielman 2006). The most commonly used algorithm for solving LPs is the *simplex* algorithm (Bradley et al. 1977). To apply the simplex algorithm to the input LP, it is converted into *augmented form* by adding m *slack* variables to the decision variables so that all constraints become equalities.

Example 2.2
Our running example expanded with slack variables has the following form:

$$\begin{aligned} \max_{x} \; & 2x_1 + 1x_2 \\ \text{subject to } & x_1 + x_2 + s_1 = 1 \\ & x_1 + s_2 = 2 \\ & x_1, x_2, s_1, s_2 \geq 0 \end{aligned}$$

So in augmented form the matrices become $A = \left[\begin{smallmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{smallmatrix}\right]$, $b = \left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right]$, and $c = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}$.

Since all results below expect a linear program in augmented form, we combine both the original decision variables and the slacks as the variable x, where $x_1, \ldots, x_n$ are the decision variables and $x_{n+1}, \ldots, x_{n+m}$ are the slacks. Every assignment to the decision variables x that satisfies the constraints is called *feasible*. The set of all feasible solutions comprises the *feasible region*, a *convex polytope* in n dimensions.

A *basic solution* of a linear program is a solution that divides the variables into two categories: m *basic* variables and n *non-basic* variables. The basic variables then create a linear basis of the current solution and can take non-zero values, while the non-basic variables are

set to zero. Setting all non-basic variables to zero leads to a solvable system of linear equations for valid bases that returns the values for the basic variables. Every solution that can be obtained by forming a basis is called a basic solution of the linear program. If a basic solution also obeys the non-negativity constraints it is a *basic feasible* solution.

All basic feasible solutions correspond to vertices of the feasible polytope and the simplex algorithm operates by traversing these basic feasible solutions of a linear program. This is no restriction, since if a feasible solution exists, there exists a basic feasible solution and if an optimal solution exists, there exists an optimal basic feasible solution. This is known as the *Fundamental Theorem of Linear Programming* (Vanderbei 2013).

In each iteration, the simplex algorithm swaps one basic variable, the *leaving* variable, with one non-basic variable, the *entering* variable, gradually increasing the objective value of the solution. For every solvable augmented canonical LP, there exists a basic feasible solution, where all slacks are basic, and all decision variables are non-basic (Sierksma and Zwols 2015).

Example 2.3
In our running example, fixing the decision variables x_1 and x_2 to zero gives us two linear equations with two unknowns, s_1 and s_2. The LP can then be solved with $s_1 = 1$ and $s_2 = 2$. Thus, x_1 and x_2 are the non-basic variables and s_1 and s_2 are the basic variables. Note that this is a non-optimal solution.

Because the distinction between basic and non-basic variables is fundamental to LP solutions obtained from the simplex algorithm, it is useful to adopt a notation that allows for the easy selection of these subsets of variables and their corresponding constraints. Following Vanderbei (2013), we introduce the index vectors $\mathcal{B}$ and $\mathcal{N}$ to denote basic and non-basic variable indices, respectively. These indices are used to slice matrices and vectors accordingly. For a matrix $M \in \mathbb{R}^{m \times n}$, $M_{\mathcal{B}} = (M_{ij})_{\substack{j \in \mathcal{B} \\ 1 \leq i \leq m}}$ selects only the columns corresponding to basic variables. For single columns, we use the shorthand $M_j = (M_{ij})_{1 \leq i \leq m}$. For vectors, the notation $v_{\mathcal{B}} = (v_i)_{i \in \mathcal{B}}$ refers to selecting the elements at indices in $\mathcal{B}$ [1].

[1] For matrix and vector notation, $^{-1}$ denotes matrix inversion, $^{\top}$ denotes matrix transposition and $^{-\top}$ their combination.

A basis is valid if the $m \times m$ *basis matrix* $B = A_{\mathcal{B}}$ formed by the m columns of A associated with the basic variables is invertible. In this case, fixing the non-basic decision variables to zero ($x_{\mathcal{N}} = 0$) and the basic decision variables to $x_{\mathcal{B}} = B^{-1}b$ results in the objective value $c_{\mathcal{B}}^{\top} x_{\mathcal{B}}$. Since any basic solution is fully defined by knowing the basic indices vector $\mathcal{B}$, we will hereafter refer to the basic indices vector as the basis of a linear program and use it to define LP solutions.

Example 2.4
In the initial basic feasible solution of our running example the slacks are the basic variables. The combined vector x has the entries $\begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \end{bmatrix}$, so $\mathcal{B} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ and $\mathcal{N} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. Here, $A_{\mathcal{B}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $A_{\mathcal{N}} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ $c_{\mathcal{B}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $c_{\mathcal{N}} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$. The solution is therefore $x_{\mathcal{B}} = B^{-1}b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ with objective value $c_{\mathcal{B}}^{\top} x_{\mathcal{B}} = \begin{bmatrix} 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} = 0$.

We now define the conditions that a typical LP solver will use to determine if an LP basis is optimal.

Definition 2.3 (Feasibility and Optimality Condition (Sierksma and Zwols 2015))
A basis $\mathcal{B}$ is feasible for a linear program with RHS bounds b and objective coefficients c if

$$B^{-1}b \geq 0 \tag{2.2}$$

and optimal if it is feasible and

$$c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^{\top} c_{\mathcal{B}} \leq 0 \tag{2.3}$$

It is important to note that the optimality condition is not a necessary condition. If the LP basis is degenerate (see Section 2.3) there can exist optimal basic solutions that do not fulfill the optimality condition.

2.2 Sensitivity Analysis of Linear Programs

Sensitivity analysis, also known as *post-optimality analysis*, determines how sensitive the optimal solution of an LP is to small changes in its parameters (Gal 1986). Since each alternative solution is reached by swapping one of the basic variables with one of the non-basic ones, it is possible to discern information about the stability of the current solution. This is done by inferring what changes would need to happen so

that a non-basic variable becoming basic would improve the solution again. For each non-basic variable, the simplex algorithm provides additional sensitivity information that can be used to determine the effect of small changes in that variable on the current basis, in the form of the *reduced cost* $\bar{c}_{\mathcal{N}}$ and *shadow prices* $\bar{y}$.

Definition 2.4 (Reduced Costs)
The *reduced costs* $\bar{c}_{\mathcal{N}}$ of the non-basic variables $x_{\mathcal{N}}$ can be computed as $\bar{c}_{\mathcal{N}} = c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^\top c_{\mathcal{B}}$.

The value $\bar{c}_i$ represents the smallest decrease in c_i that would allow x_i to enter the basis without decreasing the objective value. Equivalently, $\bar{c}_i$ equals the drop in the optimal objective value if we force $x_i = 1$ (when feasible). It can therefore be understood as a local one-sided derivative of the objective value for variable x_i. The reduced cost of a basic variable is always zero.

Since the objective coefficients of non-basic slack variables are always zero and their columns of the constraint matrix are unit vectors, the reduced costs for them simplify to

$$\begin{aligned}\bar{c}_{\mathcal{N}_s} &= c_{\mathcal{N}_s} - (B^{-1}A_{\mathcal{N}_s})^\top c_{\mathcal{B}} \\ &= 0 - I^\top B^{-\top} c_{\mathcal{B}} \\ &= -B^{-\top} c_{\mathcal{B}}.\end{aligned}$$

This quantity is the negative of the solution to the dual problem $y = B^{-\top} c_{\mathcal{B}}$ and relates changes of the constraints to the objective value.

Definition 2.5 (Shadow Price)
The *shadow price* $\bar{y}_i = (-B^{-\top} c_{\mathcal{B}})_i$ of constraint i is the change in objective value that occurs when changing the RHS of constraint i (i.e., b_i) by ± 1, provided this is feasible.

Computing the reduced cost of a slack variable therefore gives the shadow price. They have different names since they differ in their meaning: reduced costs are one-sided while shadow prices can be valid in both directions. Since the reduced costs represent the negative change in the objective value, having $\bar{c}_{\mathcal{N}} \leq 0$ guarantees that the current solution is optimal when using the simplex algorithm (see Equation 2.3).

Using the reduced costs and shadow prices, sensitivity analysis extracts intervals $[L, U]$ with $L, U \in \mathbb{R} \cup \{-\infty, \infty\}$ for the objective coef-

ficients c and constraint values b from an optimal LP solution. The interval bounds define limits on the perturbation of a single c_i or b_i such that if the perturbed $b'_i = b_i + \Delta b_i$ is inside the bounds $L \leq b'_i \leq U$, the previous basis $\mathcal{B}$ is still an optimal solution for the perturbed LP. We can use the conditions from Equations 2.2 and 2.3 to check if changes of the objective coefficients or the RHS bounds preserve the optimality of the current basis.

Example 2.5
The optimal solution basis for our running example is $\mathcal{B} = [\begin{smallmatrix}1\\4\end{smallmatrix}]$ and therefore $\mathcal{N} = [\begin{smallmatrix}2\\3\end{smallmatrix}]$. This can be confirmed by checking that Equations 2.2 and 2.3 hold.

$$B^{-1}b = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \geq 0$$

$$\begin{aligned} \bar{c}_{\mathcal{N}} = c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^{\top} c_{\mathcal{B}} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \left(\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \right)^{\top} \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} -1 \\ -2 \end{bmatrix} \leq 0 \end{aligned}$$

Since $\mathcal{N}_2$ is a slack variable, the second entry $\bar{c}_{\mathcal{N}_2} = -2$ is really a shadow price $\bar{y}_2 = 2$.

Following the definitions from Bradley et al. (1977), Vanderbei (2013), and Sierksma and Zwols (2015), we will now show the most important sensitivity analysis results for this work. This includes the range-based method for computing sensitivity bounds for changes of a single value and the 100% rule, which also applies when multiple values are changed at once. We show these results for computing the sensitivity bounds under perturbations of the objective coefficients and RHS bounds.

2.2.1 Ranges for Objective Coefficients

When adjusting an objective coefficient $c'_i = c_i + \Delta c_i$, only the optimality condition is changed and feasibility is preserved. The change in reduced costs is $\Delta \bar{c}_{\mathcal{N}} = \Delta c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^{\top} \Delta c_{\mathcal{B}}$ and optimality holds as long as the changed reduced costs still fulfill the optimality condition

from Equation 2.3.

$$\begin{aligned} & \bar{c}_{\mathcal{N}} + \Delta\bar{c}_{\mathcal{N}} \leq 0 \\ \text{iff} \quad & \Delta\bar{c}_{\mathcal{N}} \leq -\bar{c}_{\mathcal{N}} = -(c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^\top c_{\mathcal{B}}) \\ \text{iff} \quad & \Delta c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^\top \Delta c_{\mathcal{B}} \leq (B^{-1}A_{\mathcal{N}})^\top c_{\mathcal{B}} - c_{\mathcal{N}} \end{aligned} \tag{2.4}$$

If the variable x_i corresponding to c_i is non-basic, then $\Delta c_{\mathcal{B}} = 0$ and Equation 2.4 simplifies to $\Delta c_{\mathcal{N}} \leq (B^{-1}A_{\mathcal{N}})^\top c_{\mathcal{B}} - c_{\mathcal{N}}$. Since c_i is the only changed value in $c_{\mathcal{N}}$, we only need to ensure that $\Delta c_i \leq (B^{-1}A_i)^\top c_{\mathcal{B}} - c_i$. This condition can be expressed as bounds on the values of c_i'.

Proposition 2.1 (Sierksma and Zwols 2015) *An LP basis B remains optimal under changes of an objective coefficient c_i of a non-basic variable x_i as long as $L \leq c_i' \leq U$, where*

$$L = -\infty, \quad U = (B^{-1}A_i)^\top c_{\mathcal{B}} = c_i - \bar{c}_i$$

If x_i is basic instead, $\Delta c_{\mathcal{N}} = 0$ and Equation 2.4 simplifies as follows:

$$\begin{aligned} \Delta c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^\top \Delta c_{\mathcal{B}} &= -(B^{-1}A_{\mathcal{N}})^\top \Delta c_{\mathcal{B}} \\ &\leq (B^{-1}A_{\mathcal{N}})^\top c_{\mathcal{B}} - c_{\mathcal{N}} = -\bar{c}_{\mathcal{N}} \\ \text{iff} \quad (B^{-1}A_{\mathcal{N}})^\top \Delta c_{\mathcal{B}} &\geq \bar{c}_{\mathcal{N}} \end{aligned}$$

With $d = (B^{-1}A_{\mathcal{N}})_i^\top$, we have $d\Delta c_i \geq \bar{c}_{\mathcal{N}}$. Each row d_j of d poses its own restrictions onto Δc_i:

$$\Delta c_i \geq \frac{\bar{c}_{\mathcal{N}_j}}{d_j} \quad \text{if } d_j > 0$$

$$\Delta c_i \leq \frac{\bar{c}_{\mathcal{N}_j}}{d_j} \quad \text{if } d_j < 0$$

If $d_j = 0$, the row poses no restrictions on Δc_i.

Proposition 2.2 *An LP basis remains optimal under changes of an objective coefficient c_i of a basic variable x_i as long as $L \leq c_i' \leq U$, where*

$$L = c_i + \max_{d_j > 0} \left(\frac{\bar{c}_{\mathcal{N}_j}}{d_j} \right), \quad U = c_i + \min_{d_j < 0} \left(\frac{\bar{c}_{\mathcal{N}_j}}{d_j} \right), \quad d = (B^{-1}A_{\mathcal{N}})_i^\top,$$

$\max(\emptyset) = -\infty$ *and* $\min(\emptyset) = \infty$.

When adjusting c_i inside the given bounds, the basis stays the same, but the objective value of the linear program changes. The new objective value can be computed as $(c_{\mathcal{B}} + \Delta c_{\mathcal{B}})^\top x_{\mathcal{B}}$, since the solution stays the same.

Example 2.6
In our running example, the reduced costs of c_2 are -1, so following Proposition 2.1, the objective can be adjusted up to $2x_1 + 2x_2$ without losing optimality of the basis. For c_1, which belongs to a basic variable, we follow Proposition 2.2 to ensure $d\Delta c_1 \geq \bar{c}_{\mathcal{N}}$.

$$d = (B^{-1}A_{\mathcal{N}})_i^\top = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \bar{c}_{\mathcal{N}} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

$$L = c_i + \max_{d_j > 0} \left(\frac{\bar{c}_{\mathcal{N}_j}}{d_j} \right) = 2 + \max\left(\frac{-1}{1}, \frac{-2}{1} \right) = 2 + -1 = 1$$

$$U = c_i + \min_{d_j < 0} \left(\frac{\bar{c}_{\mathcal{N}_j}}{d_j} \right) = 2 + \min(\emptyset) = 2 + \infty = \infty$$

$$1 \leq c_1 \leq \infty$$

This tells us that the current basis is optimal for changes of c_1 down to $1x_1 + 1x_2$ and any changes upwards.

2.2.2 Ranges of Right-Hand Side Constraints

Adjusting the RHS changes the basic variables $x_{\mathcal{B}}$. Since everything else stays the same, the current basis remains feasible as long as $x_{\mathcal{B}} = B^{-1}b \geq 0$. Feasibility also implies optimality here since the reduced costs do not change. We perform the same perturbation analysis on this condition to obtain the sensitivity ranges.

$$x'_{\mathcal{B}} = x_{\mathcal{B}} + \Delta x_{\mathcal{B}} \geq 0 \text{ iff } B^{-1}\Delta b \geq -x_{\mathcal{B}}$$

The bounds follow from each row j individually:

$$\Delta b_i \geq \frac{-x_{\mathcal{B}_j}}{B^{-1}_{ji}} \quad \text{if } B^{-1}_{ji} > 0$$

$$\Delta b_i \leq \frac{-x_{\mathcal{B}_j}}{B^{-1}_{ji}} \quad \text{if } B^{-1}_{ji} < 0$$

Proposition 2.3 *(Bradley et al. 1977) An LP basis remains optimal when changing one RHS bound from b_i to b'_i, as long as $L \leq b'_i \leq U$,*

where

$$L = b_i + \max_{j, B^{-1}_{ji} > 0} \left(\frac{-x_{\mathcal{B}_j}}{B^{-1}_{ji}} \right), \quad U = b_i + \min_{j, B^{-1}_{ji} < 0} \left(\frac{-x_{\mathcal{B}_j}}{B^{-1}_{ji}} \right)$$

and $\max(\emptyset) = -\infty$, $\min(\emptyset) = \infty$.

The new *optimal* objective value z' can be computed from the old objective value z and shadow prices of the RHS constraints: $z' = z + \bar{y}_i(b'_i - b_i)$.

Example 2.7
As an example, we calculate the sensitivity bounds of b_1 in our running example:

$$B^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad x_{\mathcal{B}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$L = b_i + \max_{j, B^{-1}_{ji} > 0} \left(\frac{-x_{\mathcal{B}_j}}{B^{-1}_{ji}} \right) = 1 + \max \left(\frac{-1}{1} \right) = 0$$

$$U = b_i + \min_{j, B^{-1}_{ji} < 0} \left(\frac{-x_{\mathcal{B}_j}}{B^{-1}_{ji}} \right) = 1 + \min \left(\frac{-1}{-1} \right) = 2$$

Increasing the bound b_1 from 1 to 2 would therefore improve the optimal objective from $z = 2$ to

$$\begin{aligned} z' &= z + \bar{y}_i(b'_i - b_i) \\ &= 2 - \left(B^{-\top} c_{\mathcal{B}}\right)_1 \cdot 1 \\ &= 2 - \left(\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right)_1 \\ &= 4. \end{aligned}$$

2.2.3 100% Rule for Objective Values and Right-Hand Side Constraints

The sensitivity intervals $[L, U]$ are limits derived for perturbations of single coefficients or bounds. This means that, without further consideration, they only hold as long as only one such parameter is changed at a time. As soon as multiple parameters are changed simultaneously, these bounds are no longer valid. Addressing this issue, the 100%

rule (Bradley et al. 1977) provides a set of safe conditions for generalizing the interval ranges to cases where multiple parameters are changed together.

Proposition 2.4 *(Bradley et al. 1977) An LP basis remains optimal while changing multiple RHS bounds from b_i to $b'_i = b_i + \Delta b_i$ as long as*

$$\sum_{1 \leq i \leq m} \lambda_i \leq 1, \textit{ where } \lambda_i = \frac{\Delta b_i}{\Delta b_i^*} \textit{ with } \Delta b_i^* = \begin{cases} U_i - b_i & \textit{if } \Delta b_i \geq 0, \\ L_i - b_i & \textit{otherwise.} \end{cases}$$

In this context, division by $\pm\infty$ is defined as zero.

Example 2.8
In our running example, the bounds for b'_1 and b'_2 are $0 \leq b'_1 \leq 2$ and $1 \leq b'_2 \leq \infty$. Using the 100% rule reveals that, when increasing b_2, b_1 can still be freely changed inside its bounds without affecting optimality since the upper bound for b_2 is ∞ and therefore does not increase λ.

For the objective coefficients, P. Cai and J.-Y. Cai (1997) showed that the same approach can be strengthened by distinguishing between coefficients for basic and non-basic variables.

Proposition 2.5 *(P. Cai and J.-Y. Cai 1997) An LP basis remains optimal while changing multiple objective coefficients as long as the following inequality is fulfilled.*

$$\sum_{k \in \mathcal{B}} \lambda_k + \max_{l \in \mathcal{N}} \lambda_l \leq 1, \textit{ with } \lambda_i = \frac{\Delta c_i}{\Delta c_i^*}$$

$$\textit{and } \Delta c_i^* = \begin{cases} U_i - c_i & \textit{if } \Delta c_i \geq 0, \\ L_i - c_i & \textit{otherwise.} \end{cases}$$

There are methods to compute stronger approximations in the case of multiple parameter changes, such as *parametric analysis* (Bradley et al. 1977), *two-sided* shadow prices (Gal 1986), or the tolerance approach (Wendell 1985). Since these methods are quite sophisticated and computationally demanding, we focus on the efficient sensitivity analysis methods presented above.

2.3 Degeneracy and Non-Uniqueness of LP Solutions

The solution of a linear program is usually defined as the objective value z or the assignment to the decision variables x. In contrast, the simplex algorithm works with bases. In pathological cases known as *degeneracy* and *non-uniqueness*, the correspondence between bases and decision variables or solution values is no longer one-to-one.

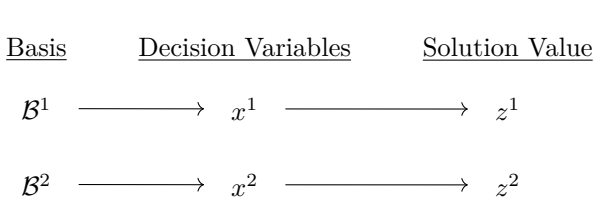
Definition 2.6 (Degenerate Bases (Sierksma and Zwols 2015))
An LP basis is *degenerate* when at least one of its basic decision variables $x_{\mathcal{B}}$ is zero. The zero-valued basic variables are then also called degenerate variables.

A degenerate basis indicates that alternative bases with the same decision variable assignment can exist. Geometrically, a degenerate basis is a vertex of the n-dimensional solution polytope where more than n facets intersect. This leads to a basic variable that is effectively non-basic and is constrained to zero. Under some circumstances, the degenerate variable can be swapped with a non-basic one to form a different basis, where all non-zero basic variables retain their values.

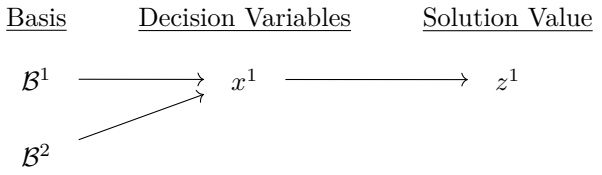
Example 2.9
When adjusting the bound of the first constraint in our running example, we end up with a degenerate optimal basis.

$$\begin{aligned} \max_{x} \; & 2x_1 + 1x_2 \\ \text{subject to } & x_1 + x_2 \leq 2 \\ & x_1 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

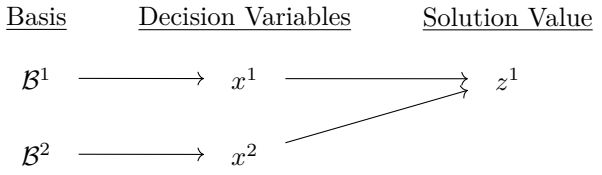
We previously found via sensitivity analysis that the basis $\mathcal{B} = \left[\begin{smallmatrix} 1 \\ 4 \end{smallmatrix}\right]$ is still optimal under this change. When computing the basic solution values, we get $x_{\mathcal{B}} = B^{-1}b = \left[\begin{smallmatrix} 1 & 0 \\ -1 & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 2 \\ 2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}\right]$. This solution is degenerate because the basic variable $s_2 = 0$. In this case there also exists an alternative basis that shares the same solution values $x_1 = 2, x_2 = 0, s_1 = 0, s_2 = 0$. It is the basis $\mathcal{B}' = \left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right]$. This can be confirmed by computing $x_{\mathcal{B}'} = \left[\begin{smallmatrix} 1 & 0 \\ -1 & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 2 \\ 2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}\right]$.



(a) Non-degenerate unique bases



(b) Degenerate bases



(c) Non-unique bases

Figure 2.1: Relation of bases to LP solutions under degeneracy and non-uniqueness. $\mathcal{B}^2$ and x^2 are placeholders for potentially many alternative bases and decision variables under degeneracy and non-uniqueness.

Non-uniqueness (also called multiplicity) occurs when multiple assignments to the decision variables $x_{\mathcal{B}}$ are optimal.

Definition 2.7 (Non-unique Bases (Sierksma and Zwols 2015))
An LP basis is *non-unique* when at least one of the reduced costs $\bar{c}_{\mathcal{N}}$ is zero.

A non-unique basis indicates that alternative solutions with different solution values can exist. This happens when the objective coefficients and at least one constraint are linearly dependent. Degeneracy and non-uniqueness are not definitive indicators of alternative solutions: they only show that alternatives *may* exist, because both conditions can occur simultaneously and then influence each other. Figure 2.1 shows the effect of degeneracy and non-uniqueness on the rela-

tion between bases, decision variables and solution values. Additionally, degeneracy in the primal LP is closely related to non-uniqueness in the dual and vice versa. We refer to Sierksma and Zwols (2015) for an explanation of the subtleties involved.

Example 2.10
Adjusting the objective coefficients in our running example from $c = \left[\begin{smallmatrix}2\\1\end{smallmatrix}\right]$ to $\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$ produces a non-unique optimal solution. We showed previously that the basis $\mathcal{B} = \left[\begin{smallmatrix}1\\4\end{smallmatrix}\right]$ is still optimal under these changes, so we first show that this is indeed a non-unique basis by calculating the reduced costs: $\bar{c}_{\mathcal{N}} = c_{\mathcal{N}} - (B^{-1}A_{\mathcal{N}})^{\top} c_{\mathcal{B}} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] - \left(\left[\begin{smallmatrix}1 & 0\\-1 & 1\end{smallmatrix}\right]\left[\begin{smallmatrix}1 & 1\\0 & 0\end{smallmatrix}\right]\right)^{\top} \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] = \left[\begin{smallmatrix}0\\-1\end{smallmatrix}\right]$. Since the reduced costs for x_2 are zero, this is a non-unique solution. The alternative solution is the basis $\mathcal{B}' = \left[\begin{smallmatrix}2\\4\end{smallmatrix}\right]$. This can be confirmed by computing $z = c_{\mathcal{B}}^{\top} B^{-1} b = \left[\begin{smallmatrix}1 & 0\end{smallmatrix}\right]\left[\begin{smallmatrix}0 & \frac{1}{2}\\1 & 0\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\2\end{smallmatrix}\right] = 2$. This differs from degeneracy because here the solution values do change: $x_{\mathcal{B}} = B^{-1}b = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$ and $x_{\mathcal{B}'} = \left[\begin{smallmatrix}1\\2\end{smallmatrix}\right]$.

Both conditions negatively affect sensitivity analysis because they break the expected guarantee that a different basis leads to a different solution. Because degeneracy or non-uniqueness allow multiple optimal bases, the change in basis that sensitivity analysis predicts as marking the loss of optimality may instead lead to a different basis that is still optimal. As a result, the computed sensitivity ranges no longer correctly indicate when the current solution ceases to be optimal. Figure 2.1 shows that, under degeneracy, sensitivity analysis for $\mathcal{B}^1$ might give different bounds than for $\mathcal{B}^2$ even though they define the same decision variable assignment.

Example 2.11
When calculating the sensitivity of c_1 in the adjusted running LP from Example 2.9 we get the same sensitivity analysis as before when looking at the basis $\mathcal{B} = \left[\begin{smallmatrix}1\\4\end{smallmatrix}\right]$, since none of the involved values changed. When looking at the alternative optimal basis $\mathcal{B}' = \left[\begin{smallmatrix}1\\2\end{smallmatrix}\right]$ instead, the effect of degeneracy becomes apparent. Under this other basis, the calculated bounds for c_1 are: $\bar{c}_{\mathcal{N}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] - \left[\begin{smallmatrix}0 & 1\\1 & -1\end{smallmatrix}\right]\left[\begin{smallmatrix}2\\2\end{smallmatrix}\right] = \left[\begin{smallmatrix}-2\\0\end{smallmatrix}\right]$, $\left[\begin{smallmatrix}0 & 1\\1 & -1\end{smallmatrix}\right]_1 \Delta c_1 \geq \left[\begin{smallmatrix}-2\\0\end{smallmatrix}\right]$, so $2 \leq c_1 \leq \infty$, which is more restrictive than the bounds we got before.

Removing redundant constraints and variables from a linear program can help to reduce the amount of degenerate and non-unique solutions.

Example 2.12
In the adjusted running LP from Example 2.9 the second constraint is redundant. Removing it helps improve the sensitivity analysis, since the number of optimal feasible bases reduces to one $\mathcal{B} = [1]$, which still gives the original sensitivity analysis bounds from Example 2.9 for c_1.

2.4 Optimal Classical Planning

Classical planning is a subfield of AI planning that aims to find solutions to discrete, fully observable, deterministic, single-player problems (tasks) with instantaneous actions. Since the goal is to create problem-independent solvers, classical planners accept tasks defined in a planning formalism, i.e., a problem-independent input language. This thesis will focus on planning tasks in the Simplified Action Structures (SAS$^+$) formalism (Bäckström and Nebel 1995). A planning task given in SAS$^+$defines a finite-domain representation of the following form.

Definition 2.8 (Planning Task)
A planning task Π is a tuple $\langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \gamma \rangle$ with:

- $\mathcal{V} = \langle v_1, \ldots, v_n \rangle$: a tuple of finite-domain variables. Each variable v_i has a domain $\mathit{dom}(v_i)$, a set of values it can take. A partial state is a partial function mapping variables to values from their respective domains: $s \colon \mathit{vars}(s) \to \bigcup_{v \in \mathcal{V}} \mathit{dom}(v)$, where $\mathit{vars}(s) \subseteq \mathcal{V}$. If $\mathit{vars}(s) = \mathcal{V}$, s is called a state. $S(\Pi)$ is the set of all states of Π.
- $\mathcal{O}$: a set of operators o. An operator is a tuple $\langle \mathit{pre}(o), \mathit{eff}(o), \mathit{cost}(o) \rangle$. The preconditions $\mathit{pre}(o)$ and effects $\mathit{eff}(o)$ are partial states and $\mathit{cost}(o) \in \mathbb{R}_{\geq 0}$.
- $\mathcal{I}$: an initial state.
- γ: a set of (partial) states called goal states.

We say a (partial) state s *satisfies* another (partial) state t if $\forall v \in \mathit{vars}(t) \colon s(v) = t(v)$, or in short we write $t \subseteq s$. An operator o is applicable in state s if $\mathit{pre}(o) \subseteq s$. Applying o in state s results in a new state

$s[\![o]\!] = s \oplus \textit{eff}(o)$, where $\textit{vars}(s \oplus \textit{eff}(o)) = \textit{vars}(s) \cup \textit{vars}(\textit{eff}(o))$ and $(s \oplus \textit{eff}(o))[v] = \begin{cases} \textit{eff}(o)[v] & \text{for all } v \in \textit{vars}(\textit{eff}(o)). \\ s[v] & \text{for all } v \in \textit{vars}(s) \setminus \textit{vars}(\textit{eff}(o)). \end{cases}$

The idea of planning as state-space search or heuristic search (Bonet and Geffner 2001) is to translate this finite-domain representation into a transition system that can be traversed with a search algorithm. A planning problem expressed in the SAS$^+$formalism induces a weighted transition system.

Definition 2.9 (Transition System)
A transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ is a directed graph with labelled edges. S is a set of states, L a set of labels, and a set of labeled transitions $T \subseteq S \times L \times S$. We also write transitions $(s, \ell, s') \in T$ as $s \xrightarrow{\ell} s'$. It has one initial state $s_0 \in S$ and a set of goal states $S_* \subseteq S$. A label ℓ *affects* a transition system $\mathcal{T}$ if and only if there exists a transition $s \xrightarrow{\ell} s'$ in T such that $s \neq s'$.

Definition 2.10 (Cost Function)
A *cost function* for transition system $\mathcal{T}$ is a function $\textit{cost}\colon L \to \mathbb{R} \cup \{-\infty, \infty\}$. We write $C(\mathcal{T})$ for the set of all cost functions for $\mathcal{T}$. A cost function is *non-negative* if $\textit{cost}(\ell) \geq 0$ for all $\ell \in L$ of $\mathcal{T}$. A cost function that is not non-negative (i.e., for which there exists an $\ell \in L$ such that $\textit{cost}(\ell) < 0$) is called *general*.

Definition 2.11 (Weighted Transition System)
A weighted transition system is a tuple $\langle \mathcal{T}, \textit{cost} \rangle$ where $\mathcal{T}$ is a transition system and *cost* is a cost function for $\mathcal{T}$. We also write $\mathcal{T} = \langle S, L, T, s_0, S_*, \textit{cost} \rangle$ for weighted transition systems.

Definition 2.12 (State Space)
The induced weighted transition system, called a *state space*, of a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \gamma \rangle$ is the transition system $\mathcal{T}_\Pi = \langle S, L, T, s_0, S_*, \textit{cost} \rangle$ where

- $S = S(\Pi)$
- $L = \mathcal{O}$
- $T = \{s \xrightarrow{o} (s \oplus \textit{eff}(o)) | s \in S, o \in \mathcal{O} \text{ such that } \textit{pre}(o) \subseteq s\}$
- $s_0 = \mathcal{I}$

- $S_* = \{s \in S | \gamma \subseteq s\}$
- $\mathit{cost}(\ell) = \mathit{cost}(o)$, where label ℓ corresponds to operator o

A sequence of actions that leads from one state to one of the goal states in S_* is called a plan.

Definition 2.13 (Plan)
Let $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ be a transition system. A sequence of labels $\pi = \langle \ell_0, \ldots, \ell_{n-1} \rangle$ is called a plan for state $s_0 \in S$ if there exists a sequence of states $\langle s_0, \ldots, s_n \rangle$ such that $s_i \xrightarrow{\ell_i} s_{i+1}$ for all $0 \leq i \leq n-1$ and $s_n \in S_*$. Given a cost function *cost* for $\mathcal{T}$, the cost of a plan is $\mathit{cost}(\pi) = \sum_{i=0}^{n-1} \mathit{cost}(\ell_i)$, where infinities are handled by the rule of left-addition so $\infty - \infty = \infty$. A plan π is *optimal* for state s if there exists no other plan π' with $\mathit{cost}(\pi') < \mathit{cost}(\pi)$. A plan for the initial state of a transition system is a *solution* for it.

Definition 2.14 (Goal distance)
Let $\mathcal{T}$ be a transition system. Given a cost function *cost* for $\mathcal{T}$, the goal distance $h^*_{\mathcal{T}}(s, \mathit{cost})$ of a state $s \in S$ is the cheapest path to a goal state if such a path exists and ∞ otherwise. We will leave out the transition system if it is obvious from context.

The objective of *optimal classical planning* is to find provably optimal solutions to a given planning problem or prove that no such plan exists. One approach that can fulfill these conditions is A* search with an admissible heuristic (Hart et al. 1968).

2.4.1 Heuristics

A heuristic function guides a search algorithm by providing an estimate of how costly the optimal path to the goal is (Pearl 1981). The goal distance is also called the perfect heuristic since it is the perfect estimator.

Definition 2.15 (Heuristic)
Given a transition system $\mathcal{T}$, a *heuristic* for $\mathcal{T}$ is a function $h \colon S \times C(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$.

Heuristics can be classified based on whether they fulfill certain properties. Depending on the search algorithm, these properties can

translate into theoretical guarantees of the search and the found solution. The important ones for this work are:

Definition 2.16 (Heuristic Properties)
Let $\mathcal{T}$ be a transition system and h a heuristic for $\mathcal{T}$. h is called:

- *goal-aware* if $h(s, \mathit{cost}) \leq 0$ for all goal states $s \in S_*$,
- *admissible* if $h(s, \mathit{cost}) \leq h^*(s, \mathit{cost})$ for all states $s \in S$ and all cost functions $\mathit{cost} \in C(\mathcal{T})$,
- *consistent* if $h(s, \mathit{cost}) \leq \mathit{cost}(\ell) + h(s', \mathit{cost})$ for all transitions $s \xrightarrow{\ell} s' \in T$ and cost functions $\mathit{cost} \in C(\mathcal{T})$.

A consistent and goal-aware heuristic is admissible (Russell and Norvig 2003), also under general cost functions (Seipp et al. 2020). One of the strongest approaches to computing domain-independent admissible heuristics is to use abstraction heuristics (Edelkamp 2001; Helmert et al. 2007; Katz and Domshlak 2010; Seipp and Helmert 2018). Abstraction heuristics simplify a planning task by not distinguishing between certain states. This results in a coarser transition system that over-approximates the original.

Definition 2.17 (Abstraction)
Let $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ be a transition system. Given a surjective *abstraction function* $\alpha \colon S \to S^\alpha$, the abstract transition system is $\mathcal{T}^\alpha = \langle S^\alpha, L, T^\alpha, \alpha(s_0), S_*^\alpha \rangle$. The abstract transitions are $T^\alpha = \{\alpha(s) \xrightarrow{\ell} \alpha(s') | s \xrightarrow{\ell} s' \in T\}$ and the abstract goal states become $S_*^\alpha = \{\alpha(s) | s \in S_*\}$.

Everything that is possible in the original task is also possible in the abstract task. The perfect goal distance in an abstraction is therefore an admissible heuristic for the original task. This means that every abstraction defines an admissible heuristic, as long as it is small enough that its perfect goal distances can be computed.

Definition 2.18 (Abstraction Heuristic)
Let $\mathcal{T}^\alpha$ be an abstract transition system for abstraction function α. The *abstraction heuristic* is the *abstract goal distance* of $\mathcal{T}^\alpha$: $h^\alpha(s, \mathit{cost}) = h^*_{\mathcal{T}^\alpha}(\alpha(s), \mathit{cost})$.

In practice, the abstract goal distances are precomputed when creating the abstraction and stored in a *lookup table* that represents the heuristic function.

There exists a hierarchy of methods to compute abstraction heuristics. They range from projections (Helmert et al. 2007), which are the coarsest method, to domain abstractions (Kreft et al. 2023), Cartesian abstractions (Seipp and Helmert 2018), and merge-and-shrink abstractions (Helmert et al. 2007), which are the most flexible method. In this thesis, we will make use of projections and Cartesian goal abstractions.

Definition 2.19 (Projection)
Let $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ be a transition system induced by planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \gamma \rangle$, and let $V \subseteq \mathcal{V}$ be a subset of the variables of Π. A *projection* is the equivalence relation $\sim \in S \times S$, such that $s \sim s'$ iff for all$v \in V, s(v) = s'(v)$. Its abstraction function $\alpha : S \to S^\alpha$ follows from the projection of $\sim$: $\alpha(s) = [s]$ for all $s \in S$. Here $[s]$ is the equivalence class of s under $\sim$, so $[s] = \{s' \in S : s \sim_{\mathcal{H}} s'\}$. The number of equivalence classes, i.e., the number of abstract states, is known as the size of the projection. A projection is also called a pattern and a collection of patterns a pattern database (PDB).

We refer to Pommerening et al. (2013) and Pommerening et al. (2021) for the systematic calculation of all interesting projections up to a certain number of variables.

Definition 2.20 (Cartesian Set)
Let $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ be a transition system. A *Cartesian set* for $\mathcal{T}$ is a set $C = A_1 \times \cdots \times A_n$ where $A_i \subseteq \mathit{dom}(v_i)$ for all $v_i \in \mathcal{V}$. A state $s \in C$ iff $s[v_i] \in A_i$ for all $1 \leq i \leq n$ and $s \in S$.

Definition 2.21 (Cartesian Abstraction)
Let $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ be a transition system. A *Cartesian abstraction* for $\mathcal{T}$ is a *partition* $P = \{C_1, \ldots, C_k\}$ of S such that every C_j is Cartesian. It induces an abstraction function $\alpha(s) = C_j$ iff $s \in C_j$.

We refer to Seipp and Helmert (2018) for the computation of Cartesian goal abstractions.

2.4.2 Cost Partitioning

The perfect abstraction heuristic would be both informative and fast to compute. However, these are conflicting goals. A fast and memory-efficient abstraction heuristic can only accurately represent a small subset of the state space dynamics if the planning problem is complex. One way to overcome this dichotomy is to compute multiple abstraction heuristics that complement each other. Maximizing over multiple different abstraction heuristics provides a more informative heuristic than computing one large abstraction with the same resources (Holte et al. 2006), so much research has been conducted on how to combine heuristic ensembles informatively while guaranteeing admissibility.

Maximizing over the component heuristics is bounded by the strength of the individual heuristics. In contrast, *additive* collections, where it is possible to admissibly sum the heuristics, can combine the strengths of multiple heuristics. They were first introduced in the form of disjunctive pattern databases (Korf and Felner 2002; Edelkamp 2001; Haslum et al. 2007) and were then generalized to arbitrary heuristics in the form of *cost relaxation* (Haslum et al. 2005). The most general method for creating additive heuristic collections is *cost partitioning* (Katz and Domshlak 2008; Katz and Domshlak 2010; Yang et al. 2008; Pommerening et al. 2015). With cost partitioning, any collection of heuristics can be transformed into an additive collection by adjusting the cost functions for the individual heuristics.

Definition 2.22 (Cost Partitioning)
Let $\mathcal{T}$ be a transition system and *cost* a cost function for it. Given a heuristic collection $\mathcal{H} = \langle h_1, \dots, h_n \rangle$, a *cost partition* is a function $\mathcal{C} \colon \mathcal{H} \to C(\mathcal{T})$ such that the following condition, called the *cost partitioning condition*, is fulfilled.

$$\sum_{i=1}^{n} \mathcal{C}(h_i)(\ell) \leq \mathit{cost}(\ell) \quad \text{for all } \ell \in L \tag{2.5}$$

The cost-partitioned heuristic is $h^{\mathcal{C}}(s) = \sum_{i=1}^{n} h_i(s, \mathit{cost}_i)$ where the sum uses the rules of left-addition, so $\infty - \infty = \infty$. If all component heuristics $h_i \in \mathcal{H}$ are admissible then $h^{\mathcal{C}}$ is admissible.

Optimal cost partitioning is a linear programming-based heuristic that computes the best possible cost partitioning.

Definition 2.23 (Optimal Cost Partitioning)
Let $\mathcal{T}$ be a transition system, *cost* a cost function for it, and $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ a heuristic ensemble over $\mathcal{T}$. Let $\mathcal{P}$ be the set of all possible cost partitions over $\mathcal{H}$. Then the optimal cost partitioning heuristic $h^{\text{OCP}}(s, \mathit{cost})$ computes the best possible cost partition $\max_{\mathcal{C} \in \mathcal{P}} h^{\mathcal{C}}(s, \mathit{cost})$.

We refer to Katz and Domshlak (2010) and Pommerening et al. (2015) for a definition of the linear program and the proof of optimality.

PART
I

Perfect Saturated Post-Hoc Optimization

3 Saturated Post-Hoc Optimization

The post-hoc optimization heuristic is an admissible heuristic for classical planning from the family of operator counting heuristics (Pommerening et al. 2014). Operator counting heuristics are defined as linear programs (LPs) that estimate the number of times each label will be used in the goal path from a state. This is achieved by minimizing an LP over the label-count variables subject to different kinds of constraints, depending on the particular operator counting heuristic. The post-hoc optimization heuristic uses constraints that ensure that the operator count satisfies the heuristic value of a given abstraction heuristic.

The *saturated* variant (SPhO) additionally uses the minimum saturated cost functions (defined below) of each heuristic in the constraints. In theory this can be relaxed to a general saturated cost function to allow heuristics for which no unique minimum saturated cost function exists. However, this work only considers abstraction heuristics as component heuristics for SPhO, so the minimum saturated cost function yields the highest heuristic values. This tightens each constraint, making the saturated version strictly stronger than the non-saturated version in theory and practice (Seipp et al. 2021). We next define minimum saturated cost functions and the SPhO LP.

Definition 3.1 (Minimum Saturated Cost Function)
The minimum saturated cost function *mscf* (Seipp et al. 2020) of an (abstract) transition system $\mathcal{T}$ with transitions T and cost function *cost* is defined as

$$\mathit{mscf}(\ell) = \sup_{a \xrightarrow{\ell} b \in T} (h^*_{\mathcal{T}}(a, \mathit{cost}) \ominus h^*_{\mathcal{T}}(b, \mathit{cost})),$$

where $x \ominus y$ is defined by

$$x \ominus y = \begin{cases} x - y, & \text{if } x, y \in \mathbb{R}, \\ -\infty, & \text{if } x = -\infty \text{ or } y = \infty, \\ \infty, & \text{if } (x = \infty \text{ and } y \neq \infty) \text{ or } (y = -\infty \text{ and } x \neq -\infty). \end{cases}$$

Definition 3.2 (Saturated Post-Hoc Optimization)
Given a transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$, a cost function *cost* for $\mathcal{T}$, and a tuple of abstraction heuristics $\mathcal{H}$ for $\mathcal{T}$, the heuristic value $h^{\text{SPhO}}(s)$ for a state s is the *objective value* of the *SPhO LP*:

$$\begin{aligned} \min_{Y} \sum_{\ell \in L_{\mathit{rel}}} \mathit{cost}(\ell) \cdot Y_\ell \qquad & \text{subject to} \\ \sum_{\ell \in L_{\mathit{rel}}} \mathit{mscf}_h(\ell) \cdot Y_\ell \geq h(s) \qquad & \text{for all } h \in \mathcal{H} \\ Y_\ell \geq 0 \qquad & \text{for all } \ell \in L_{\mathit{rel}} \end{aligned} \tag{3.1}$$

where $L_{\mathit{rel}} = L \setminus L_{-\infty}$ and $L_{-\infty}$ is the set of labels $\ell \in L$ for which there exists a heuristic $h \in \mathcal{H}$ such that $\mathit{mscf}_h(\ell) = -\infty$.

The labels $L_{-\infty}$ can be excluded from the SPhO LP because they do not affect the solution, and infinite values cannot be encoded directly in a linear program (Seipp et al. 2021). By h^{SPhO} we denote the heuristic that solves the SPhO LP for each evaluated state. We will also refer to this heuristic as *eager* SPhO when emphasizing the difference to other variants that will be introduced in the following sections.

The SPhO LP is strongly related to cost partitioning through its dual LP (Pommerening et al. 2013; Seipp et al. 2021).

Definition 3.3 (SPhO Dual)
Given a transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$, a cost function *cost* for $\mathcal{T}$, and a tuple of abstraction heuristics $\mathcal{H}$ for $\mathcal{T}$, the dual SPhO LP is

defined as:

$$\begin{aligned} \max_{w} \sum_{h \in \mathcal{H}} h(s) \cdot w_h \quad & \text{subject to} \\ \sum_{h \in \mathcal{H}} \mathit{mscf}_h(\ell) \cdot w_h \leq \mathit{cost}(\ell) \quad & \text{for all } \ell \in L_{\mathit{rel}} \\ w_h \geq 0 \quad & \text{for all } h \in \mathcal{H}. \end{aligned}$$

Intuitively, this dual LP maximizes a weight w_h for each abstraction heuristic $h \in \mathcal{H}$. In the resulting cost partition $\mathcal{C}$, each heuristic $h \in \mathcal{H}$ is assigned the cost function cost_h, where $\mathit{cost}_h(\ell) = \mathit{mscf}_h(\ell) \cdot w_h$. The value $h^{\text{SPhO}}(s)$ can then be computed as $h^{\text{SPhO}}(s) = h^{\mathcal{C}}(s) = \sum_{h \in \mathcal{H}} w_h \cdot h(s)$. We call such cost partitions, where each minimum saturated cost function is scaled by its own factor, *SPhO cost partitions*. The SPhO cost partition $\mathcal{C}$ can also be extracted from the operator counting version by computing the shadow prices $\bar{y}_i$, since these are the coefficients of the dual LP. The SPhO cost partitioning heuristic is then calculated as: $h^{\text{SPhO}}(s) = \sum_{i=1}^{n} \bar{y}_i h_i(s)$.

This perspective also explains why SPhO is an efficient non-optimal cost partitioning strategy. Since the cost functions of each abstraction heuristic are only scaled, the shortest path in each heuristic remains the same, and the abstraction heuristic is also just scaled by the same factor. This means that the abstract goal distance lookup tables only need be computed once and remain valid for all states during search. Dual feasibility corresponds to distributing each operator's cost across abstractions so that the per-operator shares sum to at most the original cost, i.e., a cost partition. We will exploit sensitivity analysis of the minimization LP to determine when its optimal basis (and thus the cost partitioning) can be reused without solving the LP again.

4 Safe Cover Rules for SPhO Optimization

When evaluating the SPhO heuristic, calling the linear program solver for every state evaluation is the clear bottleneck, whereas evaluating a fixed cost partition (i.e., computing $h^{\mathcal{C}}(s)$ by abstract goal distance lookups) is very fast. The practical motivation of this work is therefore to improve the efficiency of the SPhO heuristic by reducing the number of linear program optimization calls.

To avoid solving the SPhO LP from Definition 3.2 for each evaluated state, we define *cover rules* that decide whether an LP solution can be reused for a different state. The goal is to find cover rules that will only reuse LP solutions for states where they evaluate to the best possible heuristic value.

Definition 4.1 (Cover rule)
Let $\mathcal{T}$ be a transition system with states S. A *cover rule* for $\mathcal{T}$ is a tuple $\langle I, \mathit{Extract}, \mathit{Adapt} \rangle$ where I is the type of information that the rule stores about known solutions (e.g., optimal basis vectors), $\mathit{Extract} : \mathit{Sols} \times S \to I$ is a function such that $\mathit{Extract}(\mathit{sol}, s)$ is the information stored based on solution *sol* of the SPhO LP for state s, and finally, $\mathit{Adapt} : I \times S \to \mathbb{R} \cup \{-\infty, \infty\}$ is a partial function mapping $\langle \mathit{info}, s \rangle$

Algorithm 1 Lazy saturated post-hoc optimization with a cover rule $\langle I, \mathit{Extract}, \mathit{Adapt} \rangle$. For a given state s, check whether any previously computed information covers s. If so, (possibly) adapt the information to s and return the adapted value. Otherwise, solve the SPhO LP for s and store the relevant information.

```
1: Infos ← ∅
2: function LazySPhO(s)
3:     if there exists info ∈ Infos that covers s then
4:         return Adapt(info, s)
5:     sol ← solve SPhO LP for s
6:     Infos ← Infos ∪ {Extract(sol, s)}
7:     return GetObjectiveValue(sol)
```

to a heuristic value for s based on the stored information *info*. We say that *info* $\in I$ *covers* state $s \in S$ if *Adapt* is defined for (info, s).

To use the cover rules during an A^* search, we introduce a variant of the SPhO heuristic, called *lazy* SPhO, which is parameterized by a cover rule and only solves LPs for states that are not covered by any previously obtained LP solutions. Algorithm 1 shows it as pseudocode.

We say that a cover rule is *safe* if adapting a previously stored solution reproduces h^{SPhO} without loss of optimality.

Definition 4.2 (Safe)
A cover rule $\langle I, \mathit{Extract}, \mathit{Adapt} \rangle$ is safe if $\mathit{Adapt}(\mathit{Extract}(\mathit{sol}, s), s') = h^{\text{SPhO}}(s')$ for all states s, s' and all solutions *sol* of the SPhO LP for state s where $\mathit{Extract}(\mathit{sol}, s)$ covers s'.

Proposition 4.1 *Any lazy SPhO heuristic that uses a* safe *cover rule is equivalent to SPhO.*

Proof. For any state s', if some stored information *info* covers s', then $\mathit{info} = \mathit{Extract}(\mathit{sol}, s)$ for some state s and one of its LP solutions *sol*. The safety of the cover rule then shows $\mathit{Adapt}(\mathit{info}, s') = h^{\text{SPhO}}(s')$. Otherwise, if no stored solution covers s', lazy SPhO computes and returns $h^{\text{SPhO}}(s')$. □

Note that lazy SPhO does not necessarily minimize the number of LP solver calls required during search. Each stored solution covers a subset of states, so finding the smallest set of solutions that together

cover all states evaluated during the search corresponds to a set cover problem. Lazy SPhO greedily approximates this set cover problem in a way that can be performed online during the search. We discuss the properties of this approximation in more detail in Section 4.5.

We now present four safe cover rules that can be used to test whether an *info* covers a state $s \in S$, as used in Algorithm 1. Each subsequent rule is stronger than the previous one, allowing us to reuse a solution for a larger set of states. We introduce the following partial order between cover rules to state this formally.

Definition 4.3 (Cover Rule Generalization)
Let $R_1 = \langle I_1, \mathit{Extract}_1, \mathit{Adapt}_1 \rangle$ and $R_2 = \langle I_2, \mathit{Extract}_2, \mathit{Adapt}_2 \rangle$ be two cover rules. We say that R_2 *generalizes* R_1 if the following hold for all states $s, s' \in S$: if $\mathit{Extract}_1(\mathit{sol}, s)$ covers s' according to R_1 then $\mathit{Extract}_2(\mathit{sol}, s)$ also covers s' according to R_2, where *sol* is the SPhO LP solution for s. We say that R_2 *strictly generalizes* R_1 if R_2 generalizes R_1 and there is at least one combination of $s, s' \in S$, and *sol* where $\mathit{Extract}_2(\mathit{sol}, s)$ covers s' but $\mathit{Extract}_1(\mathit{sol}, s)$ does not.

4.1 Equal Abstract Goal Distances

One simple way to detect that an LP solution *sol* for s covers a different state s' is by checking whether the two states induce the same SPhO LP (Definition 3.2). When optimizing the SPhO LP for s and s', the only LP parameters that change are the abstract goal distances. Thus, if the two states have the same abstract goal distances, the resulting LPs will be identical.

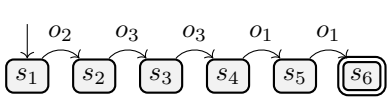
Cover Rule 1 (Eqdist Cover Rule)
Let $\mathcal{T}$ be a transition system and $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ a tuple of n heuristics for $\mathcal{T}$. Then eqdist is the cover rule $\langle \mathbb{R}_\infty^n \times \mathbb{R}_\infty, \mathit{Extract}_{\text{eqdist}}, \mathit{Adapt}_{\text{eqdist}} \rangle$ with $\mathit{Extract}_{\text{eqdist}}(\mathit{sol}, s) = \langle h_1(s), \dots, h_n(s), h^{\text{SPhO}}(s) \rangle$. A value $\mathit{info} = \langle d_1, \dots, d_n, d \rangle$ covers a state s' if and only if $d_i = h_i(s')$ for all $1 \leq i \leq n$, and in this case $\mathit{Adapt}_{\text{eqdist}}(\mathit{info}, s') = d$.

By $h^{\text{SPhO}}_{\text{eqdist}}$ we denote the lazy SPhO heuristic using Cover Rule 1.

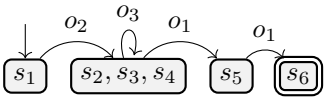
Theorem 4.2 *The cover rule* eqdist *is safe.*

Proof. Consider states s, s' and a solution *sol* of the SPhO LP for s such that *info* $=$ *Extract*$_{\text{eqdist}}(\mathit{sol}, s)$ covers s'. By definition, *info* $= \langle h_1(s), \ldots, h_n(s), h^{\text{SPhO}}(s)\rangle$ and since *info* covers s', $h_i(s) = h_i(s')$ for all $1 \leq i \leq n$. Therefore, the SPhO LPs for s and s' are identical and *Adapt*(*info*, s') $= h^{\text{SPhO}}(s) = h^{\text{SPhO}}(s')$. □

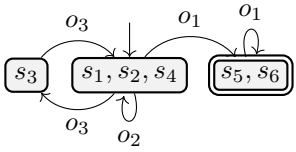
We demonstrate Cover Rule 1 and the rules below with the example in Figure 4.1. Intuitively, Cover Rule 1 avoids unnecessary LP computations when the used abstractions cannot distinguish between two states based solely on goal distances. States s_2 and s_4 in Figure 4.1 exemplify this: both abstractions fail to distinguish them. Consequently, $h^{\text{SPhO}}_{\text{eqdist}}$ skips computing the LP for s_4, because its abstract goal distances (2 and 1) are the same as those for s_2, which has already been evaluated.



(a) State Space



(b) Abstraction A



(c) Abstraction B

$$\begin{aligned} \min_x \; & x_1 + x_2 + x_3 \\ \text{subject to } & x_1 + x_2 \geq h^A(s) \\ & x_1 + x_3 \geq h^B(s) \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

(d) SPhO LP for the example

	h^A	h^B
s_1	**3**	**1**
s_2	**2**	**1**
s_3	**2**	**2**
s_4	**2**	**1**
s_5	**1**	**0**

(e) Heuristic values

	cost	$mscf_A$	$mscf_B$
o_1	**1**	**1**	**1**
o_2	**1**	**1**	**0**
o_3	**1**	**0**	**1**

(f) Cost functions

state	abstraction	RHS L	RHS value	RHS U	$\mathcal{B}$
s_1	h^A	**1**	**3**	∞	$\begin{bmatrix}1\\2\end{bmatrix}$
	h^B	**0**	**1**	**3**	
s_3	h^A	**2**	**2**	∞	$\begin{bmatrix}1\\4\end{bmatrix}$
	h^B	$-\infty$	**2**	**2**	

(g) Sensitivity information for selected states

Figure 4.1: Example task with two abstractions for showcasing cover rule generalization. The number of computed cost partitions on this task decreases with each stronger cover rule.

4.2 Sensitivity Analysis of the SPhO LP

The Cover Rule 1 presented above can never detect reuse for states with a different heuristic value than the state the solution was computed for. However, sensitivity analysis (Section 2.2) allows us to check whether a basis of an LP solution computed for state s remains optimal for a new state s'. This can be directly transformed into a cover rule that can detect unnecessary LP computations even if s and s' have different SPhO heuristic values. Using the sensitivity analysis formula from Proposition 2.3 we obtain another cover rule:

Cover Rule 2 (Range Cover Rule)
Let $\mathcal{T}$ be a transition system and $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ a tuple of heuristics for $\mathcal{T}$. Then range is the cover rule $\langle \mathbb{R}_\infty^n \times \mathbb{R}_\infty^n \times \mathbb{R}_\infty^n \times \mathbb{R}_\infty^n \times \mathbb{R}_\infty, \textit{Extract}_{\text{range}}, \textit{Adapt}_{\text{range}} \rangle$. Here $\textit{Extract}_{\text{range}}(\textit{sol}, s) = \langle h_1(s), \dots, h_n(s), L, U, \bar{y}, h^{\text{SPhO}}(s) \rangle$, where L, U are the sensitivity bounds from Proposition 2.3. A value $\textit{info} = \langle d_1, \dots, d_n, L, U, \bar{y}, d \rangle$ covers state $s' \in S$ if there exists at most one abstraction $h_i \in \mathcal{H}$ with $h_i(s') \neq d_i$ while $h_j(s') = d_j$ for all other $h_j \in \mathcal{H} \setminus \{h_i\}$, and the changed value satisfies $L_i \leq h_i(s') \leq U_i$. If no value changes, then $\textit{Adapt}_{\text{range}}(\textit{info}, s') = d$, otherwise, the new objective value can be computed as $\textit{Adapt}_{\text{range}}(\textit{info}, s') = d + \bar{y}_i(h_i(s') - d_i)$.

By $h^{\text{SPhO}}_{\text{range}}$ we denote the lazy SPhO heuristic on the grouped SPhO LP using Cover Rule 2.

Theorem 4.3 *The cover rule* range *is safe.*

Proof. Consider two states $s, s' \in S$ and a solution *sol* for the SPhO LP for s such that $\textit{info} = \textit{Extract}_{\text{range}}(\textit{sol}, s)$ covers s'. Since *info* covers s', Proposition 2.3 guarantees that *sol* is optimal for s' and there is only one abstraction heuristic h_i such that $h_j(s) = h_j(s')$ for all $h_j \in \mathcal{H} \setminus h_i$. The new heuristic value of *sol* for s' can be computed from the dual objective $h^{\text{SPhO}}(s') = \sum_{j=1}^n \bar{y}_j h_j(s') = h^{\text{SPhO}}(s) + \sum_{j=1}^n \bar{y}_j(h_j(s') - h_j(s))$. Since $h_j(s') = h_j(s)$ for all $j \neq i$, $h^{\text{SPhO}}(s) + \sum_{j=1}^n \bar{y}_j(h_j(s') - h_j(s)) = h^{\text{SPhO}}(s) + \bar{y}_i(h_i(s') - h_i(s)) = \textit{Adapt}_{\text{range}}(\textit{info}, s')$. □

Corollary 4.4 *Cover Rule* range *generalizes* eqdist.

Proof. $\mathit{Extract}_{\text{eqdist}}(\mathit{sol}, s) = \langle h_1(s), \dots, h_n(s), h^{\text{SPhO}}(s)\rangle$ covers all states s' where $d_i = h_i(s')$ for all $1 \leq i \leq n$. Since in all these cases there exists no abstraction $h_i(s') \neq d_i$, all heuristics h_i will satisfy $L_i \leq h_i(s') \leq U_i$, so $\mathit{Extract}_{\text{range}}$ will cover all states that $\mathit{Extract}_{\text{eqdist}}(\mathit{sol}, s)$ does. □

The running example in Figure 4.1 shows a situation where Cover Rule 2 leads to fewer solved LPs than cover rule eqdist by using information from sensitivity analysis. We write $\mathit{info}^{s_x}_{\text{range}}$ for the information stored by the range cover rule in state s_x. During the search, $h^{\text{SPhO}}_{\text{range}}$will compute the LP for s_1 and store $\mathit{info}^{s_1}_{\text{range}}$ from the information shown in the s_1 column of Figure 4.1g. Since only the abstract goal distance for h^B changes from s_1 to s_2 and its new value of 2 is within the lower bound L of the sensitivity analysis for s_1, $\mathit{info}^{s_1}_{\text{range}}$ covers s_2. In state s_3, both abstractions yield a different heuristic value than in s_1, so $\mathit{info}^{s_1}_{\text{range}}$ does not cover s_3 and the algorithm computes $\mathit{info}^{s_3}_{\text{range}}$. $\mathit{info}^{s_3}_{\text{range}}$ then covers s_4 since the change in h^B is within the lower bound. Finally, none of the stored information covers s_5, since its distance tuple does not share a heuristic value with any other state.

Corollary 4.5 *Cover Rule* range strictly *generalizes* eqdist.

Proof. Since Corollary 4.4 shows generalization, the state pair s_1 and s_2 of the running example proves it is strict. □

4.3 Sensitivity Analysis using the 100% Rule

For the third cover rule, we consider the 100% rule from Proposition 2.4, which yields a generalization of Cover Rule 2.

Cover Rule 3 (100% Cover Rule)
Let $\mathcal{T}$ be a transition system and $\mathcal{H} = \langle h_1, \dots, h_n\rangle$ a tuple of heuristics for $\mathcal{T}$. Then 100% is the cover rule $\langle \mathbb{R}^n_\infty \times \mathbb{R}^n_\infty \times \mathbb{R}^n_\infty \times \mathbb{R}^n_\infty \times \mathbb{R}_\infty, \mathit{Extract}_{\text{100\%}}, \mathit{Adapt}_{\text{100\%}}\rangle$. Here $\mathit{Extract}_{\text{100\%}}(\mathit{sol}, s) = \langle h_1(s), \dots, h_n(s), L, U, \bar{y}, h^{\text{SPhO}}(s)\rangle$, where L, U are the sensitivity bounds from Proposition 2.4. A value $\mathit{info} = \langle d_1, \dots, d_n, L, U, \bar{y}, d\rangle$ covers state $s' \in S$ if $\sum_{i=1}^n \lambda_i \leq 1$, where $\lambda_i = \frac{h_i(s') - d_i}{\Delta d_i^*}$ and

$$\Delta d_i^* = \begin{cases} U_i - d_i & \text{if } h_i(s') - d_i \geq 0. \\ L_i - d_i & \text{otherwise.} \end{cases}$$

Division by $\pm\infty$ is defined as zero in this case. The new objective value can be computed as $\mathit{Adapt}_{100\%}(\mathit{info}, s') = d + \sum_{i=1}^{n} \bar{y}_i(h_i(s') - d_i)$.

By $h^{\text{SPhO}}_{100\%}$ we denote the lazy SPhO heuristic on the grouped SPhO LP using Cover Rule 3.

Theorem 4.6 *The cover rule* 100% *is safe.*

Proof. Consider two states $s, s' \in S$ and a solution *sol* for the SPhO LP for s such that $\mathit{info} = \mathit{Extract}_{100\%}(\mathit{sol}, s)$ covers s'. Since *info* covers s', Proposition 2.4 guarantees that *sol* is optimal for s'. The new heuristic value of *sol* for s' can be computed from the dual objective $h^{\text{SPhO}}(s') = \sum_{j=1}^{n} \bar{y}_j h_j(s') = h^{\text{SPhO}}(s) + \sum_{j=1}^{n} \bar{y}_j(h_j(s') - h_j(s)) = \mathit{Adapt}_{100\%}(\mathit{info}, s')$. □

Corollary 4.7 *Cover Rule* 100% *generalizes* range.

Proof. If the cover rule $\mathit{Extract}_{\text{range}}(\mathit{sol}, s) = \langle h_1(s), \dots, h_n(s), L, U, \bar{y}, h^{\text{SPhO}}(s)\rangle$ covers state s', there exists at most one abstraction $h_i \in \mathcal{H}$ with $h_i(s') \neq d_i$ while $h_j(s') = d_j$ for all other $h_j \in \mathcal{H} \setminus \{h_i\}$. Therefore, all $\lambda_j = 0$ for all $j \neq i$. Further, $L_i \leq h_i(s') \leq U_i$, therefore $L_i - d_i \leq h_i(s') - d_i \leq U_i - d_i$. It follows that if $h_i(s') \geq d_i$, $\frac{h_i(s')-d_i}{U_i-d_i} \leq 1$, otherwise $\frac{h_i(s')-d_i}{L_i-d_i} \leq 1$, so $\sum_{k=1}^{n} \lambda_k \leq 1$. □

As an example, we again consider the task from Figure 4.1 and the sensitivity information for state s_1. Since 100% generalizes range $\mathit{info}^{s_1}_{100\%}$ covers s_2 and s_4. In contrast to Cover Rule 2, $\mathit{info}^{s_1}_{100\%}$ covers s_3, since both changes lie within their bounds and $\frac{2-3}{1-3} + \frac{2-1}{3-1} \leq 1$. Finally, $\mathit{info}^{s_1}_{100\%}$ does not cover s_5 since $\frac{1-3}{1-3} + \frac{0-1}{0-1} \not\leq 1$.

Corollary 4.8 *Cover Rule* 100% strictly *generalizes* range.

Proof. Since Corollary 4.7 shows generalization, the state pair s_1 and s_3 of the running example proves it is strict. □

4.4 Exact Sensitivity Analysis

The previously presented cover rules are approximations of the reusability of a linear program basis. Our final rule checks exact reusability of a basis under changes in the constraint bounds. It is therefore the strongest cover rule that can be derived from looking

at the solution basis of the SPhO LP. A basis remains optimal under changes in its constraint bounds Δb if and only if all basic variables remain positive: $x_{\mathcal{B}} \geq 0$ (see Equation 2.2 and Section 2.2.2).

Cover Rule 4 (Exact Cover Rule)
Let $\mathcal{T}$ be a transition system and $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ a tuple of heuristics for $\mathcal{T}$. Then exact is the cover rule $\langle \mathbb{R}_\infty^{n \times n} \times \mathbb{R}_\infty^n \times \mathbb{R}_\infty, \mathit{Extract}_{\text{exact}}, \mathit{Adapt}_{\text{exact}} \rangle$. Here $\mathit{Extract}_{\text{exact}}(\mathit{sol}, s) = \langle B^{-1}, \bar{y}, h^{\text{SPhO}}(s) \rangle$, where B^{-1} is the inverse basis matrix of the LP solution *sol*. A value $\mathit{info} = \langle B^{-1}, \bar{y}, d \rangle$ covers state $s' \in S$ if $B^{-1} \begin{bmatrix} h_1(s') \\ \dots \\ h_n(s') \end{bmatrix} \geq 0$. The new objective value can be computed as $\mathit{Adapt}_{\text{exact}}(\mathit{info}, s') = \sum_{i=1}^{n} \bar{y}_i h_i(s')$.

By $h^{\text{SPhO}}_{\text{exact}}$ we denote the lazy SPhO heuristic on the grouped SPhO LP using Cover Rule 4. We call this approach the *exact sensitivity analysis* for the h^{SPhO} heuristic, since it allows us to compute h^{SPhO} while reusing a previous basis for *exactly* those states s where this is possible.

Theorem 4.9 *The cover rule* exact *is safe.*

Proof. Consider two states $s, s' \in S$ and a solution *sol* for the SPhO LP for s such that $\mathit{info} = \mathit{Extract}_{\text{exact}}(\mathit{sol}, s)$ covers s'. Since *info* covers s', the feasibility condition (Equation 2.2) guarantees that *sol* is optimal for s'. The new heuristic value of *sol* for s' can be computed from the dual objective $h^{\text{SPhO}}(s') = \sum_{j=1}^{n} \bar{y}_j h_j(s') = \mathit{Adapt}_{\text{exact}}(\mathit{info}, s')$. □

Corollary 4.10 *Cover Rule* exact *generalizes* 100%.

Proof. See Bradley et al. (1977), Section 3.7, for a proof that the 100% cover rule only covers states s' where $x_{\mathcal{B}} + \Delta x_{\mathcal{B}} \geq 0$. □

In our running example, it is only necessary to look at the state pair s_1 and s_5, since generalization already shows that $\mathit{info}^{s_1}_{\text{exact}}$ will cover all other states. The basis $\mathcal{B}$ for s_1 is $\left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right]$, $B = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and $B^{-1} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & -1 \end{smallmatrix}\right]$, therefore exact checks that for s_5 $\left[\begin{smallmatrix} 0 & 1 \\ 1 & -1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] \geq 0$, which holds, so $\mathit{info}^{s_1}_{\text{exact}}$ covers also s_5.

Corollary 4.11 *Cover Rule* exact strictly *generalizes* 100%.

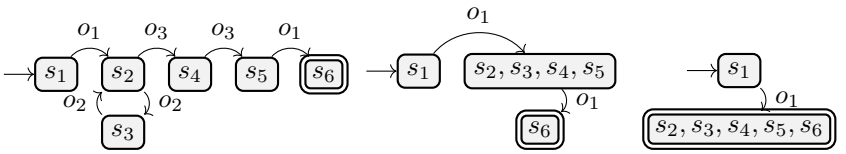
Proof. Since Corollary 4.10 shows generalization, the state pair s_1 and s_5 of the running example proves it is strict. □

4.5 Global Behavior of Greedy Cover Rules

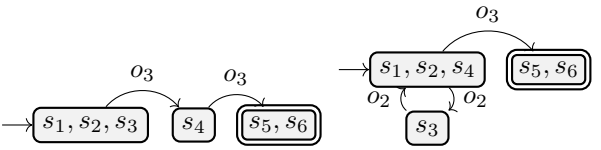
Formally, minimizing the number of LP solver calls can be viewed as a set cover problem. Each LP solution corresponds to a set containing all states for which that solution yields the optimal SPhO heuristic value. Since LP solutions can cover multiple states and multiple LP solutions may cover the same state, computing the minimum number of required SPhO LPs constitutes a non-trivial set cover instance. More precisely, because the planner evaluates states sequentially, our problem corresponds to the *online* version of set cover (Alon et al. 2003).

An important consequence of the approximative nature of our algorithm is that a locally stronger cover rule does not necessarily lead to a globally better solution. In other words, even a strictly more general cover rule can cause lazy SPhO to compute more LP solutions. Figure 4.2 shows such a situation with the *range* and *100%* cover rules. Evaluating the states in numerical order leads *range* to compute an LP for state s_1 and s_2, since both h^A and h^B change their heuristic values. $\mathit{info}^{s_1}_{\text{range}}$ then covers s_3 and s_4 since only one abstraction changes their heuristic value and both lie inside their respective sensitivity ranges. s_5 is not covered by $\mathit{info}^{s_2}_{\text{range}}$ again since two abstract goal distances change. This means that cover rule *range* solves three linear programs in this example.

Cover Rule *100%* in contrast does not compute an LP for state s_2 since $\mathit{info}^{s_1}_{100\%}$ covers s_2. This is the case because the change in h^A is inside the sensitivity bound and the lower bound for h^B is $-\infty$, so it does not contribute to the λ bound. $\mathit{info}^{s_1}_{100\%}$ does not cover s_3 and s_4, even though all changes are inside the sensitivity bounds, since the change in h^A already produces a λ of one and both h^C and h^D do not change toward an infinite bound. Additionally, $\mathit{info}^{s_3}_{100\%}$ does not cover s_4 as well since the change in h^C is outside its bound. Finally, none of the previously computed solutions cover s_5, since $h^C(s_5) = 0$ is never inside the sensitivity bounds. This means that cover rule *100%* computes four LP solutions, which is one more than *range*, even though *100%* strictly generalizes *range*.



(a) State Space (b) Abstraction A (c) Abstraction B



(d) Abstraction C (e) Abstraction D

	h^A	h^B	h^C	h^D
s_1	**2**	**1**	**2**	**1**
s_2	**1**	**0**	**2**	**1**
s_3	**1**	**0**	**2**	**2**
s_4	**1**	**0**	**1**	**1**
s_5	**1**	**0**	**0**	**0**

(f) Heuristic values

$$\begin{aligned} \min_x \; & x_1 + x_2 + x_3 \\ \text{subject to } & x_1 \geq h^A(s) \\ & x_1 \geq h^B(s) \\ & x_3 \geq h^C(s) \\ & x_2 + x_3 \geq h^D(s) \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

(g) SPhO LP for the example

Figure 4.2: An adversarial example where a strictly more general cover rule leads to more LP computations. Computing lazy SPhO for the numerical evaluation order leads to three solved LPs with the range cover rule, but four with the 100% cover rule. The Figure continues on the next page.

		RHS		
state	abstraction	L	value	U
s_1	h^A	1	2	∞
	h^B	$-\infty$	1	2
	h^C	1	2	∞
	h^D	$-\infty$	1	2
s_2	h^A	0	1	∞
	h^B	$-\infty$	0	1
	h^C	1	2	∞
	h^D	$-\infty$	1	2
s_3	h^A	0	1	∞
	h^B	$-\infty$	0	1
	h^C	2	2	∞
	h^D	$-\infty$	2	2
s_4	h^A	0	1	∞
	h^B	$-\infty$	0	1
	h^C	1	1	∞
	h^D	$-\infty$	1	1

(h) Sensitivity information for selected states

	cost	$mscf_A$	$mscf_B$	$mscf_C$	$mscf_D$
o_1	1	1	1	0	0
o_2	1	0	0	0	1
o_3	1	0	0	1	1

(i) Cost functions

Continuation of Figure 4.2

4.6 Cover Rule Experiments

We empirically test the main theoretical claims: (i) The dominance relation among our cover rules: Each stronger rule should use fewer solutions than a weaker one in most cases. (ii) The safety of our cover rules: Each run of lazy SPhO should expand exactly the same states as standard SPhO. We first show the safety of our cover rules, then analyze the number of solved LPs required by the different SPhO heuristic variants, before turning to the performance in terms of solved problems and runtime.

4.6.1 Evaluations and Expansions

To verify that our lazy SPhO algorithm is not only in theory equivalent to eager SPhO, we compare the number of evaluations and expansions over our benchmark problems. We find that they are the same over all versions as shown in Table 4.1, showing that our implementation correctly captures the theoretical equivalence.

	evaluations (mean)	expansions (mean)
h^{SPhO}	1466917.02	831233.62
$h^{\text{SPhO}}_{\text{eqdist}}$	1466917.02	831233.62
$h^{\text{SPhO}}_{\text{range}}$	1466917.02	831233.62
$h^{\text{SPhO}}_{100\%}$	1466917.02	831233.62
$h^{\text{SPhO}}_{\text{exact}}$	1466917.02	831233.62

Table 4.1: Mean of the state evaluations and expansions of the SPhO heuristic and our lazy variants. It demonstrates that our implementations compute the same heuristics as h^{SPhO}.

4.6.2 Number of Solved LPs

Figures 4.3 to 4.6 shows a comparison of the number of LP computations needed by the new lazy SPhO heuristic variants in comparison to the eager SPhO heuristic from the literature. Overall, we typically observe reductions of several orders of magnitude in the number of LPs to solve, while preserving heuristic accuracy. However, the effect size varies by domain and task. Although the successive generalization of

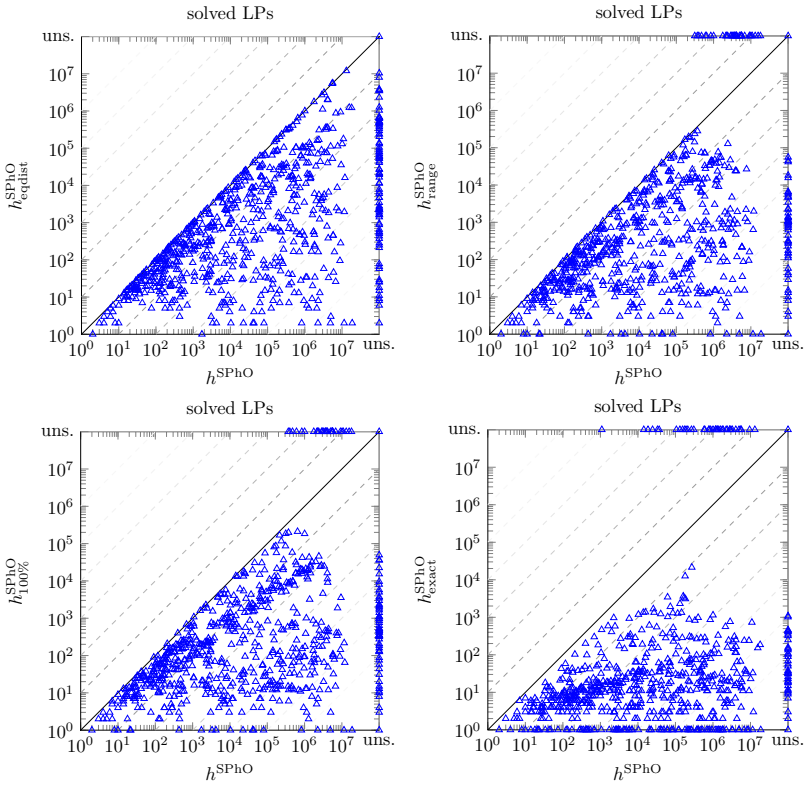


Figure 4.3: A comparison of the number of solved LPs between the eager SPhO heuristic h^{SPhO} and lazy SPhO with all presented cover rules on a logarithmic scale. Each point represents a comparison of two SPhO configurations on one planning task. Points below the diagonal indicate that the algorithm on the y-axis solves fewer LPs than the algorithm on the x-axis. Tasks that are not solved within the resource limits appear at "uns." on the axes.

our cover rules is not guaranteed to result in strictly fewer solved LPs, there is a significant difference between all our cover rules. There is only one task where we observe that $h^{\text{SPhO}}_{100\%}$ solves more LPs than $h^{\text{SPhO}}_{\text{range}}$. In all other tasks and comparisons, the generalization translates to fewer LP solver calls. The comparison between the cover rules shows that the difference between h^{SPhO} and $h^{\text{SPhO}}_{\text{eqdist}}$ is the biggest, and that $h^{\text{SPhO}}_{\text{exact}}$ is again a big step up from $h^{\text{SPhO}}_{100\%}$.

Figure 4.7 shows the geometric mean of the required LP solver calls for all SPhO heuristic variants on *all* considered IPC benchmark do-

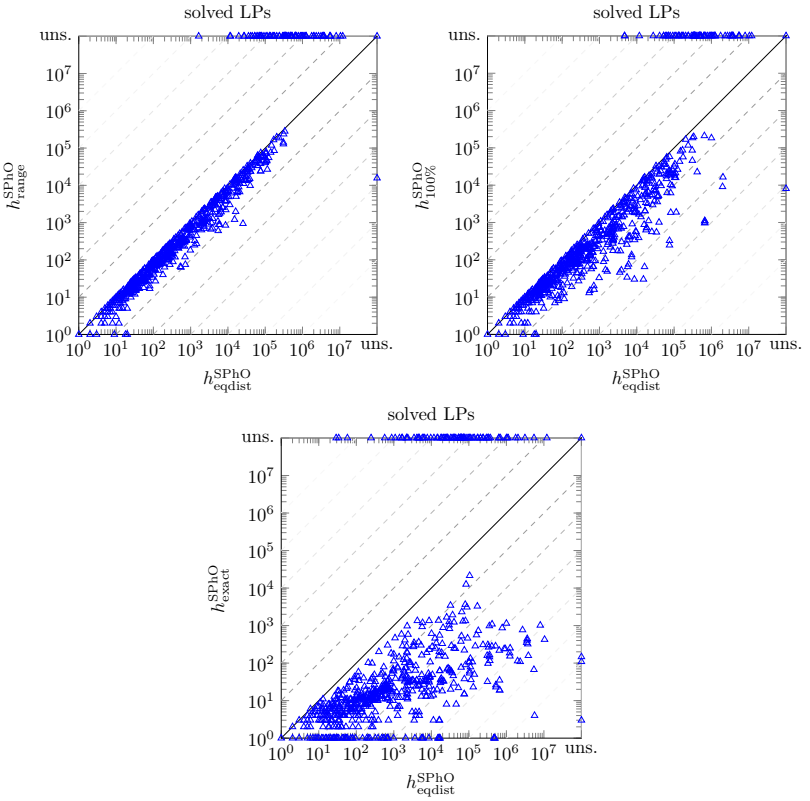


Figure 4.4: A comparison of the number of solved LPs between lazy SPhO with the *tuple* cover rule against all other presented cover rules. The plots work the same as in Figure 4.3.

mains and the Mystery and VisitAll-14 domains. Note that the figure only considers tasks solved by all variants. Our new approaches, which skip unneeded LP computations, reduce the average number of solved LPs by orders of magnitude compared to the eager h^{SPhO} approach (with variability across domains). However, the effectiveness of the presented cover rules, i.e., the number of LP computations we can avoid, depends on the planning domain. In the Mystery domain, we reduce the number of solved LPs by about three orders of magnitude. In the VisitAll-14 domain, however, all cover rules besides $h^{\text{SPhO}}_{100\%}$ and $h^{\text{SPhO}}_{\text{exact}}$ make no difference and only $h^{\text{SPhO}}_{\text{exact}}$ is able to significantly reduce the number of solved LPs.

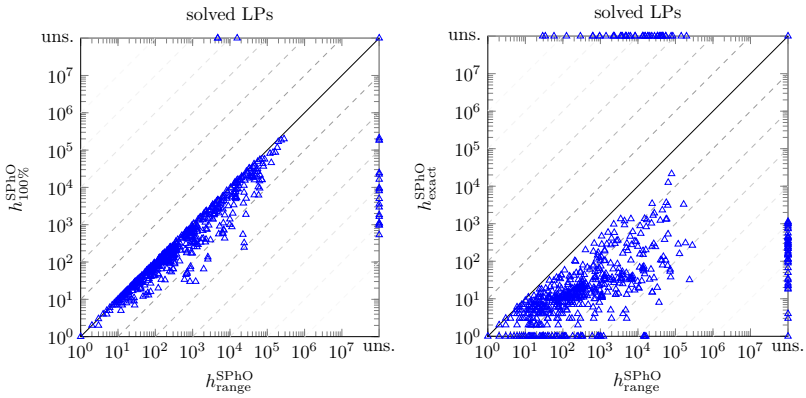


Figure 4.5: A comparison of the number of solved LPs between lazy SPhO with the *range* cover rule against the stronger presented cover rules. The plots work the same as in Figure 4.3.

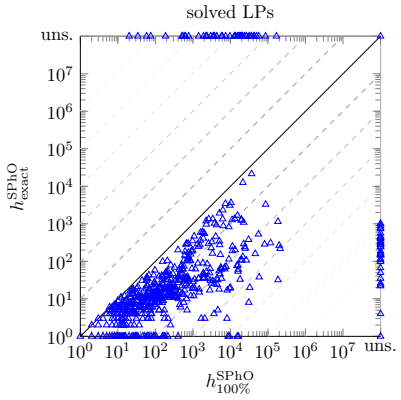


Figure 4.6: A comparison of the number of solved LPs between lazy SPhO with the *percent* cover rule against the *exact* cover rule. The plots work the same as in Figure 4.3.

4.6.3 Coverage and Runtime

A natural question is whether the observed reduction in the number of LP solver calls to compute the SPhO heuristic is reflected in the number of solved tasks, i.e., the planner coverage, and the planner runtime.

Table 4.2 shows the total number of solved planning tasks using the different SPhO variants. We see that all our new lazy approaches solve many more problems than eager h^{SPhO} due to the lower number of LPs solved. However, we also see that the heuristic with the lowest number

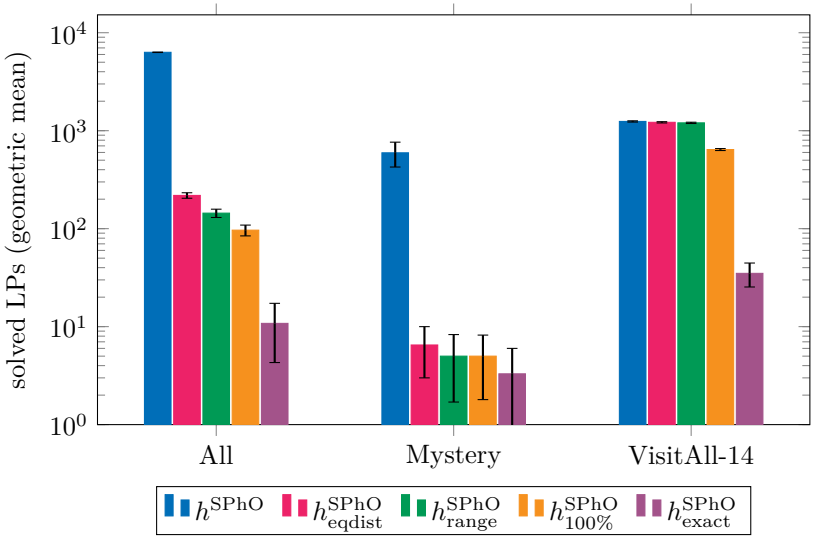


Figure 4.7: Comparison of the number of solved LPs (geometric mean) for eager h^{SPhO} and our lazy variants on a logarithmic scale over all domains and the Mystery and VisitAll-14 domains.

h^{SPhO}	$h^{\text{SPhO}}_{\text{eqdist}}$	$h^{\text{SPhO}}_{\text{range}}$	$h^{\text{SPhO}}_{100\%}$	$h^{\text{SPhO}}_{\text{exact}}$
865	**975**	881	901	892

Table 4.2: Number of solved tasks (out of 1884) of h^{SPhO} and our new lazy variants.

of LP computations, $h^{\text{SPhO}}_{\text{exact}}$, does not solve the highest number of tasks. The reason for this is that the complexity of evaluating applicability of the different cover rules for a state varies greatly. Evaluating the $h^{\text{SPhO}}_{\text{eqdist}}$ heuristic requires constant overhead, regardless of how many LP solutions are stored, while the other cover rules need to loop over all stored cost partitions. This leads to $h^{\text{SPhO}}_{\text{eqdist}}$ having the highest coverage score among the online variants, even though it computes more LPs than all other cover rules.

The same is reflected in the runtime comparison in Figure 4.8. Only the simple cover rules show a clear advantage over SPhO. While the more expensive cover rules still pay off for most problems, but especially for problems where SPhO takes a long time, they can perform worse than eager SPhO. This shows that when these cover rules have

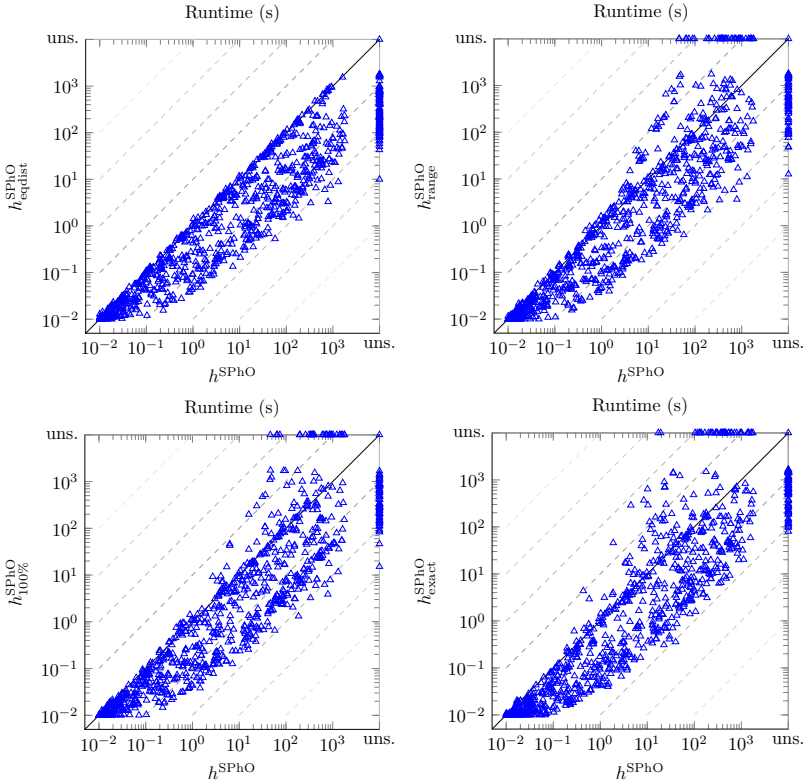


Figure 4.8: Runtime comparison between the eager SPhO heuristic h^{SPhO} and our lazy variants on a logarithmic scale. Problems that are not solved within the resource limits appear on the "uns." axes.

to evaluate many states and cost partitions the complexity of the evaluation can be a disadvantage.

5 Transforming the SPhO LP for Solution Reusability

In the previous chapter, we looked into when we can reuse the SPhO LP solutions and how to compute this reusability. We now look into reformulations of the SPhO LP that improve the effectiveness of sensitivity analysis and therefore improve the solution reusability.

5.1 The Effect of Degeneracy and Non-Uniqueness in SPhO

Until now, we looked at the reusability of LP solutions, but from a planning perspective we are actually interested in the reusability of a cost partition, since this results in the lowest number of LP solver calls. Problematically, our cover rules evaluate when a previously computed LP solution can be reused, not a cost partition. For non-degenerate and unique LP solutions this poses no problem, because under these conditions there is a one-to-one correspondence between LP solutions and cost partitions. Degeneracy and non-uniqueness, though, break this one-to-one correspondence, as shown in Section 2.3. Since our cover rules only compute the reusability of the LP basis, under degeneracy and non-uniqueness they are not guaranteed to compute the reusabil-

ity of a cost partition. This creates a mismatch that weakens all cover rules from the planning perspective.

As an example, consider three states: s_0, s_1, and s_2. Further, assume that the SPhO LP for state s_0 has two optimal bases, $\mathcal{B}^1$ and $\mathcal{B}^2$, and performing sensitivity analysis using basis $\mathcal{B}^i$ allows us to efficiently compute the heuristic value for state s_i. This is visualized in Figure 5.1. Usually, LP solvers only provide one of these optimal bases at a time. So regardless of which basis ($\mathcal{B}^1$ or $\mathcal{B}^2$) the solver returns, an additional LP computation becomes necessary to derive the heuristic values for both states s_1 and s_2.

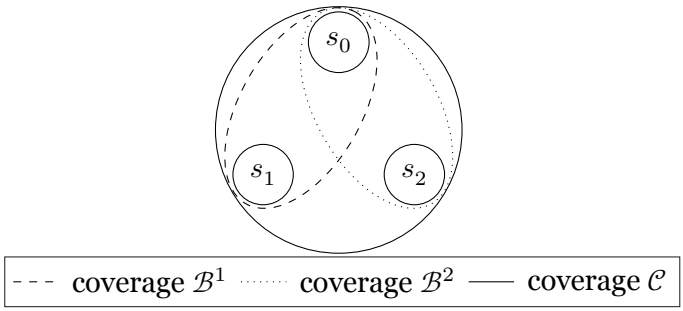


Figure 5.1: Diagram showing the potential mismatch between LP basis reusability and cost partition reusability if there exist multiple bases for the same cost partition $\mathcal{C}$. If two optimal bases for s_0 exist, it can be that they both only cover a subset of the states that $\mathcal{C}$ covers.

Additionally, non-uniqueness makes it important to reason about the alternative LP solutions of a state. So far, the topic of alternative optimal solutions for LP-based heuristics has not been discussed in the planning literature, because it is irrelevant when optimizing the LP for every state. However, when reusing LP solutions between different states, alternative solutions are of high interest, as their reusability can vary strongly. With the definitions of degeneracy and uniqueness, we formulate the relationship between sensitivity analysis and cost partitioning more precisely.

Theorem 5.1 *If an optimal LP solution sol for the SPhO LP for state s is non-degenerate and unique and computes the cost partition $\mathcal{C}$, exact sensitivity analysis covers all states s' where $\mathcal{C}$ remains optimal.*

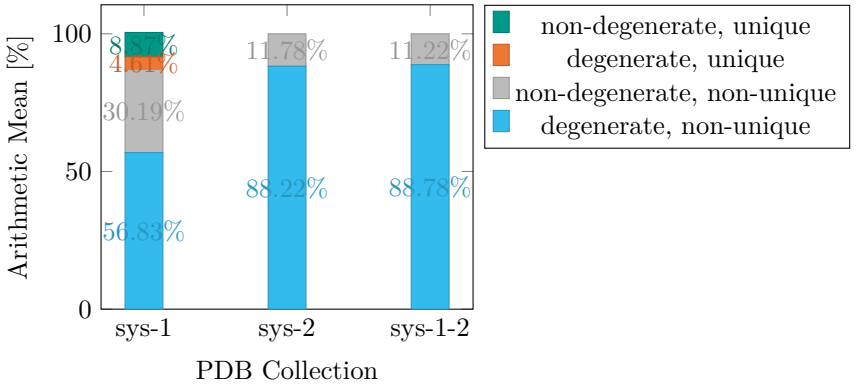


Figure 5.2: Average percentage of degenerate and non-unique solutions for the SPhO heuristic over different PDB sets. Sys-x are all systematic patterns (Pommerening et al. 2013; Pommerening et al. 2021) of size x, so sys-2 are only systematic patterns of size 2 and sys-1-2 is the combination of both patterns of size 1 and 2.

Proof. Since *sol* is non-degenerate and unique, there are no other solutions for the SPhO LP for state s. Therefore, there are also no other cost partitions besides $\mathcal{C}$ for s that yield $h^{\text{SPhO}}(s)$. As a consequence, *sol* is reusable for exactly those states s' for which $\mathcal{C}$ is optimal. □

We analyze the occurrence of degeneracy and uniqueness in SPhO LP solutions in Figure 5.2. It shows that the LPs computed for h^{SPhO} over different sets of pattern database heuristics have many alternative solutions. We tested both systematic patterns of size 1, 2, and combined, in contrast to just generating patterns up to a given size systematically. We wanted to investigate how subsets of patterns being present would affect degeneracy and non-uniqueness, since there is a high chance that patterns of size 1 are redundant when patterns of size 2 are used.

Surprisingly, the bar plot in Figure 5.2 shows that removing patterns of size 1 from the collection decreases the average number of degenerate solutions only very slightly. This means that the additional smaller patterns are not an important reason for the high number of alternative solutions. Considering only size-1 patterns has much fewer degenerate solutions than the other two variants and even gives rise to some non-degenerate unique solutions. Since patterns of size 1 don't share any variables with other patterns a lower number of degenerate

and non-unique solutions is expected, but it is still very high. These solutions suggest that there is a high amount of redundant information in the SPhO LP. We will therefore now show techniques that will remove redundancies from the LP.

5.2 Grouping Rows and Columns of the SPhO LP

Since degenerate and non-unique LP bases harm the performance of our cover rules, it is beneficial to reduce their occurrences. So, if we can reformulate the LP in a way that does not alter the computed heuristic value and at the same time reduces the number of optimal bases, ideally resulting in a single optimal basis (non-degeneracy and uniqueness), it becomes more likely that we can avoid the need for such additional LP computations. We are therefore interested in reformulating the SPhO LP to reduce the number of optimal bases by reducing degeneracy or non-uniqueness.

Although guaranteeing that all bases of a linear program are non-degenerate and unique is a very challenging problem (Matoušek and Škovroň 2007), simply reducing degeneracy and non-uniqueness in linear programs is easier. We reiterate that duplicate columns can lead to degenerate solutions and duplicate rows can lead to non-unique solutions (Sierksma and Zwols 2015), so removing redundancies will reduce the occurrences of degenerate and non-unique bases. Any removed constraint or variable reduces the number of bases of the LP. If the removed basis was part of an optimal solution, this reduction will improve the sensitivity analysis by at least reducing the pool of bases with slightly wrong bounds or even improving the bounds of the new bases. Additionally, removing redundancies from an LP will result in a smaller program that is usually faster to solve.

For the SPhO heuristic, duplicate rows (abstractions with identical minimum saturated cost functions) create redundant constraints and thus can increase degeneracy, while duplicate columns (labels with identical minimum saturated costs under all abstractions) can lead to non-unique optima. Grouping abstractions and labels by identical effect in the LP eliminates duplicates without changing the objective value, reducing the number of optimal bases and typically improving reuse via more stable sensitivity ranges.

Pommerening et al. (2013) proposed to group duplicate labels together. We argue that the same is possible for abstractions, identify a stronger label grouping and argue that even stronger redundancy reduction methods are possible.

For abstractions the two important quantities for the SPhO LP are the minimum saturated cost function and the heuristic estimate for the current state. Since the heuristic estimate changes for every state we treat it as an unknown variable and only look at the minimum saturated costs. All abstractions with the same minimum saturated costs will produce the same constraint, but with possibly different bounds. We combine all equivalent constraints and argue that maximizing over the abstractions always keeps the tightest bound.

Definition 5.1 (Abstraction-Grouped SPhO LP)
Given a transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$, a cost function *cost* for $\mathcal{T}$, a tuple of abstraction heuristics $\mathcal{H}$ for $\mathcal{T}$ and the equivalence relation:

$$h \sim_{\mathcal{H}} h' \leftrightarrow \forall \ell \in L_{rel} : \mathit{mscf}_h(\ell) = \mathit{mscf}_{h'}(\ell)$$

the abstraction-grouped *SPhO LP* is:

$$\begin{aligned} \min_{Y} \sum_{\ell \in L_{rel}} \mathit{cost}(\ell) \cdot Y_\ell \qquad & \text{subject to} \\ \sum_{\ell \in L_{rel}} \mathit{mscf}_{[h]}(\ell) \cdot Y_\ell \geq \max_{h' \in [h]} h'(s) \qquad & \text{for all } [h] \in \mathcal{H}/\sim_{\mathcal{H}} \\ Y_\ell \geq 0 \qquad & \text{for all } \ell \in L_{rel} \end{aligned}$$

where $[h]$ is the equivalence class of h under the equivalence relation $\sim_{\mathcal{H}}$: $[h] = \{h' \in \mathcal{H} : h \sim_{\mathcal{H}} h'\}$ and $\mathit{mscf}_{[h]} := \mathit{mscf}_h(\ell)$ for any $h \in [h]$.

We will now show that this grouping does not change the heuristic value by showing that the SPhO LP and the abstraction-grouped SPhO LP encode the same heuristic. The proof is straightforward since we only exclude redundant constraints, so the feasible region of the LP stays the same.

Theorem 5.2 *Let $h^{\mathit{SPhO}}_{\text{h-grouped}}$ be the heuristic that solves the abstraction-grouped SPhO LP for every state. Then $h^{\mathit{SPhO}} = h^{\mathit{SPhO}}_{\text{h-grouped}}$.*

Proof. Since both LPs have the same objective functions, it suffices to show that their feasible regions are equivalent. Let F be the feasi-

ble region of the original SPhO LP, and G be the feasible region of the abstraction-grouped SPhO LP. First, we show $F \subseteq G$. Let Y be a feasible assignment to the original SPhO LP. Consider any equivalence class $[h] \in \mathcal{H}/\sim_{\mathcal{H}}$. For every $h' \in [h]$, feasibility of Y in the original LP implies:

$$\sum_{\ell \in L_{rel}} \mathit{mscf}_{h'}(\ell) \cdot Y_\ell \geq h'(s).$$

By definition of $\sim_{\mathcal{H}}$, we have $\mathit{mscf}_{h'}(\ell) = \mathit{mscf}_{h}(\ell)$ for all $h, h' \in [h]$, so all inequalities above share the same left-hand side.

$$\sum_{\ell \in L_{rel}} \mathit{mscf}_{[h]}(\ell) \cdot Y_\ell \geq h'(s) \text{ for all } h' \in [h]$$

Therefore the h-grouped constraint

$$\sum_{\ell \in L_{rel}} \mathit{mscf}_{[h]}(\ell) \cdot Y_\ell \geq \max_{h' \in [h]} h'(s)$$

is fulfilled for any choice of $[h]$ and Y is feasible in the abstraction-grouped LP. Second, we show $G \subseteq F$. Let Y' be feasible in the abstraction-grouped LP, then any heuristic $h' \in \mathcal{H}$ with equivalence class $[h'] \in \mathcal{H}/\sim_{\mathcal{H}}$ fulfills

$$\sum_{\ell \in L_{rel}} \mathit{mscf}_{[h']}(\ell) \cdot Y'_\ell \geq \max_{h \in [h']} h(s).$$

Since $\max_{h \in [h']} h(s) \geq h'(s)$, it follows that Y' is feasible in the original LP. In combination, it follows that $F = G$. □

Abstraction-grouping on the dual SPhO LP (Definition 3.3), gives a label-grouping strategy for the primal LP that is more general than what was proposed by Pommerening et al. (2013). Grouping abstractions the same way in this dual LP removes columns in the primal that have the same body and groups them based on their coefficients.

Definition 5.2 (Label-Grouped SPhO LP)
Given a transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$, a cost function *cost* for $\mathcal{T}$, a tuple of abstraction heuristics $\mathcal{H}$ for $\mathcal{T}$ and the equivalence relation:

$$\ell \sim_L \ell' \Leftrightarrow \forall h \in \mathcal{H} : \mathit{mscf}_{h}(\ell) = \mathit{mscf}_{h}(\ell')$$

the label-grouped *SPhO LP* is:

$$\begin{aligned} \min_{Y} \sum_{[\ell]\in L_{rel}/\sim_L} \min_{\ell'\in[\ell]} \mathit{cost}(\ell') \cdot Y_\ell \qquad & \text{subject to} \\ \sum_{[\ell]\in L_{rel}/\sim_L} \mathit{mscf}_h(\ell) \cdot Y_\ell \geq h(s) \qquad & \text{for all } h \in \mathcal{H} \\ Y_\ell \geq 0 \qquad & \text{for all } [\ell] \in L_{rel}/\sim_L \end{aligned}$$

As before this grouping does not change the heuristic.

Theorem 5.3 *Let $h^{SPhO}_{\ell\text{-grouped}}$ be the heuristic that solves the label-grouped SPhO LP for every state. Then $h^{SPhO} = h^{SPhO}_{\ell\text{-grouped}}$.*

Proof. The dual label-grouped SPhO LP has the form:

$$\begin{aligned} \max_{w} \sum_{h\in\mathcal{H}} h(s) \cdot w_h \qquad & \text{subject to} \\ \sum_{h\in\mathcal{H}} \mathit{mscf}_h(\ell) \cdot w_h \leq \min_{\ell'\in[\ell]} \mathit{cost}(\ell') \qquad & \text{for all } [\ell] \in L_{rel}/\sim_L \\ w_h \geq 0 \qquad & \text{for all } h \in \mathcal{H} \end{aligned}$$

The proof on this dual LP follows directly from the proof for Theorem 5.2, since the flip of the inequality is accounted for by the change to minimization. □

Since both LP reductions work on the different dimensions of the LP matrix it is possible to combine them without losing the equivalence guarantee.

Definition 5.3 (Grouped SPhO LP)
Given a transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$, a cost function *cost* for $\mathcal{T}$, a tuple of abstraction heuristics $\mathcal{H}$ for $\mathcal{T}$ and the equivalence relations:

$$h \sim_{\mathcal{H}} h' \leftrightarrow \forall \ell \in L_{rel} : \mathit{mscf}_h(\ell) = \mathit{mscf}_{h'}(\ell)$$
$$\ell \sim_L \ell' \Leftrightarrow \forall h \in \mathcal{H} : \mathit{mscf}_h(\ell) = \mathit{mscf}_h(\ell')$$

the grouped *SPhO LP* is:

$$\begin{aligned} \min_{Y} \sum_{[\ell]\in L_{rel}/\sim_L} \min_{\ell'\in[\ell]} \mathit{cost}(\ell') \cdot Y_\ell \qquad & \text{subject to} \\ \sum_{[\ell]\in L_{rel}/\sim_L} \mathit{mscf}_h(\ell) \cdot Y_\ell \geq \max_{h'\in[h]} h'(s) \qquad & \text{for all } [h] \in \mathcal{H}/\sim_{\mathcal{H}} \\ Y_\ell \geq 0 \qquad & \text{for all } [\ell] \in L_{rel}/\sim_L \end{aligned}$$

$h^{\text{SPhO}}_{\text{grouped}}$ is the heuristic that solves the grouped SPhO LP for every state.

These reduction rules eliminate basic redundancies from the SPhO LP. They could be improved by identifying duplication with respect to a scaling factor. In general, it is possible to remove all redundant constraints from a non-parametric LP by solving a linear program for each constraint (Paulraj and Sumathi 2010). This means it would be possible to remove all redundancies in the dimension of the labels, but it would require solving thousands of LPs in the preprocessing phase of the heuristic. With the abstractions this is not possible, since their constraints are parametric with respect to the evaluated state. This means that redundant constraints are state dependent and removing all of them would only work for each state individually. As a consequence this would first require solving much more LPs per state, but more importantly mean that the LP for every state would be different, so sensitivity analysis would no longer work. We also did not investigate eliminating all redundant labels against the described fast redundancy check since we do not expect it to make a big difference.

5.3 Grouped SPhO LP Experiments

We evaluated the label and abstraction grouped SPhO heuristic with the same setup as in Section 4.6. Grouping has, like the cover rules, no effect on the number of evaluated or expanded states.

First we show that grouping has a clear positive effect on the number of degenerate and non-unique solutions encountered when searching with SPhO in Figure 5.3. When grouping labels and abstractions for simple pattern databases of size 1, non-unique solutions only appear around 9% of the time in contrast to over 80% before. The effect on large patterns is much lower, but unique solutions appear around 18% of the time in comparison to not at all before.

Grouping has a very large effect on the number of variables and constraints in the produced LP. We show the difference in both in Figure 5.4, which is often multiple orders of magnitude.

Figure 5.5 shows that grouping also has a beneficial effect on the number of solved LPs. The effect gets stronger for better cover rules, but also more mixed. Grouping moderately increases the coverage of all cover rules as well, as shown in Table 5.1. Most notable is the effect for $h^{\text{SPhO}}_{\text{exact}}$ which solves 45 new tasks and becomes the second-strongest

cover rule. This is interesting since it goes against the result from without grouping, where the more expensive cover rules did not pay off. With grouping $h^{\text{SPhO}}_{\text{exact}}$ nearly offsets its expensive matrix multiplication cover test through its savings in LP computations in comparison to $h^{\text{SPhO}}_{\text{eqdist}}$.

Finally, we show in Figure 5.6 that the combination of better LP reusability and smaller LPs leads to a runtime improvement from which again $h^{\text{SPhO}}_{\text{exact}}$ profits the most.

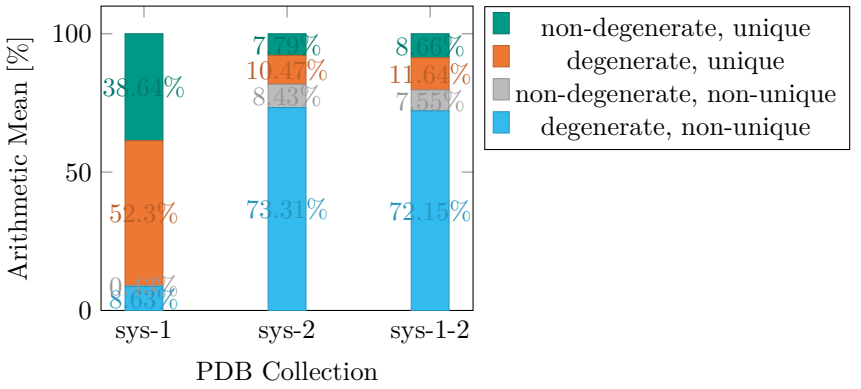


Figure 5.3: Average percentage of degenerate and non-unique solutions for the *grouped* SPhO heuristic over different PDB sets. See Figure 5.2 for an explanation of the PDB collections.

	h^{SPhO}	$h^{\text{SPhO}}_{\text{eqdist}}$	$h^{\text{SPhO}}_{\text{range}}$	$h^{\text{SPhO}}_{100\%}$	$h^{\text{SPhO}}_{\text{exact}}$
	865	975	881	901	892
grouped	898	**987**	890	914	937

Table 5.1: Comparison of the number of solved tasks (out of 1884) of the previous and the grouped configurations of saturated post-hoc optimization

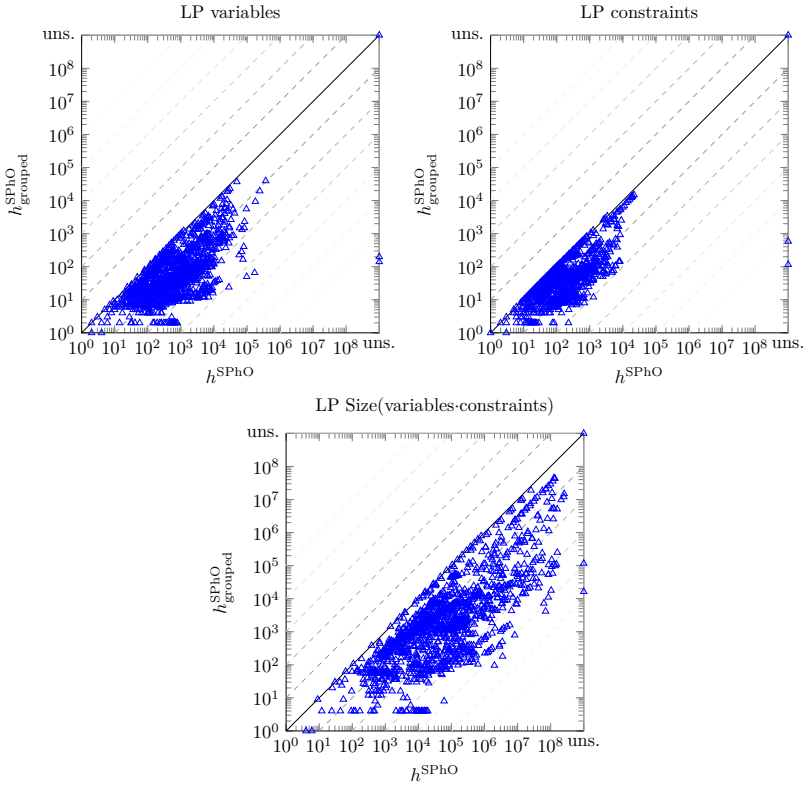


Figure 5.4: Per-task comparison of the grouped and ungrouped SPhO LP size. Tasks that are not solved within the resource limits appear at "uns." on the axes.

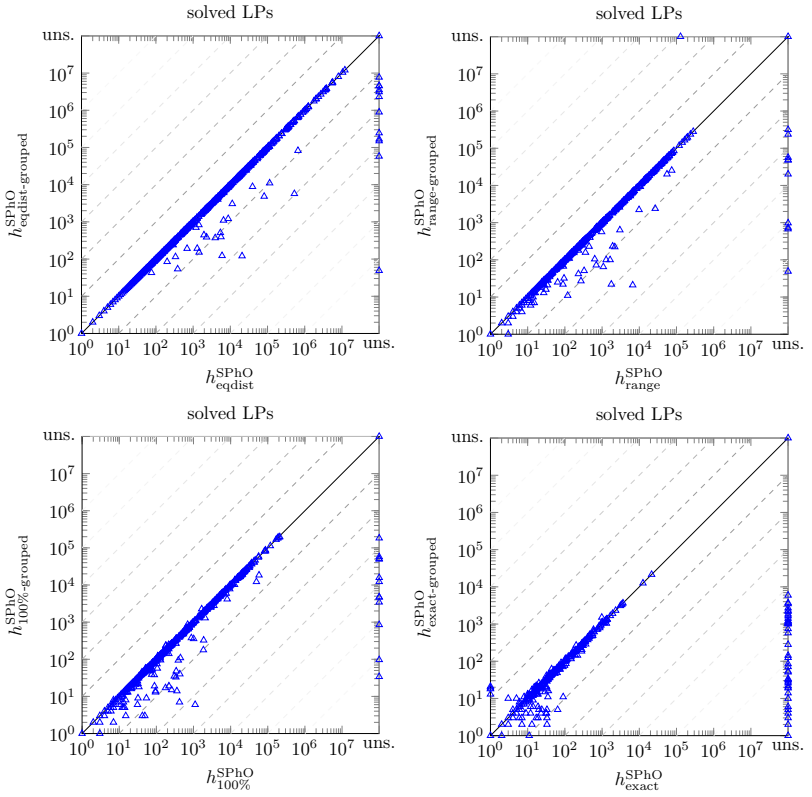


Figure 5.5: Comparison of the solved LPs between the grouped and non-grouped versions of SPhO. Tasks that are not solved within the resource limits appear at “uns.” on the axes.

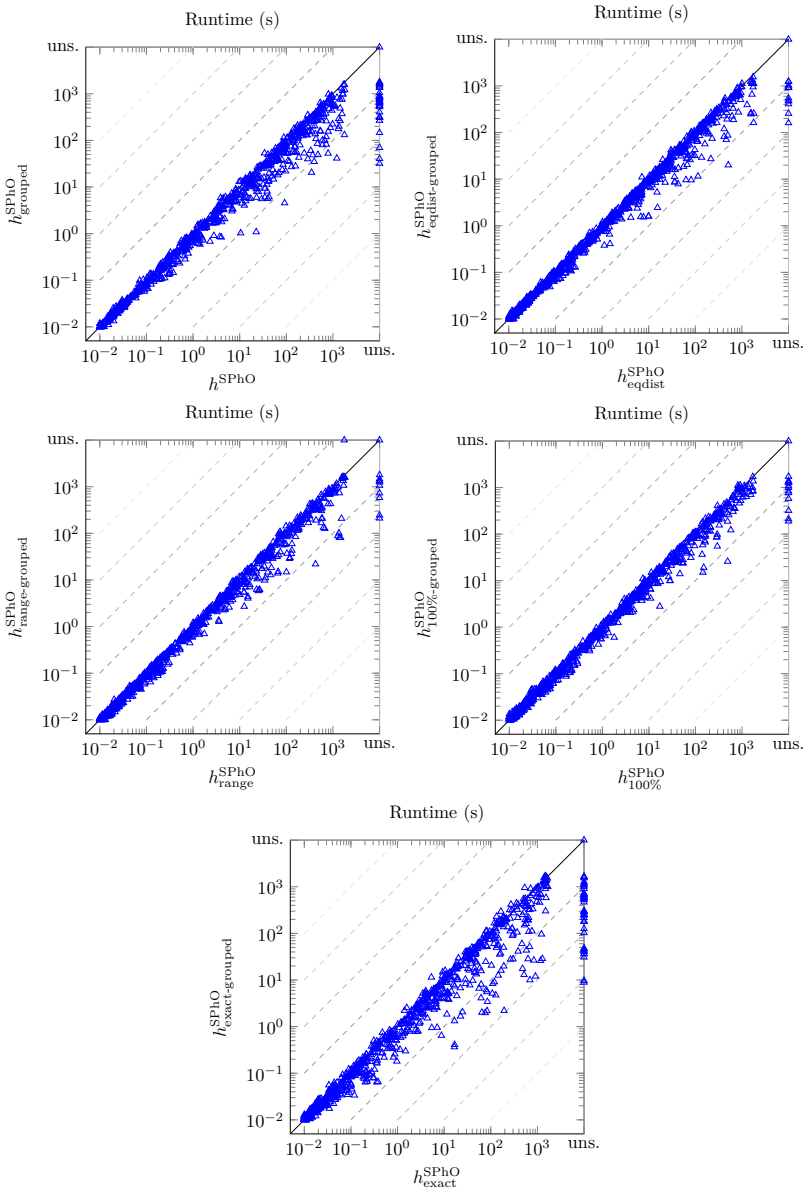


Figure 5.6: Runtime comparison between the grouped and non-grouped versions of SPhO. Tasks that are not solved within the resource limits appear at "uns." on the axes.

5.4 Versatile Cost Partitions

As discussed previously in Chapter 5, there can be multiple SPhO cost partitions $\mathcal{C}_1, \dots, \mathcal{C}_j$ that yield the same heuristic value $h^{\mathcal{C}_1}(s) = \dots = h^{\mathcal{C}_j}(s)$ for a given state s. Although they are all equally optimized for the state s, their heuristic values for other states s' might differ. Therefore, instead of accepting the arbitrary cost partition that the LP solver returns, it is beneficial to choose one according to a tiebreaking mechanism. This tiebreaking should increase the reusability of the obtained cost partition by preferring cost partitions that will lead to higher estimates on other states s'. We call such a cost partition a *versatile* cost partition. To formalize a beneficial tiebreaking, we need to define when a cost partition is more preferable than others. The only case when a clear preference for all possible unseen states exists is when a cost partition generalizes another.

Definition 5.4 (Cost Partition Generalization)
A cost partition $\mathcal{C}$ *generalizes* another cost partition $\mathcal{C}'$ if for all states $s \in S$, it holds that $h^{\mathcal{C}}(s) \geq h^{\mathcal{C}'}(s)$. The generalization is *strict* if there exists at least one state s such that $h^{\mathcal{C}}(s) > h^{\mathcal{C}'}(s)$.

Generalization becomes interesting when cost partitions optimized for state s are reused for another state s'. The goal is to compute a cost partition that will also produce good results for other states. To formalize this, we define that a cost partition is more versatile if it has better generalization than some other cost partition optimized for the same state.

Definition 5.5 (Versatile Cost Partition)
Let $\mathcal{C}_s = \{\mathcal{C}_1, \dots, \mathcal{C}_j\}$ be a set of j cost partitions that are optimal for a given state $s \in S$. If there exists a cost partition $\mathcal{C} \in \mathcal{C}_s$ such that $\mathcal{C}' \in \mathcal{C}_s$ strictly generalizes $\mathcal{C}$, then $\mathcal{C}'$ is more *versatile* for state s than $\mathcal{C}$.

We use the term versatile instead of *general* to restrict ourselves to cost partitions that are optimal for at least the current state s and to avoid confusion with general cost partitioning (Pommerening et al. 2015).

LP solvers usually do not return the set of all optimal bases, or solution coefficients, since features like solution pools are only imple-

mented for mixed integer programs. Although it is possible to obtain a set of alternative solutions with multiple LP solver calls, finding a representative set is still active research (Kakkad et al. 2024). To find more versatile cost partitions, we therefore opt for detecting cost partitions $\mathcal{C}$ that can be transformed into a more versatile version $\mathcal{C}'$.

Now we formalize and prove a practical definition of when a SPhO cost partition $\mathcal{C}'$ is more versatile than another cost partition $\mathcal{C}$.

Theorem 5.4 *Let $\mathcal{C}(h_i) = w_{h_i} \cdot \mathit{mscf}_i, \mathcal{C}'(h_i) = w'_{h_i} \cdot \mathit{mscf}_i$ be two SPhO cost partitions over the same abstraction heuristics $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ for a transition system $\mathcal{T}$. Then $\mathcal{C}'$ generalizes $\mathcal{C}$ if the cost function multipliers of $\mathcal{C}'$ are at least as high as the multipliers of $\mathcal{C}$, so $w'_h \geq w_h$ for all $h \in \mathcal{H}$. The generalization is strict if there exists one multiplier that is strictly greater for a heuristic that is non-zero for some states, so there exists a heuristic $h_j \in \mathcal{H}$ such that $w'_{h_j} > w_{h_j}$ and there exists a state $s \in S$ such that $h_j(s, \mathit{mscf}_{h_j}) \neq 0$.*

This condition is not necessary for generalization but only sufficient.

Proof. Since abstraction heuristics compute optimal goal distances, h^α is non-negative. Saturating an abstraction heuristic preserves this property, so $h^\alpha(s, \mathit{cost}) \geq 0$ for all states $s \in S$ when $\mathit{cost} = w \cdot \mathit{mscf}$ and $w \geq 0$.

Let $\mathcal{C}(h_i) = w_{h_i} \cdot \mathit{mscf}_i$ and $\mathcal{C}'(h_i) = w'_{h_i} \cdot \mathit{mscf}_i$ be two SPhO cost partitions over $\mathcal{H} = \langle h_1, \dots, h_n \rangle$. For all states $s \in S$:

$$\begin{aligned} h^{\mathcal{C}'}(s) - h^{\mathcal{C}}(s) &= \sum_{i=1}^{n} w'_{h_i} h_i(s, \mathit{mscf}_i) - \sum_{i=1}^{n} w_{h_i} h_i(s, \mathit{mscf}_i) \\ &= \sum_{i=1}^{n} (w'_{h_i} - w_{h_i}) h_i(s, \mathit{mscf}_i). \end{aligned}$$

Since $w'_{h_i} \geq w_{h_i}$ and $h_i(s, \mathit{mscf}_i) \geq 0$ for all $1 \leq i \leq n$, each term $(w'_{h_i} - w_{h_i}) h_i(s, \mathit{mscf}_i) \geq 0$, hence $h^{\mathcal{C}'}(s) \geq h^{\mathcal{C}}(s)$.

For strict generalization there needs to exist one $w'_{h_j} > w_{h_j}$, where $1 \leq j \leq n$ with a state $s' \in S$ such that $h_j(s, \mathit{mscf}_{h_j}) \neq 0$. Then for all such states $s' \in S$ the term $(w'_{h_j} - w_{h_j}) h_j(s', \mathit{mscf}_j) > 0$. If we use the result from generalization for all other states and cost multipliers, it follows that $h^{\mathcal{C}'}(s') > h^{\mathcal{C}}(s')$ for all states $s' \in S$ where $h_j(s, \mathit{mscf}_{h_j}) \neq 0$. □

The issue with this result is that the SPhO LP already optimizes each w_{h_i} to be as high as possible. In practice, this means that any set of SPhO cost partitions that are optimal for a given state s will be incomparable with respect to Theorem 5.4. Thus, for any two cost partitions $\mathcal{C}(h_i) = w_{h_i} \cdot \mathit{mscf}_i$ and $\mathcal{C}'(h_i) = w'_{h_i} \cdot \mathit{mscf}_i$, there will be indices i, j such that $w_{h_i} > w'_{h_i}$ and $w_{h_j} < w'_{h_j}$, unless there is a heuristic $h_i(s, \mathit{mscf}_i) = 0$. If an abstraction heuristic returns a zero estimate $h(s) = 0$ for a state $s \in S$, its LP variable w_h does not affect the objective value. In this case, any feasible weight could be chosen by the LP solver, making this a redundant dimension in the optimization problem that causes non-uniqueness in the linear program. We exploit this slack to construct a more versatile cost partition $\mathcal{C}'$ by increasing such w_h as much as possible while keeping all constraints tight or feasible. This approach guarantees that $h^{\mathcal{C}'}(s') \geq h^{\mathcal{C}}(s')$ for all states s' where at least one heuristic h with increased w_h has $h(s') > 0$, and $h^{\mathcal{C}}(s') = h^{\mathcal{C}'}(s')$ for all other states. Therefore, the possibility of reuse with a cover rule is higher when using $\mathcal{C}'$ instead of $\mathcal{C}$.

If there exist multiple h_i in $\mathcal{H}$ that are zero for a given state $s \in S$, there can exist multiple different more versatile cost partitions that are incomparable with respect to Theorem 5.4. This means that it is unclear which cost partition is preferable for future states. We will therefore randomly select an order or let the LP solver select one when implementing tiebreaking strategies.

Note that tiebreaking for more versatile cost partitions is only beneficial for lazy SPhO with a cover rule that can generalize to states with different abstract goal distances. $h^{\text{SPhO}}_{\text{eqdist}}$ will never benefit from tiebreaking since it is too restrictive. For the other cover rules, we can guarantee that a more versatile cost partition will cover more states. As discussed in Section 4.5, locally better decisions do not guarantee a globally better solution here. Therefore, tiebreaking for a more versatile cost partition does not guarantee fewer solved LPs.

5.4.1 Tiebreaking with the Increase Weights Algorithm

We established that increasing the SPhO weight for a heuristic with a zero goal distance results in a more versatile cost partition. To compute these types of cost partitions we define a greedy procedure for increasing weights for zero-valued heuristics in Algorithm 2. It needs

Algorithm 2 Greedily increases the heuristic weights of a SPhO LP solution *sol* to a more versatile cost partition.

```
1: procedure IncreaseWeights(𝓗, sol, remain, s)
2:   for h ∈ 𝓗 with h(s) = 0, in random order do
3:     Δw = min_{ℓ∈L} { remain(ℓ) / mscf_h(ℓ) | mscf_h(ℓ) > 0 }
4:     w_h += Δw
5:     for ℓ ∈ L do
6:       remain(ℓ) −= mscf_h(ℓ) · Δw
```

to know the remaining costs for all labels $\ell \in L$, which we compute as $\mathit{remain}(\ell) = \mathit{cost}(\ell) - \sum_{h \in \mathcal{H}} \mathit{mscf}_h(\ell) \cdot w_h$. Then the procedure iterates over the heuristics h with $h(s) = 0$ in a random order and increases the weight w_h only if the remaining label costs allow for an increase. The algorithm increases the weight only as far as the remaining costs allow, since cost functions with the increased weight still form a cost partition:

$$\Delta w = \min_{\ell \in L} \left\{ \frac{\mathit{remain}(\ell)}{\mathit{mscf}_h(\ell)} \,\middle|\, \mathit{mscf}_h(\ell) > 0 \right\}$$

$$\Delta w \cdot \mathit{mscf}_h(\ell) \leq \mathit{remain}(\ell) \text{ for all } \ell \in L$$

$$\Delta w \cdot \mathit{mscf}_h(\ell) \leq \mathit{cost}(\ell) - \sum_{h' \in \mathcal{H}} \mathit{mscf}_{h'}(\ell) \cdot w_{h'} \text{ for all } \ell \in L$$

$$\Delta w \cdot \mathit{mscf}_h(\ell) + \sum_{h' \in \mathcal{H}} \mathit{mscf}_{h'}(\ell) \cdot w_{h'} \leq \mathit{cost}(\ell) \text{ for all } \ell \in L$$

This shows that the algorithm produces a valid cost partition in every iteration. Each iteration of the for loop in the algorithm is equivalent to a pivot step in the simplex algorithm and is guaranteed to stay optimal since we ensure two conditions. We only change coefficients of variables that will be multiplied with zero and make sure that the result is still a valid cost partition, ensuring feasibility.

If the IncreaseWeights procedure found a higher weight for at least one heuristic, we want to feed this solution back into the LP solver, to obtain sensitivity analysis for this new solution. We do so by solving the same SPhO LP again, but warm-starting it with the basis found by the IncreaseWeights algorithm. Re-solving the LP is cheap as long as the LP solver detects that the provided solution is already optimal. If negative costs occur in the minimum saturated cost functions and the

provided solution is degenerate or non-unique, the simplex algorithm may fail to detect its optimality, reject it, and search its own solution again. This happens since the optimality condition (Equation 2.3) is not necessary for all optimal bases of degenerate linear programs. In our implementation of SPhO with IncreaseWeights, we therefore skip the procedure when we detect negative saturated costs. This occurs for roughly 30% of our benchmark tasks.

5.4.2 Tiebreaking with Small Weights

Our notion of more versatile cost partitions is equivalent to not ignoring any heuristic in the optimization process even if it has a heuristic value of zero for the current state. Instead of manually pivoting to a more versatile solution, it is possible to encode this preference in the SPhO LP itself.

Definition 5.6 (Epsilon SPhO LP)
Given a transition system $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$, a cost function *cost* for $\mathcal{T}$, and a tuple of abstraction heuristics $\mathcal{H}$ for $\mathcal{T}$, the heuristic value $h^{\text{SPhO}}_{eps}(s)$ for a state s is the *objective value* of the ***SPhO**$_{eps}$ LP*:

$$\min_x \sum_{\ell \in L_{rel}} \mathit{cost}(\ell) \cdot Y_\ell \qquad \text{subject to}$$

$$\sum_{\ell \in L_{rel}} \mathit{mscf}_h(\ell) \cdot Y_\ell \geq \begin{cases} h(s) \text{ for all } h \in \mathcal{H} & \text{if } h(s) \neq 0 \\ \epsilon & \text{otherwise} \end{cases}$$

$$Y_\ell \geq 0 \text{ for all } \ell \in L_{rel}$$

When ϵ is chosen sufficiently small, $h^{\text{SPhO}}_{eps}(s)$ is equal to $h^{\text{SPhO}}(s)$ but prioritizes more versatile cost partitions. From simplex pivot operations it follows that ϵ has to be at least as small as $\frac{1}{\max_{\ell \in L, h \in \mathcal{H}} \mathit{mscf}_h(\ell)}$. In our experiments it was sufficient to select $\epsilon = 1 \times 10^{-8}$. There are two issues with this approach. First, ϵ has to be manually selected and second, if ϵ becomes very small the LP solver can run into numerical problems.

5.5 Versatile Cost Partition Experiments

We now analyze the effect of tiebreaking on h^{SPhO} using the same experimental setup as in Section 4.6. As stated before, the presented tiebreaking rules have no effect on h^{SPhO} and $h^{\text{SPhO}}_{\text{eqdist}}$, so we will only present results for the other three cover rules.

First, we compare the effect on the number of solved LPs. Figure 5.7 shows that, except for the exact cover rule, tiebreaking is not making much of a difference. When comparing the tiebreaking mechanisms, we see that the increasing weights tiebreaking has less of an effect than the ϵ tiebreaking, but gives a positive improvement more consistently. The ϵ tiebreaking has, especially for the 100% cover rule, a surprisingly high number of tasks where the number of LP solver calls increases slightly. This can be attributed to the non-linear effect of better cover rules that was discussed in Section 4.5 and applies directly to the more versatile cover rules as well, but might also stem from the increased numerical instability. That only the exact cover rule benefits from the tiebreaking indicates that the expansion in sensitivity ranges is subtle and can therefore only be taken advantage of by a strong cover rule. The number of encountered degenerate and non-unique bases

	h^{SPhO}	$h^{\text{SPhO}}_{\text{eqdist}}$	$h^{\text{SPhO}}_{\text{range}}$	$h^{\text{SPhO}}_{100\%}$	$h^{\text{SPhO}}_{\text{exact}}$
	865	975	881	901	892
grouped	898	**987**	890	914	937
grouped-IW	—	—	890	914	939
grouped-ϵ	—	—	890	909	937

Table 5.2: Number of solved tasks (out of 1884) of the previous and the new tiebreaking lazy variants.

is nearly unchanged between the baseline (without tiebreaking) and the two tiebreaking approaches. This was expected with the increasing weights algorithm since more versatile solutions only exist for non-unique bases and do not find a unique alternative, and the tiebreaking also does not affect the degeneracy of the found solutions. We expected a slight decrease in degeneracy for the ϵ tiebreaking because the approach is similar to the perturbation method (Sierksma and Zwols 2015), a common way of countering degeneracy. However, our experiments suggest that this does not take effect here, as there are too many degenerate non-zero constraints that we do not perturb.

Our tiebreaking approaches influence coverage very little as shown in Table 5.2. As with the solved LPs, exact sensitivity analysis benefits from the tiebreaking and the 100% rule is harmed by it.

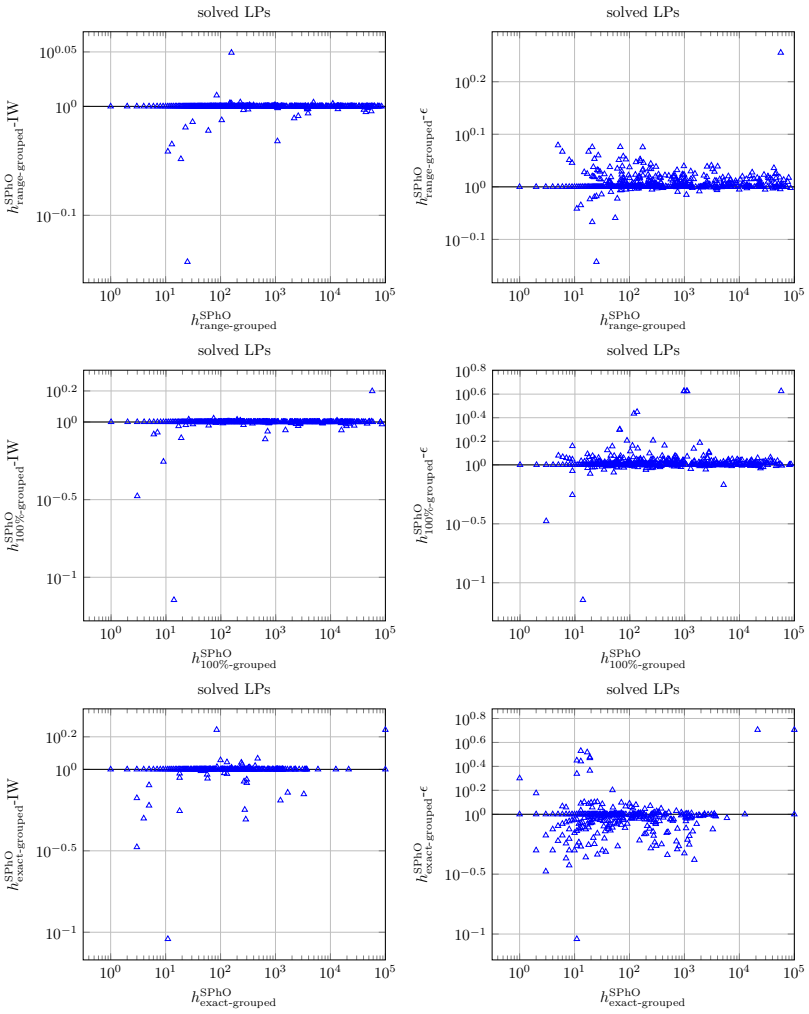


Figure 5.7: Comparison of the solved LPs between the increase weights, ϵ tiebreaking, and no tiebreaking for the grouped SPhO LP. Each point (x, y) represents a comparison of two algorithms on a single planning task, where the algorithm on the x-axis results in a value of x and the algorithm on the y-axis computes $x \cdot y$. A point at $y = 2$ therefore means that the algorithm on the y-axis has double the value as the algorithm on the x-axis for this task. So for points below $y = 1$ the algorithm on the y-axis performs better here.

6 Discussion

Even though our application of sensitivity analysis to the SPhO LP is new, sensitivity analysis and the concept of reusing solutions to minimize effort are not. In linear programming, it is somewhat common to repeatedly solve similar LPs. Modern LP solvers such as CPLEX therefore use a number of techniques to reuse information about the previously solved LP and its solution, sometimes referred to as *advanced* or *warm starts*. Thus, implementations of LP-based heuristics such as eager h^{SPhO} use such warm starts and information about previously solved LPs by default. This is also a big reason why the huge reductions in the number of solved LPs are not reflected in speedups of the same magnitude. The LP solver is already trying to help as much as possible.

However, there is an important difference between using LP solvers with warm starts and the theory presented in this thesis. An LP solver with warm starts stores only the most recently solved LP and information about its solution. In contrast, our cover rules store all previously encountered solutions, making them much more useful for our approach.

The same can be said about reducing redundancies. Any modern LP solver uses presolving procedures that, among other things, reduce redundancies in the linear program (E. Andersen and K. Andersen

1995). Our approach here differs from them because we reduce redundancies in a parametric linear program rather than a fixed one. We only reduce redundancies that hold for all possible states in the planning problem, not just for a single one. We could therefore only rely on presolving procedures in the dimension of the labels, but restricting a presolving process to a single dimension is, to our knowledge, not possible with CPLEX. There might exist other LP solvers or programs for presolving that are not as well known in the planning community that we could have used to reduce the redundancies. This would be an interesting point for future work.

7 Offline Saturated Post-Hoc Optimization

Our previous results show that for many planning problems the number of unique SPhO cost partitions that suffice to guide A* search as best as possible is much smaller than the number of states in the problem. Similar results have been shown for potential heuristics by Seipp et al. (2015). A few of them can already perfectly cover a large set of states. A similar result was shown for another non-optimal cost partitioning strategy, saturated cost partitioning (Seipp et al. 2020). For this heuristic, the authors showed that when maximizing over a pool of 1000 sampled cost partitions, the number of partitions that actually contribute is small if they are not actively diversified. In the second part of this thesis we will show that the saturated cost partitioning heuristic with diversified orders is very close to the perfect saturated cost partitioning heuristic when only considering 1000 sampled states.

So, for both saturated cost partitioning and potential heuristics, similar results show that a small, representative set of cost partitions is often enough to obtain a heuristic that is close to the best possible one. For both heuristics, there exist very successful algorithms that precompute a set of cost partitions before starting the A* search. Both of these build upon the work of Karpas et al. (2011), which first suggested the strategy of computing an informative sample of cost partitions. Their

work did this for optimal cost partitioning and used a sampling method from Haslum et al. (2007), a method that was also used by the later works.

Since these previous works, together with our results, suggest that a similar sampling-based approach will work for SPhO as well, we introduce it as a practical approximation of h^{SPhO}, in contrast to lazy SPhO, which computes the perfect SPhO heuristic. We construct $h^{\text{SPhO}}_{\text{offline}}$ using the diversification procedure described by Seipp et al. (2020). We allocate 200 seconds of preprocessing time to offline SPhO. During this time, we initialize two sets: an empty set C of cost partitions and a set of states $S_{sample} = \{s_0\}$ containing the initial state. Next, we sample 999 additional states using the method of Haslum et al. (2007). The resulting set of 1000 states S_{sample} is then used to filter for relevant cost partitions. Until the preprocessing time is up, the algorithm samples a state and solves the SPhO LP for it. The resulting cost partition is then added to C *only if* it improves the heuristic for at least one state in S_{sample} compared to all partitions already in C. During the search, $h^{\text{SPhO}}_{\text{offline}}$ is computed by maximizing over all cost partitions in C.

7.1 Experimental Evaluation

We compare the offline SPhO algorithm with SPhO and the lazy variants we introduced previously. We will call the standard SPhO heuristic *eager* SPhO in this part to distinguish it clearly from offline SPhO. In contrast to lazy SPhO, offline SPhO approximates the SPhO heuristic, so the number of expansions is different. Figure 7.1 shows that there is little difference in expansions between offline SPhO and eager SPhO. While $h^{\text{SPhO}}_{\text{offline}}$ does not retain full heuristic accuracy, the ensemble of 1000 states seems to capture a meaningful set of cost partitions, since the expansions until the last f-layer show very few tasks where $h^{\text{SPhO}}_{\text{offline}}$ is at a disadvantage.

Since offline SPhO does not have much expansion overhead and does not compute any cost partitions during the search, it is the strongest algorithm in terms of coverage. Table 7.1 shows that it improves over all discussed lazy SPhO variants.

In contrast, we see in Figure 7.2 that the strong coverage of $h^{\text{SPhO}}_{\text{offline}}$ is not necessarily reflected in runtime, as it typically takes much longer to solve a problem than standard h^{SPhO}. The reason for this is that $h^{\text{SPhO}}_{\text{offline}}$

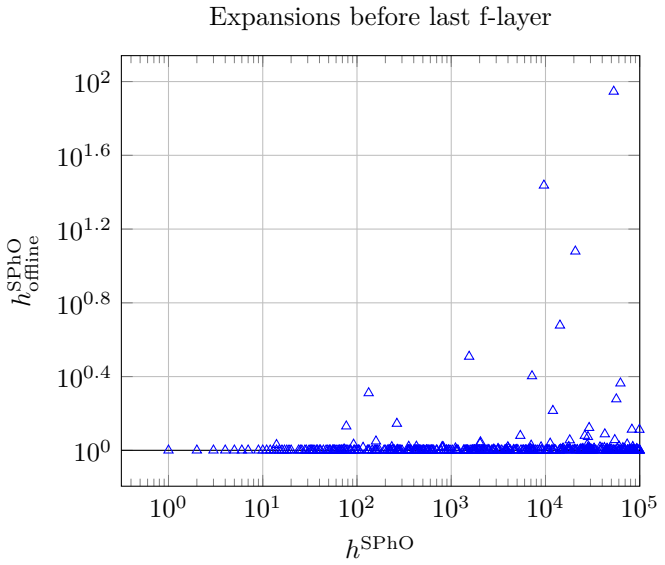


Figure 7.1: Comparison of the expansions until the last f-layer between $h^{\text{SPhO}}_{\text{offline}}$ and eager h^{SPhO} on a relative plot (see Figure 5.7 for an explanation). We exclude all states on the final f-layer (the layer with the goal state), since their expansion order depends on tiebreaking, which can be a source of non-determinism.

	h^{SPhO}	$h^{\text{SPhO}}_{\text{eqdist}}$	$h^{\text{SPhO}}_{\text{range}}$	$h^{\text{SPhO}}_{100\%}$	$h^{\text{SPhO}}_{\text{exact}}$	$h^{\text{SPhO}}_{\text{offline}}$
	865	975	881	901	892	**1009**
grouped	898	987	890	914	937	—
grouped-IW	—	—	890	914	939	—
grouped-ϵ	—	—	890	909	937	—

Table 7.1: Number of solved tasks (out of 1884) of the eager SPhO heuristic h^{SPhO} and our new lazy variants in comparison to offline SPhO.

uses a fixed precomputation time (200 seconds), after which it quickly solves most of the tasks that are also solved by h^{SPhO}.

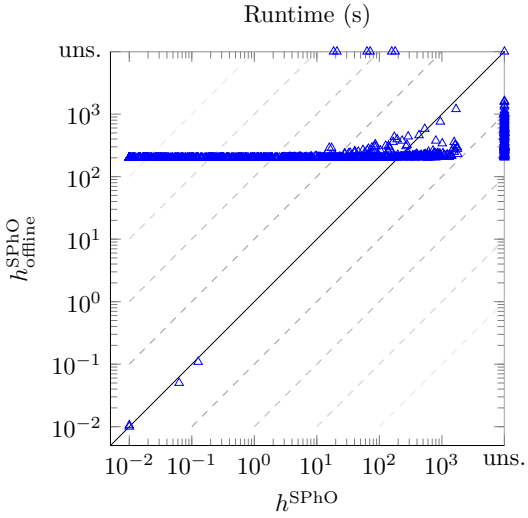


Figure 7.2: Runtime comparison between the eager SPhO and offline SPhO. Problems that are not solved within the resource limits appear on “uns.” axes.

PART
II

Perfect Saturated Cost Partitioning

8 Introduction

After studying the reusability of cost partitions for the SPhO heuristic, we now focus on another state-of-the-art cost partitioning strategy: saturated cost partitioning (SCP) (Seipp et al. 2020). In contrast to the SPhO heuristic, which solves a non-optimal cost partitioning problem optimally as a linear program, SCP is a greedy non-optimal cost partitioning strategy. Given a set of abstraction heuristics $\mathcal{H}$, it *saturates* each heuristic $h \in \mathcal{H}$ in a given order ω, preserving unused costs for subsequent heuristics. Formally, we define an SCP heuristic as follows.

Definition 8.1 (Saturated Cost Partitioning (Seipp et al. 2020))
Let $\mathcal{T}$ be a transition system, *cost* a cost function for $\mathcal{T}$, and $\mathcal{H}$ a set of abstraction heuristics for $\mathcal{T}$. Given a permutation $\omega = \langle h_1, \dots, h_n \rangle$, called an order over $\mathcal{H}$, the h^{SCP} heuristic $h^{\mathcal{C}^{\text{SCP}}_\omega}$ is the cost partitioning heuristic induced by the saturated cost partition $\mathcal{C}^{\text{SCP}}_\omega = \{h_i \mapsto \mathit{cost}_i\}$, where the cost functions cost_i are computed as:

$$
\begin{aligned}
\mathit{remain}_0 &= \mathit{cost} \\
\mathit{cost}_i &= \mathit{saturate}(h_i, \mathit{remain}_{i-1}) && \text{for all } 1 \leq i \leq n \\
\mathit{remain}_i &= \mathit{remain}_{i-1} - \mathit{cost}_i && \text{for all } 1 \leq i \leq n
\end{aligned}
$$

where *saturate* computes the minimum saturated cost function (Definition 3.1) for the abstraction under the given cost function.

In theory, the saturation does not need to be the minimum saturated cost function, and dropping this requirement allows the use of other heuristic classes for saturated cost partitioning. In this work, however, we rely on the properties of the unique minimum saturated cost function and therefore restrict the saturated cost partitioning algorithm to abstraction heuristics. Unlike the calculation of the minimum saturated cost function, infinite values in the computation of the remaining costs are handled with the rules of left addition. This means that $\infty - \infty = \infty$. We simplify the notation for the SCP heuristic as follows: $h^{\text{SCP}}_{\omega} := h^{\mathcal{C}^{\text{SCP}}_{\omega}}$. The saturated post-hoc optimization heuristic and the saturated cost partitioning heuristic are not related beyond both being non-optimal cost partitioning heuristics. In particular, Seipp et al. (2021) showed that they are incomparable. This means that they have different strengths and weaknesses, so there exist problems where either cost partitioning heuristic dominates the other.

8.1 The Factorial Barrier to Perfect SCP

The accuracy of a saturated cost partitioning heuristic over abstraction heuristics $\mathcal{H}$ strongly depends on the order ω in which the SCP algorithm considers the heuristics $h \in \mathcal{H}$ (Seipp, Keller, et al. 2017b). It is therefore beneficial to maximize over multiple SCP heuristics h^{SCP}_{ω} computed for different orders ω (Seipp et al. 2020). In principle, we can obtain the best possible, the *perfect SCP heuristic*, h^{SCP}_{*}, by maximizing over *all* orders ω of $\mathcal{H}$.

Definition 8.2 (Perfect SCP)
Given a tuple of heuristics $\mathcal{H}$, let $\Omega(\mathcal{H})$ be the set of all orders of $\mathcal{H}$. The perfect saturated cost partitioning heuristic over $\mathcal{H}$ is the heuristic that maximizes over all saturated cost partitioning orders: $h^{\text{SCP}}_{*} = \max_{\omega \in \Omega(\mathcal{H})} h^{\text{SCP}}_{\omega}$

However, computing h^{SCP}_{*} naively in this way for $n = |\mathcal{H}|$ heuristics requires enumerating all $n!$ orders and storing n lookup tables for each order. This factorial explosion makes the naive computation of h^{SCP}_{*} infeasible even for small values of n.

Consequently, previous work by Seipp, Keller, et al. (2017b) greedily approximated good orders for single states s by ordering those heuristics h first that have a high ratio of $h(s)$ to the sum of saturated costs that h wants to take away from other heuristics. Greedy orders can then also be optimized further with local search. To cover multiple states, Seipp et al. (2020) proposed a diversification procedure that iteratively generates new SCP heuristics for (optimized) greedy orders ω for a pool of sampled states. However, they only keep the heuristic if it increases the heuristic estimate for at least one sampled state.

Instead of approximating h^{SCP}_*, we present the first approach to compute h^{SCP}_* that is practically feasible for moderate numbers of abstractions. In Part I, we demonstrated how to compute a perfect cost partitioning heuristic more efficiently. In this part, we make it computable at all. While our algorithm considers all orders, it only does so *implicitly*, which significantly reduces the number of computed and stored lookup tables. The main reductions come from two advancements. First, we share the computations and lookup tables between orders that have the same prefix. Second, we approximate equivalence between orders and enumerate only those orders that can lead to different SCP heuristics. We begin by showing the benefits of expressing a saturated cost partitioning heuristic as a graph over its saturated component heuristics. Later, we will explore equivalence between orders in Section 10.1.

9 SCP Heuristics as Computational Graphs

To optimize SCP Heuristics and make the computation of h^{SCP}_* feasible, we first look at the way a collection of saturated cost partitioning heuristics can be efficiently represented. Previous work like Seipp, Keller, et al. (2017b) and Seipp (2017) maximizes over a set of orders, while storing each heuristic independently. We show that computing and storing each order individually in such a collection of SCP heuristics is wasteful. Every time orders share an order prefix, they will compute the same cost partitioning heuristic for all abstraction heuristics in the prefix. When computing all possible orders, this significantly reduces computational effort and memory consumption. To share the lookup tables and abstraction functions between multiple saturated cost partitioning heuristics, we represent them as computational graphs over max and sum operations. This structure of computational graphs was first proposed by Coles et al. (2008). They showed that interesting ensembles of additive and non-additive (disjunctive) heuristics and many existing admissible heuristics can be expressed in this flexible framework.

Definition 9.1 (Additive-Disjunctive Heuristic Graph)
An additive-disjunctive heuristic graph (ADHG) (Coles et al. 2008) is

a directed acyclic graph G with a single root node r and three types of nodes: *sum*, *max*, and *evaluator* nodes. Sum and *max* nodes each represent a mathematical operation over their children, while evaluator nodes each have an assigned heuristic h_n. Each node n in an ADHG G evaluates to a heuristic value $G(s, n)$ by evaluating its children $succ(n)$ or its assigned heuristic.

$$G(s,n) = \begin{cases} \max_{c \in succ(n)} G(s,c), & \text{if } n \text{ is a } \textit{max} \\ \sum_{c \in succ(n)} G(s,c), & \text{if } n \text{ is a } \textit{sum} \\ h_n(s) & \text{otherwise} \end{cases}$$

The ADHG heuristic value is the heuristic value of the root node r: $h^{\text{ADHG}}(s) := G(s, r)$.

Example 9.1
Let G be an ADHG for heuristics $\{h_1, h_2, h_3\}$ in Figure 9.1. The heuristic value of $G(s)$ can be calculated by evaluating all saturated cost partitions at the leaf nodes. For the first sum node, we obtain $sum_1(s) = h_1(mscf_1, s) + h_2(mscf_2, s) + h_3(mscf_3, s)$ for all $s \in S$, where $mscf_1 = saturate(h_1, cost)$, $mscf_2 = saturate(h_2, cost - mscf_1)$ and $mscf_3 = saturate(h_3, cost - mscf_1 - mscf_2)$. The overall ADHG heuristic value is then $h^{\text{ADHG}}(s) = \max\{sum_1(s), \dots, sum_6(s)\}$.

The heuristic computed by an ADHG is admissible if the graph is *safely additive*.

Definition 9.2 (Admissible ADHGs (Coles et al. 2008))
Let G be an ADHG for a transition system $\mathcal{T}$ with cost function *cost*. The heuristic $G(s)$ is *admissible* if the *maximum aggregate cost* $mac(\ell, n)$ of label ℓ at node n is less than or equal to the cost assigned by the cost function for all nodes in G.

$$mac(\ell, n) \leq cost(\ell) \text{ for all } n \in G.$$

And the maximum aggregate cost is recursively defined as follows:

$$mac(\ell,n) = \begin{cases} \max_{c \in succ(n)} mac(\ell,c), & \text{if } n \text{ is a } \textit{max} \\ \sum_{c \in succ(n)} mac(\ell,c), & \text{if } n \text{ is a } \textit{sum} \\ cost_{h_n}(\ell) & \text{otherwise} \end{cases}$$

and $cost_{h_n}(\ell)$ is the cost of ℓ in the evaluator node heuristic h_n.

This is always given for ADHGs derived from SCP heuristics because the maximum aggregate cost checks the cost partitioning condition Equation 2.5. For a proof idea, see Theorem 1 and 2 and their proofs in Coles et al. (2008).

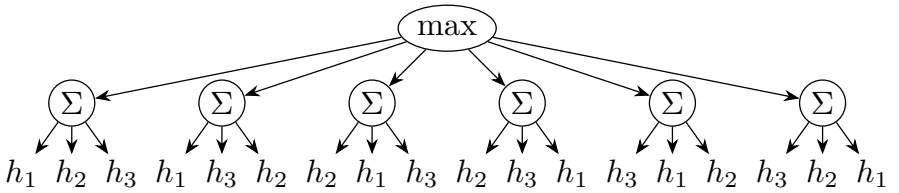


Figure 9.1: Naive all-order ADHG for $\mathcal{H} = \{h_1, h_2, h_3\}$. Cost functions are omitted for conciseness, so heuristic h_i at different locations represents the same abstraction heuristic but evaluated under different cost functions.

9.1 General ADHG Reduction Rules

Previous work used ADHGs to represent small, hand-crafted heuristics (Coles et al. 2008). To our knowledge, we are the first to use ADHGs to represent large, algorithmically generated heuristics containing up to millions of nodes. Since such large ADHGs can significantly impact evaluation efficiency, it becomes important automatically to minimize their size. To achieve this, we will now define automatic general reductions of ADHGs. They are general in the sense that they apply to ADHGs over arbitrary types of heuristics, not just those for computing h^{SCP}_*.

First, we want to remove all duplicated nodes and subgraphs from an ADHG. Duplicated nodes are nodes that will always evaluate to the same heuristic value because they represent the same mathematical operation. Duplicated subgraphs are a subgraph of duplicated nodes. This can be ensured by first only including unique evaluator nodes and giving each a unique index. Depending on the type of heuristics involved, this step might be more or less complex, since heuristic equivalence is in general a hard problem that involves iterating over all states $s \in S$. This step might also be done approximately, but then there is no guarantee of eliminating all structural redundancies.

For saturated cost partitioning heuristics over abstraction heuristics it is possible to employ our approach to eliminate all structural re-

dundancies, since equivalence between the evaluators can be checked by comparing the abstraction heuristics. We approximately eliminate duplicates by merging all abstraction instances $h_i \in \mathcal{H}$ that receive the same cost function. This approximation may miss equivalences since there can exist abstractions with different IDs that express the same heuristic. After all evaluator nodes have a unique index, internal nodes receive a unique index based on their type (sum/max) and the set of indices of their child nodes. It is important to give internal nodes a different type of index than evaluator nodes, e.g., by giving evaluator nodes positive indices and internal nodes negative indices. If two internal nodes have the same type and the same child-index set, they are structurally equivalent and can be merged.

While this eliminates structural redundancy, it does not address mathematical redundancy. After removing all obviously duplicate nodes and subgraphs, we therefore now address mathematical redundancies by creating reduction rules derived from the properties of the mathematical expression represented by the ADHG. The first proposition follows from the associativity of the sum and max operations.

Proposition 9.1 *Let c be an internal node in an ADHG with parent node p. If c and p are the same type, then they can be merged without altering the ADHG heuristic. The merge operation consists of:*

1. *Adding all children of c as children of p.*
2. *Removing the edge between p and c.*

An example is shown in the change from Figure 9.2c to Figure 9.2d. Consequently, in an ADHG that is fully reduced with respect to Proposition 9.1, all children of sum nodes are max nodes and vice versa.

Furthermore, maximum and sum operations on a single operand are the identity function, which is another point of possible reductions.

Proposition 9.2 *Let n be an internal node in an ADHG with a single child c. Then n can be removed from the ADHG without altering the ADHG heuristic. If n has parent nodes $p_1, \ldots, p_k$, n is removed by adding c as a child to each p_i for $1 \leq i \leq k$ and removing n as a child from each p_i. Otherwise, if n was the root, c becomes the new root. In both cases the edge between n and c is removed.*

An example reduction is shown in the change between Figures 9.2b and 9.2c.

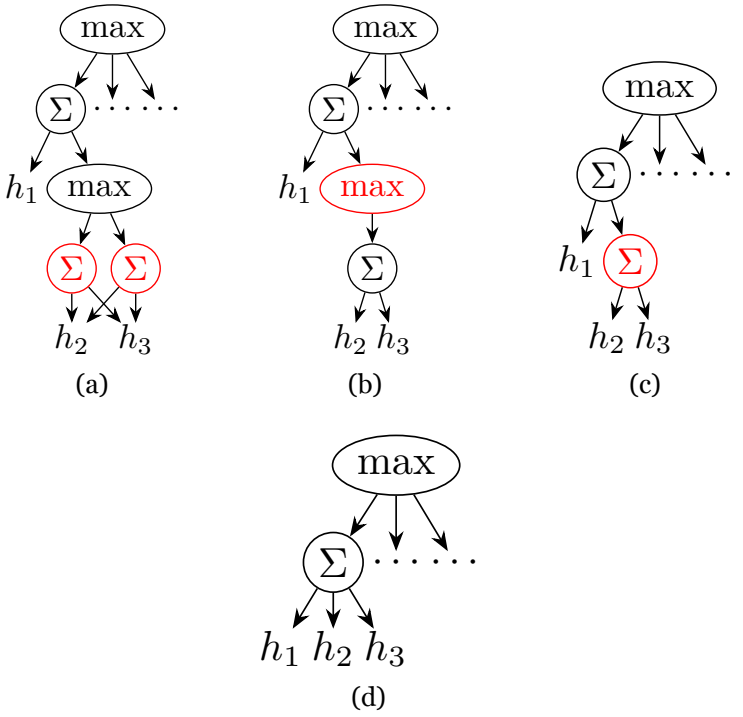


Figure 9.2: General reduction rules applied to a partially drawn example ADHG. Figure 9.2b shows the ADHG from Figure 9.2a without duplicated nodes. Applying Proposition 9.2 to Figure 9.2b yields the ADHG for Figure 9.2c and applying Proposition 9.1 to Figure 9.2c results in Figure 9.2d.

The final general reduction rule in this section simplifies common factors in max nodes and is essential for the proof of Theorem 10.4.

Proposition 9.3 *Let m be a max node in an ADHG that is fully reduced with respect to Proposition 9.1. If there exists a max or evaluator node n that is a grandchild of m and is a child of all children of m, then n can be factored out without altering the ADHG heuristic. Node n is factored out by:*

1. *Removing the edge between all children of m and n.*
2. *Creating a new sum node s and adding both m and n as its children.*
3. *Replacing all edges between m and its parents with an edge to s.*

Proof. Addition is distributive over the maximum operation. Therefore, any term $\max(a + b_1, a + b_2, \dots, a + b_n)$, where $a, b_1, \dots, b_n \in \mathbb{R} \cup \{-\infty, \infty\}$, is equal to $a + \max(b_1, \dots, b_n)$. □

An example for this reduction rule is shown in Figure 10.1.

Proposition 9.3 only applies if n appears in all grandchildren. With a small modification, it is possible to apply the same technique when n is only a grandchild of some of its children.

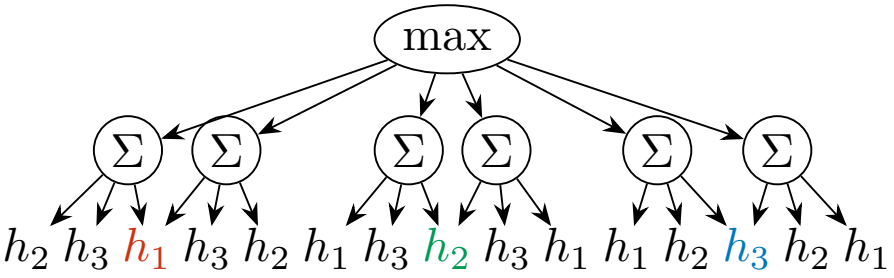
Proposition 9.4 *Let m be a max node in an ADHG that is fully reduced with respect to Proposition 9.1. If there exists a max or evaluator node n that is a grandchild of more than one of m's children, then n can be factored out without altering the ADHG heuristic. Node n is factored out by:*

1. *Splitting m into two max nodes m_1 and m_2, where m_1 contains all child nodes for which n is not a grandchild and m_2 contains all that have n as a grandchild.*
2. *Make m_1 a child of all parents of m.*
3. *Make m_2 a child of m_1.*
4. *Apply Proposition 9.3 on m_2.*

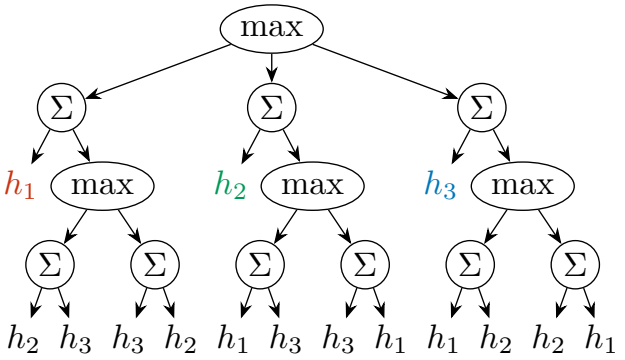
ADHGs can be interpreted as *symbolic expression graphs*, where the heuristics serve as variables and internal nodes represent the operation they evaluate to. A natural next step would therefore be to employ a *term rewriting system* or, more specifically, a *graph rewriting system* (Ehrig et al. 2006). Such systems could uncover and simplify more complex symmetric, associative, and distributive structures within the ADHGs. Due to the complexity of these approaches, the most practical option would be to employ an existing abstract algebra library like OSCAR (Decker et al. 2025). However, since all these approaches are quite distant from classical planning, we leave the integration of such tools and the development of better optimizations through reductions of ADHGs as future work.

9.2 Prefix-Merged ADHGs for SCP Heuristics

The ADHG structure was introduced as a general tool for representing additivity relations between arbitrary heuristics. However, it is partic-



(a) ADHG for Figure 9.1 with equivalent heuristics combined.



(b) Reduced representation of Figure 9.3a after applying Proposition 9.4.

Figure 9.3: Two ADHGs representing the same h^{SCP}_* heuristic for $\mathcal{H} = \{h_1, h_2, h_3\}$ as Figure 9.1 but as more compact graphs. Cost functions are omitted for conciseness, so heuristic h_i at different locations represents the same abstraction heuristic but evaluated under different cost functions.

ularly well suited for our use case because the common prefix of multiple orders can be captured as common ancestors in a graph. All orders for saturated cost partitioning that share a prefix will produce the same component heuristics until the orders differ. Since those will be shared between all orders, as shown in Figure 9.3a, it is possible to factor out the common prefixes step by step by applying Proposition 9.4. In the end, this will produce an ADHG where each layer with evaluator nodes corresponds to saturating one additional heuristic in an order, as shown in Figure 9.3b. We will call such an ADHG *prefix-merged*.

By putting evaluator nodes for shared heuristics higher in the graph, the ADHG can represent common prefixes of multiple orders compactly by creating a hierarchy of evaluator nodes. Conceptually, max nodes represent branching points where a common order prefix

splits into different saturation orders and sum nodes sequentially add more heuristics to the saturation order along a specific branch.

In a prefix-merged ADHG, the max node in the first layer has n branches leading to n lookup tables. Each of the n nodes in the second max node layer has $n-1$ branches. There are therefore $n \cdot (n-1)$ lookup tables in this layer. The final layer is an exception because it combines the last two lookup tables of each order. Therefore, it contains twice as many lookup tables as the previous layers. Figure 9.3b is an example of such an ADHG. Using the permutation function $P(x, y) = \frac{x!}{(x-y)!}$, the number of lookup tables in layer l for $1 \leq l \leq n-2$ is $P(n, l)$. The final layer contains $2P(n, n-1) = P(n, n-1) + P(n, n) = 2n!$, so when counting the final layer as two, it simplifies to just $P(n, l)$. An ADHG therefore reduces the number of lookup tables from the naive $n \cdot n!$ to $\sum_{l=1}^{n} P(n, l) = \sum_{k=0}^{n-1} \frac{n!}{k!}$. Since $\sum_{k=0}^{n-1} \frac{1}{k!}$ approximates Euler's number e, this yields approximately $e \cdot n!$ lookup tables. So this reduction reduces the number of lookup tables, but does not improve over the factorial scaling.

10 Eliminating Equivalent Orders

The previous reduction rules were post-hoc reduction rules that simplified the ADHG nodes after they were generated. These rules can be applied during the generation of an ADHG by checking completed subgraphs. This helps to reduce the memory bottleneck of computing h_*^{SCP}. However, it does not reduce the factorial complexity of iterating over all orders. To further reduce the complexity of computing h_*^{SCP}, we need to define a reduction that can eliminate redundant subgraphs before they are generated. We achieve this by eliminating equivalent orders from being considered during the generation of the h_*^{SCP} ADHG.

10.1 Order Independence Between Heuristics

To further reduce the size of ADHGs, we now exploit the specific properties of ADHGs for SCP. It is based on the insight that two different orders ω and ω' for SCP can induce the same SCP heuristic, i.e., $h_\omega^{\text{SCP}}(s) = h_{\omega'}^{\text{SCP}}(s)$ for all $s \in S$. Detecting and even predicting this equivalence between SCP heuristics is essential for efficiently computing h_*^{SCP}, as it allows us to drastically reduce the number of orders we need to consider explicitly. In contrast to the previous post-hoc reduc-

tion rules, this not only reduces the representation size of the heuristic, but also the complexity of generating the ADHG.

Figure 9.2 shows an example of how much simpler an ADHG can become through order independence. In this ADHG, the order of h_2 and h_3 does not matter after saturating h_1. With reduction rules, the ADHG is transformed step by step from Figures 9.2a to 9.2d. With a predictive test that could show the order-independence between h_2 and h_3, Figure 9.2d could have been computed immediately, skipping all other ADHGs.

Checking whether two orders yield the same SCP heuristic is challenging but can be approximated by comparing the underlying cost partitions.

Proposition 10.1 (Equivalent Orders) *Let ω and ω' be two orders over the same set of heuristics $\mathcal{H}$, and let cost be a cost function. If their cost partitions $\mathcal{C}_{\omega}^{\text{SCP}}$ and $\mathcal{C}_{\omega'}^{\text{SCP}}$ assign the same cost function to the same heuristics, so $\mathcal{C}_{\omega}^{\text{SCP}}(h_i) = \mathcal{C}_{\omega'}^{\text{SCP}}(h_i)$ for all $1 \leq i \leq n$, then $h_{\omega}^{\text{SCP}} = h_{\omega'}^{\text{SCP}}$. We say ω and ω' are* equivalent *under cost if this holds.*

The proof follows directly from the definition of saturated cost partitioning. Proposition 10.1 is not a necessary condition for the equivalence of two SCP heuristics since different cost functions can still result in equivalent SCP heuristics.

Example 10.1
Consider a cost function *cost* and a set of two heuristics $\mathcal{H} = \{h_1, h_2\}$ that induce the same abstract transition system $\mathcal{T}^{\alpha}$. The abstract transition system consumes all remaining costs when it is saturated. In this case, both possible orders $\omega_1 = \langle h_1, h_2 \rangle, \omega_2 = \langle h_2, h_1 \rangle$ produce the same SCP heuristic, since in both instances one of the abstract transition systems $\mathcal{T}^{\alpha}$ receives the full cost. However, the two underlying cost partitions $\mathcal{C}_{\omega_1} = \{h_1 \mapsto \textit{cost}, h_2 \mapsto 0\}, \mathcal{C}_{\omega_2} = \{h_1 \mapsto 0, h_2 \mapsto \textit{cost}\}$ differ.

To relate equivalent orders to abstraction heuristics, we define order-independent heuristics.

Definition 10.1 (Order Independence)
Two heuristics h_1 and h_2 are *order-independent* under cost function *cost* if $\omega = \langle h_1, h_2 \rangle$ and $\omega' = \langle h_2, h_1 \rangle$ are equivalent under *cost*.

To extend pairwise order independence to sets of heuristics, we introduce the notion of an *order-independent partition*.

Definition 10.2 (Order-Independent Partition)
Let $I = \{D_1, \dots, D_n\}$ be a partition of the set of heuristics $\mathcal{H}$, and let *cost* be a cost function for $\mathcal{H}$. Then I is an *order-independent partition*, and each D_i is an *order-dependent class* of this order-independent partition if the following condition holds. For every pair of orders ω, ω' from $\Omega(\mathcal{H})$ that preserve the relative order of heuristics within each order-dependent class, the resulting SCP heuristics h^{SCP}_{ω} and $h^{\text{SCP}}_{\omega'}$ are equal. Formally:

$$\forall \omega, \omega' \in \Omega(\mathcal{H}) : \left(\forall i, \omega|_{D_i} = \omega'|_{D_i}\right) \Rightarrow h^{\text{SCP}}_{\omega} = h^{\text{SCP}}_{\omega'} \tag{10.1}$$

where $\omega|_{D_i}$ is the restriction of ω onto the heuristics in the order-dependent class D_i.

Example 10.2
Consider the sets $\mathcal{H} = \{h_1, h_2, h_3\}$, $D_1 = \{h_1, h_2\}$ and $D_2 = \{h_3\}$. Then $I = \{D_1, D_2\}$ is an order-independent partition under *cost* if $\mathcal{C}^{\text{SCP}}_{\langle h_1, h_2, h_3\rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_1, h_3, h_2\rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_3, h_1, h_2\rangle}(h_i)$ and $\mathcal{C}^{\text{SCP}}_{\langle h_2, h_1, h_3\rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_2, h_3, h_1\rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_3, h_2, h_1\rangle}(h_i)$ for all $1 \leq i \leq 3$.

Order-independent partitions generalize the notion of order-independent pairs by requiring pairwise order independence between all elements in different order-dependent classes.

Theorem 10.2 *Let $I = \{D_1, \dots, D_n\}$ be a partition of the heuristics $\mathcal{H}$. It is an order-independent partition under cost function cost if for every order-dependent class $D \in I$, every heuristic $h_i \in D$, is order independent of every $h_j \in \mathcal{H} \setminus D_a$ under any cost function that can be obtained by saturating abstractions from $\mathcal{H} \setminus \{h_i, h_j\}$.*

Proof. Let $I = \{D_1, \dots, D_n\}$ be an order-independent partition of $\mathcal{H}$. Every order that preserves the relative positioning of heuristics inside each order-dependent class can be reached by subsequently swapping two order-adjacent heuristics from different order-dependent classes $h_1 \in D_a$ and $h_2 \in D_b$. Each such swap preserves the overall heuristic value if h_1 and h_2 are order-independent under the remaining costs after saturating all $h_i \in \mathcal{H} \setminus \{h_1, h_2\}$ that appear before h_1 and h_2 in the order. □

Theorem 10.2 shows that it is sufficient to check order independence between pairs of heuristics to detect order-independent partitions if order independence holds for all cost functions that can occur during the saturation process. To compute order-independent partitions of heuristics from pairwise order-independence information, we build a graph that reflects the opposite information, i.e., order *dependence*.

Definition 10.3 (Order-Dependence Graph)
The order-dependence graph for saturated cost partitioning is defined as $G_D(\mathcal{H}, \mathit{cost}) = \langle V, E \rangle$, for heuristics $\mathcal{H}$ and cost function *cost*. The vertex set V contains one vertex $v_h \in V$ for each heuristic $h \in \mathcal{H}$ and an edge $\langle h_1, h_2 \rangle \in E$ if $h_1 \in \mathcal{H}$ and $h_2 \in \mathcal{H}$ are order-dependent under any remaining cost function that can be reached by saturating abstractions from $\mathcal{H} \setminus \{h_1, h_2\}$.

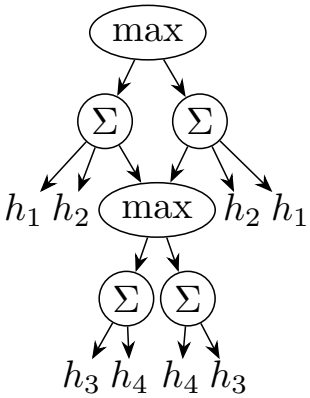
Given an order-dependence graph G_D, we can compute its connected components $D_1, \ldots, D_n$ which form the order-dependent classes of an order-independent partition $I = \{D_1, \ldots, D_n\}$.

Theorem 10.3 *The connected components $D_1, \ldots, D_n$ of an order-dependence graph $G_D(\mathcal{H}, \mathit{cost})$ form an order-independent partition $I = \{D_1, \ldots, D_n\}$ under cost.*

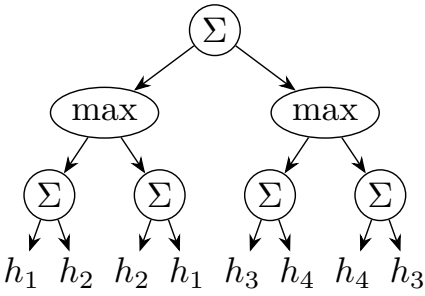
Proof. Let D_1 and D_2 be two connected components of an order-dependence graph $G_D(\mathcal{H}, \mathit{cost})$. Consider that the partition $I = \{D_1, D_2, \ldots\}$ is *not* an order-independent partition. This would only occur if either (1) D_1 and D_2 are not disjoint, or (2) there exists a heuristic $h \in D_1$ and a heuristic $h' \in D_2$ that are order-dependent under *cost*. However, by definition, two connected components of a graph are disjoint (contradicting Condition 1). Additionally, according to Definition 10.3, if two heuristics h and h' are order-dependent under *cost*, there is an edge $\langle h, h' \rangle$ in $G_D(\mathcal{H}, \mathit{cost})$, meaning h and h' cannot belong to different components (contradicting Condition 2). Thus, any two heuristics of two separate components are order-independent, which by Theorem 10.2 shows that the connected components of the order-dependence graph form an order-independent partition. □

We are only interested in the connected components and not in the actual graph structure of an order-dependence graph. It is therefore possible to efficiently build the graph implicitly with a disjoint set

(union-find) (Tarjan and Leeuwen 1984), making this an efficient approach to determine order-independent partitions, if pairwise order checks can be computed efficiently.



(a) Prefix-merged ADHG only containing orders where D_1 precedes D_2.



(b) ADHG from Figure 10.1a after applying Proposition 9.3.

Figure 10.1: Example of an order-independent partition with order-dependent classes leading to less complex ADHGs when applying Theorem 10.4. The set of heuristics is $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$ and the order-independent partition is $I = \{D_1, D_2\}$ where $D_1 = \{h_1, h_2\}$ and $D_2 = \{h_3, h_4\}$ are order-dependent classes. Note that, although the number of nodes increases from Figure 10.1a to Figure 10.1b, the number of mathematical operations required to evaluate the graph decreases.

10.2 An Order-Independence Reduction Rule

In the previous section we defined order-independent partitions for SCP and showed how to construct them from order-independent pairs of heuristics. Now, we derive a reduction rule specific to h^{SCP}_* for ADHGs exploiting order-independent partitions. This rule allows us to avoid constructing the full ADHG for a set of order-independent heuristics by directly constructing an equivalent, more concise ADHG. This allows us to *implicitly* consider all orders for saturated cost partitioning, making the computation of h^{SCP}_* easier.

To reduce the size of an ADHG for h^{SCP}_*, we aim to exclude all redundant orders. For equivalent orders, this can be achieved by including only one representative order from each order-dependent class. We choose an arbitrary order for the order-dependent classes, $D_1, \dots, D_n$, and only include the order where all elements of D_i are placed before all elements of D_{i+1}. There always exists at least one order in each class of equivalent orders that fulfills this requirement, as the definition of an order-independent partition is that the order between order-dependent classes is swappable. An example of such an ADHG can be found in Figure 10.1a.

When including only one order for each order-dependent class, the result is a reducible ADHG. Since the ordering of D_i before D_{i+1} is artificial and does not affect the heuristic value, we can completely remove the ordering between different order-dependent classes. This can be shown by applying Proposition 9.3 to an order-reduced ADHG.

Theorem 10.4 *Given an order-independent partition $I = \{D_1, \dots, D_n\}$, let G be the prefix-merged ADHG that includes only the orders where all elements of D_i are placed before all elements of D_{i+1}. Then Proposition 9.3 can be applied $n-1$ times until G has a sum node over one subgraph for each $D \in I$.*

Proof. G contains equivalent subgraphs for all orders of D_{i+1} under every subgraph for D_i. Regardless of the order chosen for D_i, the abstraction heuristics from D_{i+1} will receive the same costs according to Definition 10.2. This means that the max node max_{D_i} that branches over all orders of D_i contains the same subgraph $max_{D_{i+1}}$ in each of its branches, so Proposition 9.3 can be applied. This can be repeated

for all D_i with $i > 1$. The resulting ADHG has a sum node over one subgraph for each $D \in I$. □

Given a set of order-independent heuristics, we can therefore directly generate the reduced graph instead of first generating the semi-reduced graph and then iteratively applying Proposition 9.3. The following is an example of the reduction described in Theorem 10.4.

Example 10.3
Consider the set of heuristics $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$, where the order-independent partition is $I = \{D_1, D_2\}$ with order-dependent classes $D_1 = \{h_1, h_2\}$ and $D_2 = \{h_3, h_4\}$ under *cost*. Figure 10.1 shows two ADHGs representing h_*^{SCP} for $\mathcal{H}$. The first ADHG is a prefix-merged representation that considers all heuristics from D_1 before those from D_2. The second ADHG applies Proposition 9.3 as follows: In Figure 10.1a, the term $c := \max(h_3 + h_4, h_4 + h_3)$ is a common summand in both sum subgraphs. Therefore, the formula $\max(h_1 + h_2 + c, h_2 + h_1 + c)$ can be simplified to $\max(h_1 + h_2, h_2 + h_1) + c = \max(h_1 + h_2, h_2 + h_1) + \max(h_3 + h_4, h_4 + h_3)$. Summation is not commutative here because we are not showing the cost functions. Since the order of the heuristics influences the resulting cost partitioning and therefore the heuristic value, $h_1 + h_2$ is not the same as $h_2 + h_1$ because the abstractions receive different costs.

As this reduction is not post-hoc, computing order-independent partitions and generating the ADHG as a sum over the orders within each order-dependent class reduces the complexity of computing the perfect saturated cost partitioning heuristic. By exploiting order-independence in this way, we reduce the ADHG size from scaling in the number of heuristics $O(|\mathcal{H}|!)$ to a possibly much smaller scaling in the size of the largest order-dependent class $O(|D_1|! + \cdots + |D_n|!)$, which is $O(\max_{1 \leq i \leq n} |D_i|!)$.

This strategy can also be applied to subgraphs, increasing the reduction possibilities further. Whenever an abstraction is saturated, it can be removed from the set of heuristics to consider in the subgraph below. It is beneficial to revisit the order-independence graph after a reduction in the number of heuristics, since more orders can become independent.

Example 10.4
Let $\mathcal{H} = \{h_1, h_2, h_3\}$ be a set of heuristics for saturated cost partitioning under cost function *cost*. Assume all orders over $\mathcal{H}$ are non-equivalent except $\omega_a = \langle h_1, h_2, h_3 \rangle$ and $\omega_b = \langle h_1, h_3, h_2 \rangle$. In this case, the order-dependence graph $G_D(\mathcal{H}, \mathit{cost})$ will have one connected component, since for every pair $h_a, h_b \in \mathcal{H}$, there exist two orders $\omega, \omega' \in \Omega(\mathcal{H}) \setminus \{\omega_a, \omega_b\}$ where only the position of h_a, h_b is swapped, and all these orders are assumed to be non-equivalent. After saturating h_1, heuristics h_2 and h_3 become order-independent since we know that ω_a and ω_b are equivalent.

10.3 Computing Order Independence

We showed how to use order-independent partitions for reducing the size of the ADHG, but left out how to check order-independence for now. Since we want to avoid the computation of all orders, we cannot check if two orders are equivalent, but need to predict it. To be able to efficiently classify heuristics into order-independent partitions, we now define several sufficient criteria guaranteeing that two abstraction heuristics are order-independent.

As mentioned in Section 10.1, we are interested in cases where a change in saturation order does not affect the cost partitions. To capture order independence, we therefore need to define conditions under which an abstraction heuristic yields the same cost partitioning for different cost functions.

Theorem 10.5 *Two abstraction heuristics h_1 and h_2 are order-independent under cost function cost, if their minimum saturated cost functions are non-negative and form a cost partition.*

Proof. Let h_1 and h_2 be two abstraction heuristics, and let $\mathit{mscf}_1 = \mathit{saturate}(h_1, \mathit{cost})$ and $\mathit{mscf}_2 = \mathit{saturate}(h_2, \mathit{cost})$ be their minimum saturated cost functions. For order-independence, it needs to hold that the two orders $\omega = \langle h_1, h_2 \rangle$ and $\omega' = \langle h_2, h_1 \rangle$ are equivalent, i.e., the induced cost partitions $\mathcal{C}^{\mathrm{SCP}}_{\omega}$, $\mathcal{C}^{\mathrm{SCP}}_{\omega'}$ are equivalent. This is the case if $\mathit{saturate}(h_1, \mathit{cost}) = \mathit{saturate}(h_1, \mathit{cost} - \mathit{mscf}_2)$ and $\mathit{saturate}(h_2, \mathit{cost}) = \mathit{saturate}(h_2, \mathit{cost} - \mathit{mscf}_1)$. Assume that mscf_1 and mscf_2 are non-negative, i.e., no label is assigned negative costs, and that they form a cost partition under *cost*, i.e., $\mathit{mscf}_1 + \mathit{mscf}_2 \leq \mathit{cost}$.

Given the latter, saturating h_2 under the remaining cost $\mathit{rem} = \mathit{cost} - \mathit{mscf}_1$ offers each label at least the cost that h_2 needs for its minimum saturated cost function:

$$\begin{aligned} & \mathit{cost} \geq \mathit{mscf}_1 + \mathit{mscf}_2 \\ \text{iff} \quad & \mathit{cost} - \mathit{mscf}_1 \geq \mathit{mscf}_2 \\ \text{iff} \quad & \mathit{rem} \geq \mathit{mscf}_2 \end{aligned} \tag{10.2}$$

2 Under the assumption of *non-negativity* of the minimum saturated cost functions, saturating h_2 under *rem* offers each label at most the cost that h_2 is offered by the original cost function.

$$\begin{aligned} & \mathit{mscf}_1 \geq 0 \\ \text{iff} \quad & \mathit{cost} - \mathit{mscf}_1 \leq \mathit{cost} \\ \text{iff} \quad & \mathit{rem} \leq \mathit{cost} \end{aligned} \tag{10.3}$$

In combination, Equations 10.2 and 10.3 bound the costs after saturating h_1 from above and below: $\mathit{mscf}_2 \leq \mathit{rem} \leq \mathit{cost}$. Since changing the cost function of an abstraction to its saturated cost function does not change the abstract goal distance of any state, it holds that $h_2(\mathit{cost}, s) = h_2(\mathit{mscf}_2, s) = h_2(\mathit{rem}, s)$ for all states $s \in S$, from which it follows that $\mathit{saturate}(h_2, \mathit{cost}) = \mathit{saturate}(h_2, \mathit{rem})$. The same argument for h_1 follows from symmetry. □

The order independence conditions in Theorem 10.5 have two disadvantages. First, they are based on the minimum saturated cost function of a heuristic, which requires calculating the lookup tables, i.e., abstract goal distances, of an abstraction. Therefore, it cannot alleviate the costly computation of the lookup tables, making it an impractical predictor for determining order independence. Second, they depend on the original cost function. This makes it difficult to derive the order-independence of two heuristics $h_1, h_2 \in \mathcal{H}$ because we need to compute the remaining cost functions that can be obtained by saturating the heuristics $\mathcal{H} \setminus \{h_1, h_2\}$ (Theorem 10.2) to build the order-dependence graph.

In the following, we describe two sufficient conditions derived from Theorem 10.5 that do not rely on knowing the exact cost function. The first condition exploits that infinite label costs can never cause order dependence.

Theorem 10.6 (*scp**-∞) *If the cost of a label ℓ is infinite in cost function cost, then ℓ is never the cause of order-dependence under any cost function that can be reached by subtracting saturated costs from cost.*

Proof. If $\mathit{cost}(\ell) = \infty$, saturating any heuristic will always leave the remaining costs for ℓ infinite, since cost functions use left addition. Therefore, all abstractions will always receive the same cost for ℓ regardless of saturation order. □

For the second condition, we exploit the observation that two heuristics always form a cost partition if for every label at least one of the minimum saturated cost functions is guaranteed to be zero under any cost function.

Definition 10.4 (Saturation-Affecting Labels)
A label ℓ is *saturation-affecting* for an abstraction heuristic h if there exists a cost function *cost* under which the minimum saturated cost $\mathit{saturate}(h, \mathit{cost})(\ell)$ is neither zero nor negative infinity[1]. We write $L_h^{\mathit{sat\text{-}aff}}$ for the set of saturation-affecting labels for heuristic h.

It is easy to verify that every saturation-affecting label is also affecting (see Definition 2.9). To see that the opposite is not true, consider a label ℓ that only labels transitions between goal states. Then ℓ is affecting, but not saturation-affecting. As we are interested in labels that are not saturation-affecting for any abstraction heuristic, it is sufficient to show that the sets of saturation-affecting labels do not overlap.

Lemma 10.7 *If two heuristics h_1 and h_2 have disjoint sets of saturation-affecting labels, they are order-independent under any cost function.*

Proof. If two heuristics h_1 and h_2 have disjoint sets of saturation-affecting labels, none of the label costs that change upon saturating h_1 influence the heuristic values of h_2, and vice versa. Formally, for every label ℓ either $\mathit{mscf}_1(\ell)$ or $\mathit{mscf}_2(\ell)$ is zero or infinite, so the cost partitioning condition is fulfilled. The minimum saturated cost functions do not need to be *non-negative*, since having more cost available

[1] This formulation is similar to the affecting labels of Seipp, Keller, et al. (2017a) when extended to general cost functions (allowing negative costs), and explicitly states the connection to minimum saturated cost functions.

does not change the minimum saturated costs for a label that is not saturation-affecting. □

To approximate whether two heuristics have disjoint sets of saturation-affecting labels, we over-approximate the set of saturation-affecting labels for a heuristic.

Theorem 10.8 (*scp***-aff*) *Let* $L_h^{\textit{non-goal}}$ *be the set of labels* $\ell \in L$ *that label at least one transition* $s \xrightarrow{\ell} s' \in T^\alpha$ *between two distinct states* s *and* s'*, where at most one of the states is a goal state. If the sets* $L_{h_1}^{\textit{non-goal}}$ *and* $L_{h_2}^{\textit{non-goal}}$ *for two heuristics* h_1 *and* h_2 *are disjoint, then* h_1 *and* h_2 *are order-independent under any cost function cost.*

Proof. If a label ℓ is saturation-affecting for a heuristic h^α, then by the definition of minimum saturated cost functions there must be two abstract states s and s' such that $h^*_{\mathcal{T}^\alpha}(s) \ominus h^*_{\mathcal{T}^\alpha}(s') \neq 0$. So $h^*_{\mathcal{T}^\alpha}(s) \neq h^*_{\mathcal{T}^\alpha}(s')$ or $h^*_{\mathcal{T}^\alpha}(s) = h^*_{\mathcal{T}^\alpha}(s') = \infty$. (Abstract goal distances are always non-negative, so they cannot be $-\infty$.) In both cases, the two states cannot both be goal states at the same time, so $L_{h^\alpha}^{\textit{non-goal}} \subseteq L_{h^\alpha}^{\textit{sat-aff}}$. Therefore, if $L_{h_1}^{\textit{non-goal}} \cap L_{h_2}^{\textit{non-goal}} = \emptyset$ then $L_{h_1}^{\textit{sat-aff}} \cap L_{h_2}^{\textit{sat-aff}} = \emptyset$. Using Lemma 10.7 it follows that h_1 and h_2 are order-independent. □

Combining the conditions of Theorems 10.6 and 10.8 yields an efficient approximation of the order independence of two heuristics.

Theorem 10.9 (*scp***-aff-*∞) *For heuristics* h_1 *and* h_2*, let* $L_{h_1}^{\textit{non-goal}}$ *and* $L_{h_2}^{\textit{non-goal}}$ *be sets of labels as in Theorem 10.8. Then* h_1 *and* h_2 *are order-independent if*

$$(L_{h_1}^{\textit{non-goal}} \cap L_{h_2}^{\textit{non-goal}}) \setminus \{\ell \in L \mid \textit{cost}(\ell) = \infty\} = \emptyset.$$

All the order independence results from Theorems 10.6 to 10.9 can, in contrast to Theorem 10.5, be used for pruning connections in the order-dependence graph. This is because they generalize to all cost functions that can be reached by subtracting saturated costs from the original cost function. When creating an ADHG with any of the introduced approximation methods, we repeatedly test the order independence of the remaining not-yet-saturated heuristics at each node. This is because as demonstrated before in Example 10.4 it is beneficial even if the used independence check would be the perfect one. With our

approximations, it can even happen that, under a new remaining cost, the set of labels with infinite costs has grown. Therefore, with a smaller set of heuristics, the repeated check is much more likely to detect order-independence.

Recomputing the order-dependence graph is cheap because Theorem 10.8 is independent of any cost function. Therefore, the order-dependence graph does not need to be re-computed when the remaining costs or the set of heuristics change when using only Theorem 10.8. Since Theorem 10.6 is a simple check, itself or the combination (Theorem 10.9) is also cheaply computable.

We described two sufficient conditions for checking order-independence of heuristics. Other conditions for order-independence can be derived by approximating the conditions in Theorem 10.5 in different ways. While such investigations are interesting, we leave them for future work.

10.4 The Graph-SCP Algorithm

Combining all the results from the previous sections, we can now present an overview of the algorithm for computing h_*^{SCP}. The main idea is to compute order-independent partitions over all abstraction heuristics in $\mathcal{H}$ using the order-dependence graph $G_D(\mathcal{H}, \mathit{cost})$ with an approximative order-independence check between each pair of heuristics h_1 and h_2. Any of the Theorems 10.6, 10.8 and 10.9 can be used here, and we compare their performance in Section 11.1. These checks are computationally efficient, as affecting labels are independent of the cost function and can therefore be precomputed.

The main algorithm loop utilizes the conflict graph to decouple as many orders as possible by introducing a sum node over each component of the conflict graph, the order-dependent classes. Each class is captured by adding a new max node and only considers the abstractions within its class D_i for all subsequent steps. The classes are then assigned sum nodes and a lookup node for every abstraction heuristic contained, which are then removed from D_i for that branch. Then the order-dependence graph is subsequently recomputed on the now smaller set of heuristics and the new remaining costs.

Algorithm 3 shows this procedure in pseudocode. Additionally, we do not add abstraction heuristics with all-zero saturated costs to the

graph and apply the reduction rules from Section 9.1 wherever possible to minimize the graph's size. Consequently, the implementation differs significantly from the pseudocode.

Algorithm 3 Builds the unreduced ADHG representation for h^{SCP}_*

```
1: ▷ Input: H: set of heuristics, cost: cost function ◁
2: ▷ Output: ADHG for h*^SCP ◁
3: procedure BuildScpAdhg(H, cost)
4:     root ← CreateSumNode
5:     BuildScpSubtree(H, cost, root)
6:     return root
7: procedure BuildScpSubtree(H, cost, parent)
8:     I ← connected components of G_D(H, cost)
9:     for D ∈ I do
10:        maxNode ← createMaxNode
11:        attach maxNode to parent
12:        for h ∈ D do
13:            sumNode ← CreateSumNode
14:            attach sumNode to maxNode
15:            lookupNode ← CreateLookupNode(h, mscf_h(cost))
16:            attach lookupNode to sumNode
17:            if |D − {h}| > 0 then
18:                buildSCPSubtree(conflict − {h}, cost − mscf_h(cost),
                   sumNode)
```

11 Experimental Evaluation

The ADHG data structure and our reduction rules enable us to compute h_*^{SCP} without explicitly considering all possible orders of the component heuristics. The main questions we aim to answer with our experiments are:

- How helpful is the ADHG structure compared to a naive computation of h_*^{SCP}?
- How helpful are the order-independence approximations?
- How good are existing approximations of h_*^{SCP}?
- How close is h_*^{SCP} to optimal cost partitioning?

11.1 Computing the Perfect SCP Heuristic

In the first experiment, we compare different ways of computing h_*^{SCP}. The configurations vary in their treatment of order independence among heuristics and node sharing in the ADHG:

1) *scp*-naive* enumerates all heuristic orders.

2) *scp***-tree* represents h^{SCP}_* as a tree, ignores order independence, and applies only the general reduction rules from Section 9.1.

3) *scp***-graph* is like *scp***-tree*, but shares equivalent internal nodes, resulting in a graph.

4) *scp**-∞ uses a graph and exploits order independence between heuristics, applying only the infinite cost rule from Theorem 10.6.

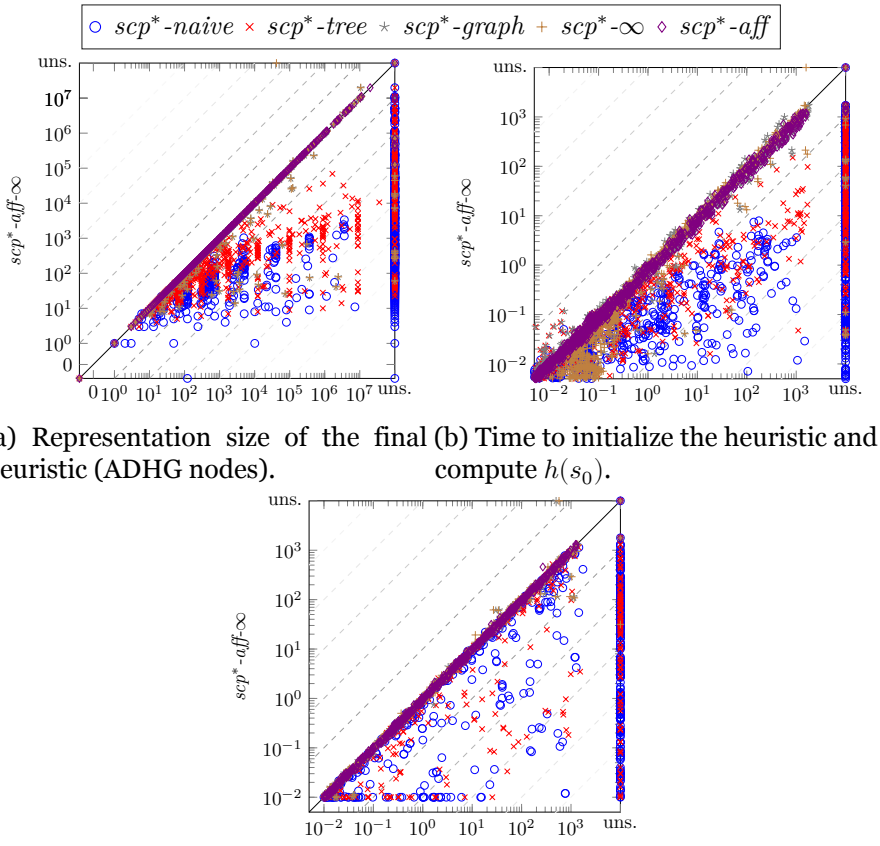
5) *scp***-aff* uses a graph and exploits order independence between heuristics, applying only the saturation-affecting labels rule from Theorem 10.8.

6) *scp***-aff*-∞ combines the latter two approaches. It uses a graph and applies both order-independence rules of *scp**-∞ and *scp***-aff*, i.e., the rule from Theorem 10.9.

The *initialize finished* row in Table 11.1 shows how often it is possible to construct the data structures required to compute the initial heuristic value $h^{\text{SCP}}_*(s_0)$ using the naive and ADHG-based approaches, as configurations become more sophisticated. The naive approach *scp***-naive*, which considers all orders explicitly, is infeasible for larger tasks because it usually runs out of memory during initialization as soon as more than ten abstractions are present ($\geq 3\,628\,800$ orders).

The table also shows that building h^{SCP}_* as a graph instead of a tree significantly raises the number of finished initializations, and adding the order-independence conditions further increases this number. Although our more advanced configurations run out of time more frequently than the naive variant, they avoid this memory bottleneck for constructing the data structure of the heuristic. Remarkably, we can compute $h^{\text{SCP}}_*(s_0)$ with *scp***-aff*-∞ even for tasks with up to 160 abstractions, corresponding to $\sim 10^{284}$ orders. However, due to the factorial nature of orders, computing h^{SCP}_* within the resource limits remains very challenging.

Figures 11.1a to 11.1c visualize the representation size of the heuristic (ADHG nodes), the time it takes to create the heuristic representation, and the resulting A^* search time for the different approaches. The plots reveal a significant advantage of the graph representation over the tree and naive variants across the benchmark tasks. Using a graph instead of a tree reduces the number of nodes in the ADHG by up



(a) Representation size of the final heuristic (ADHG nodes).

(b) Time to initialize the heuristic and compute $h(s_0)$.

(c) Search time for solving the planning tasks (after initialization).

Figure 11.1: Comparison of h^{SCP}_* variants. Each point represents one planning task, comparing *scp***-aff*-∞ on the y-axis with another variant (indicated by color) on the x-axis. Points below the diagonal indicate that *scp***-aff*-∞ is preferable for that task (Figure 11.1a: smaller representation size, Figure 11.1b: faster heuristic initialization, Figure 11.1c: faster problem-solving).

	initialize			search		
	finished	time	mem.	solved	time	mem.
h^{OCP}	1685	192	7	499	1186	0
$h^{\text{SCP}}_{\text{div}}$	1184	**0**	**0**	**1080**	40	764
$h^{\text{SCP}}_{\text{rnd}}$	**1884.0**	**0.0**	**0.0**	993.4	**6.2**	**88.4**
scp-naive*	1101	**47**	736	728	173	**200**
scp-tree*	1313	309	262	872	**149**	292
scp-graph*	1550	254	80	**983**	160	407
scp-*∞	1551	283	**50**	982	165	404
scp-aff*	1560	271	53	982	157	421
scp-aff-*∞	**1568**	261	55	**983**	156	429

Table 11.1: Summary of outcomes for optimal cost partitioning, two approximative SCP algorithms, and the h^{SCP}_* variants. All configurations use Cartesian goal abstractions for a benchmark suite of 1884 groundable planning tasks. The initialization part consists of all operations needed to compute $h(s_0)$. We average results for $h^{\text{SCP}}_{\text{rnd}}$ over five random seeds. The mem. (memory) and time columns refer to tasks that ran out of memory or time.

to four orders of magnitude (Figure 11.1a). This trend is also visible in the time it takes to create the necessary data structures in Figure 11.1b.

Considering search performance, the number of solved tasks is the same for both *scp*-graph* and *scp*-aff-*∞, and decreases by one if only one of the order-independent rules is applied (*scp*-*∞ and *scp*-aff*). However, the advantage of using order-independence conditions is evident in the ability to generate the ADHGs on larger abstraction sets. This is shown in the higher number of finished initializations in Table 11.1 and the large number of unsolved tasks on the x-axis in Figure 11.1a. This shows that, while those rules help to compute h^{SCP}_* for more tasks, the underlying tasks are too challenging to solve. Finally, Figure 11.1c compares the time it takes to solve planning tasks, showing that the tree and naive approaches are typically slower than *scp*-aff-*∞ due to their larger representation sizes. Comparing *scp*-aff-*∞ and

scp**-graph*, there is no clear winner for runtime or representation size, even though using the reduction rules clearly speeds up the generation of the ADHG.

11.2 Heuristic Accuracy

Now that we can efficiently compute h^{SCP}_*, we can, for the first time, evaluate the approximation quality of approximate SCP heuristics from the literature. For this analysis, we use our best approach *scp***-aff-*∞ to compute h^{SCP}_*, which takes advantage of all presented optimizations.

In particular, we compare against the approximate SCP heuristics $h^{\text{SCP}}_{\text{rnd}}$ and $h^{\text{SCP}}_{\text{div}}$. The former considers a single random ordering of the abstractions (reported results are averages over five runs). The latter iteratively samples a new state s until hitting a fixed time limit of $T = 100$ seconds. It then approximates a useful order ω for s with a greedy algorithm and retains h^{SCP}_ω if it is *diverse*. An order is diverse if it yields a higher estimate for any of a set of 1000 independently sampled states than all the SCP heuristics already retained (Seipp et al. 2020). We also empirically compare the perfect SCP heuristic h^{SCP}_* with the optimal cost partitioning heuristic h^{OCP} (Pommerening et al. 2015), which is known to theoretically dominate h^{SCP}_* (Pommerening et al. 2015; Seipp, Keller, et al. 2017a).

The left part of Table 11.1 holds results for these heuristics, and Figure 11.2 shows the necessary state expansions, i.e., the expansions before the last f-layer, for the compared approaches. Comparing h^{SCP}_* with $h^{\text{SCP}}_{\text{rnd}}$, we see that considering all orders instead of a single random one yields significantly more informative heuristic estimates, reducing the necessary expansions by a large margin. However, the picture is different for the state-of-the-art approach of computing diverse orders, $h^{\text{SCP}}_{\text{div}}$. Although h^{SCP}_* has fewer state expansions than $h^{\text{SCP}}_{\text{div}}$ for 334 tasks, the difference in heuristic accuracy between h^{SCP}_* and $h^{\text{SCP}}_{\text{div}}$ is small. This shows that, when considering the IPC domains and Cartesian goal abstractions, only a few orders are needed to obtain a heuristic estimate close to h^{SCP}_*. Furthermore, $h^{\text{SCP}}_{\text{div}}$ proves to be a remarkably efficient approach to obtaining relevant orders. In practice, using more orders than those found by $h^{\text{SCP}}_{\text{div}}$ offers little improvement when working with a moderate number of abstraction heuristics. Comparing the

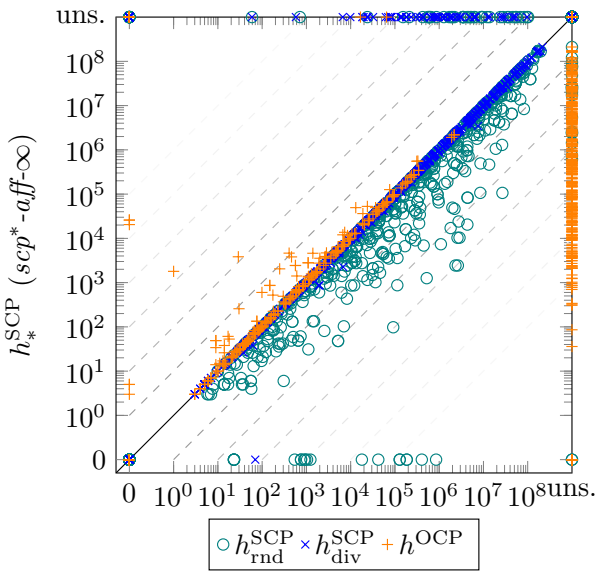


Figure 11.2: Necessary state expansions (i.e., expansions before the last f-layer) for the perfect SCP heuristic, the approximate SCP heuristics, and the optimal cost partitioning heuristic.

optimal cost partitioning heuristic h^{OCP} and the perfect SCP heuristic h_*^{SCP}, we see that having better cost partitions can indeed yield stronger heuristic estimates that are practically relevant due to a significant reduction in necessary expansions. However, for most tasks where we can compute h^{OCP}, the difference is small.

PART III

Conclusion

12 Conclusions

This work studied the perfect versions of two successful non-optimal cost partitioning heuristics: saturated post-hoc optimization and saturated cost partitioning. In the first part, we presented several cover rules derived from the sensitivity analysis of the SPhO linear program. These cover rules are provably safe in predicting when a cost partition computed with the SPhO LP can be reused in another state and result in the same heuristic value as if the LP had been solved. We showed that these cover rules can significantly reduce the number of LP computations in most planning tasks, and that a stronger cover rule nearly always results in fewer cost partitions. In addition, we showed how degeneracy and non-uniqueness in the SPhO LP can be reduced to improve cover rule performance and reduce the solution time of the linear programs. We further showed that there are clear tiebreaking criteria that can be used to find more versatile cost partitions that can be reused in more states. Our work suggests that other LP-based planning heuristics might benefit from a similar analysis, even though sensitivity analysis is most effective for simple LPs.

In the second part, we studied the perfect saturated cost partitioning heuristic, which maximizes over individual SCP heuristics computed for all orders of the component heuristics. Computing h_*^{SCP}

naively is infeasible due to the factorial scaling of maximizing over all orders. We established a connection to additive-disjunctive heuristic graphs and showed that they are well suited to addressing the computational challenge of computing h^{SCP}_*. By computing order-independent information to exclude provably equivalent orders, we were able to significantly reduce the size of these ADHGs. This representation allowed us to compute h^{SCP}_* for the first time for many planning tasks with a meaningful number of component heuristics. Building on this, we conducted the first empirical comparison of the perfect SCP heuristic, h^{SCP}_*, with state-of-the-art sampling-based SCP heuristics and the optimal cost partitioning heuristic. Our experiments show that state-of-the-art SCP heuristics provide heuristic information that is nearly perfect, relative to their upper bound of h^{SCP}_*, when using a moderate number of abstraction heuristics. This suggests that the greatest potential for obtaining even stronger cost partitioning heuristics lies in developing methods that go beyond SCP and reduce the remaining gap to optimal cost partitioning.

Bibliography

Alon, Noga, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor (2003). "The online set cover problem." In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: Association for Computing Machinery, pp. 100–105.

Andersen, Erling and Knud Andersen (Dec. 1995). "Presolving in linear programming." In: *Mathematical Programming* 71, pp. 221–245.

Bäckström, Christer and Bernhard Nebel (1995). "Complexity Results for SAS^+ Planning." In: *Computational Intelligence* 11.4, pp. 625–655.

Bonet, Blai and Héctor Geffner (2001). "Planning as Heuristic Search." In: *Artificial Intelligence* 129.1, pp. 5–33.

Bradley, Stephen P., Arnoldo C. Hax, and Thomas L. Magnanti (1977). *Applied Mathematical Programming*. Addison-Wesley.

Cai, Pu and Jin-Yi Cai (1997). "On the 100% rule of sensitivity analysis in linear programming." In: *Computing and Combinatorics*. Ed. by Tao Jiang and D. T. Lee. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 460–469.

Coles, Andrew, Maria Fox, Derek Long, and Amanda Smith (2008). "Additive-Disjunctive Heuristics for Optimal Planning." In: *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*. Ed. by Jussi Rinta-

nen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen. AAAI Press, pp. 44–51.

Dantzig, George B. (2002). "Linear Programming." In: *Operations Research* 50.1, pp. 42–47.

Decker, Wolfram, Christian Eder, Claus Fieker, Max Horn, and Michael Joswig, eds. (2025). *The Computer Algebra System OSCAR: Algorithms and Examples*. 1st ed. Vol. 32. Algorithms and Computation in Mathematics. Springer.

Drexler, Dominik, Daniel Gnad, Paul Höft, Jendrik Seipp, David Speck, and Simon Ståhlberg (2023). "Ragnarok." In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Edelkamp, Stefan (2001). "Planning with Pattern Databases." In: *Proceedings of the Sixth European Conference on Planning (ECP 2001)*. Ed. by Amedeo Cesta and Daniel Borrajo. AAAI Press, pp. 84–90.

Ehrig, Hartmut, Karsen Ehrig, Ulrike Prange, and Gabriele Taentzer (2006). *Fundamentals of Algebraic Graph Transformation*. 1st. Springer-Verlag.

Fišer, Daniel, Rostislav Horčík, and Antonín Komenda (2020). "Strengthening Potential Heuristics with Mutexes and Disambiguations." In: *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*. Ed. by J. Christopher Beck, Erez Karpas, and Shirin Sohrabi. AAAI Press, pp. 124–133.

Gal, T. (1986). "Shadow Prices and Sensitivity Analysis in Linear Programming under Degeneracy." In: *OR Spectrum* 8, pp. 59–71.

Gestrin, Elliot, Gustaf Söderholm, Paul Höft, Mauricio Salerno, Jendrik Seipp, and Daniel Gnad (2025). "Explainable Planning via Counterfactual Task Analysis for the Beluga Challenge and Beyond." In: *ICAPS 2025 Workshop on Human-Aware and Explainable Planning (HAXP)*.

Goldman, Robert P., Susanne Biundo, and Michael Katz, eds. (2021). *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*. AAAI Press.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Haslum, Patrik, Blai Bonet, and Héctor Geffner (2005). "New Admissible Heuristics for Domain-Independent Planning." In: *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*. AAAI Press, pp. 1163–1168.

Haslum, Patrik, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig (2007). "Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning." In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*. AAAI Press, pp. 1007–1012.

Helmert, Malte (2006). "The Fast Downward Planning System." In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.

Helmert, Malte, Patrik Haslum, and Jörg Hoffmann (2007). "Flexible Abstraction Heuristics for Optimal Sequential Planning." In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*. Ed. by Mark Boddy, Maria Fox, and Sylvie Thiébaux. AAAI Press, pp. 176–183.

Höft, Paul (2026). *Experiment data for the PhD Thesis "Computing Perfect Cost Partitioning Heuristics for Classical Planning"*. https://doi.org/10.5281/zenodo.18376875.

Höft, Paul, David Speck, Florian Pommerening, and Jendrik Seipp (2024). "Versatile Cost Partitioning with Exact Sensitivity Analysis." In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, pp. 276–280.

Höft, Paul, David Speck, and Jendrik Seipp (2023a). "Dofri." In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Höft, Paul, David Speck, and Jendrik Seipp (2023b). "Sensitivity Analysis for Saturated Post-hoc Optimization in Classical Planning." In: *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*. Ed. by Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Rădulescu. IOS Press, pp. 1044–1051.

Höft, Paul, David Speck, and Jendrik Seipp (2025). "Representing Perfect Saturated Cost Partitioning Heuristics in Classical Planning." In: *Proceedings of the Twenty-Second International Conference on Principles of Knowledge Representation and Reasoning (KR 2025)*. Ed. by Magdalena Ortiz, Renata Wassermann, and Torsten Schaub. IJCAI Organization, pp. 821–831.

Holte, Robert C., Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy (2006). "Maximizing over Multiple Pattern Databases

Speeds up Heuristic Search." In: *Artificial Intelligence* 170.16–17, pp. 1123–1136.

Kakkad, Dev A., Ignacio E. Grossmann, Bianca Springub, Christos Galanopoulos, Leonardo Salsano de Assis, Nga Tran, and John M. Wassick (2024). "Iterative MILP algorithm to find alternate solutions in linear programming models." In: *Optimization and Engineering* 25.4, pp. 2401–2424.

Karpas, Erez, Michael Katz, and Shaul Markovitch (2011). "When Optimal Is Just Not Good Enough: Learning Fast Informative Action Cost Partitionings." In: *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*. Ed. by Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert. AAAI Press, pp. 122–129.

Katz, Michael and Carmel Domshlak (2008). "Optimal Additive Composition of Abstraction-based Admissible Heuristics." In: *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*. Ed. by Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen. AAAI Press, pp. 174–181.

Katz, Michael and Carmel Domshlak (2010). "Optimal admissible composition of abstraction heuristics." In: *Artificial Intelligence* 174.12–13, pp. 767–798.

Kelner, Jonathan A. and Daniel A. Spielman (2006). "A Randomized Polynomial-Time Simplex Algorithm for Linear Programming." In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, pp. 51–61.

Koenig, Sven, Roni Stern, and Mauro Vallati, eds. (2023). *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press.

Korf, Richard E. and Ariel Felner (2002). "Disjoint Pattern Database Heuristics." In: *Artificial Intelligence* 134.1–2, pp. 9–22.

Kreft, Raphael, Clemens Büchner, Silvan Sievers, and Malte Helmert (2023). "Computing Domain Abstractions for Optimal Classical Planning with Counterexample-Guided Abstraction Refinement." In: *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. Ed. by Sven Koenig, Roni Stern, and Mauro Vallati. AAAI Press, pp. 221–226.

Matoušek, Jiří and Petr Škovroň (2007). "Removing Degeneracy May Require a Large Dimension Increase." In: *Theory of Computing* 3.8, pp. 159–177.

Paulraj, S. and P. Sumathi (2010). "A comparative study of redundant constraints identification methods in linear programming problems." In: *Mathematical Problems in Engineering* 2010.

Pearl, Judea (1981). "Heuristic Search Theory: Survey of Recent Results." In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI 1981)*. Ed. by Patrick J. Hayes. William Kaufmann, pp. 554–561.

Pearl, Judea (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pommerening, Florian (2017). "New Perspectives on Cost Partitioning for Optimal Classical Planning." PhD thesis. University of Basel.

Pommerening, Florian, Malte Helmert, Gabriele Röger, and Jendrik Seipp (2015). "From Non-Negative to General Operator Cost Partitioning." In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, pp. 3335–3341.

Pommerening, Florian, Thomas Keller, Valentina Halasi, Jendrik Seipp, Silvan Sievers, and Malte Helmert (2021). "Dantzig-Wolfe Decomposition for Cost Partitioning." In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*. Ed. by Robert P. Goldman, Susanne Biundo, and Michael Katz. AAAI Press, pp. 271–280.

Pommerening, Florian, Gabriele Röger, and Malte Helmert (2013). "Getting the Most Out of Pattern Databases for Classical Planning." In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. Ed. by Francesca Rossi. AAAI Press, pp. 2357–2364.

Pommerening, Florian, Gabriele Röger, Malte Helmert, and Blai Bonet (2014). "LP-based Heuristics for Cost-optimal Planning." In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*. Ed. by Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do. AAAI Press, pp. 226–234.

Rintanen, Jussi, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, eds. (2008). *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*. AAAI Press.

Russell, Stuart and Peter Norvig (2003). *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Seipp, Jendrik (2017). "Better Orders for Saturated Cost Partitioning in Optimal Classical Planning." In: *Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017)*. Ed. by Alex Fukunaga and Akihiro Kishimoto. AAAI Press, pp. 149–153.

Seipp, Jendrik (2021). "Online Saturated Cost Partitioning for Classical Planning." In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*. Ed. by Robert P. Goldman, Susanne Biundo, and Michael Katz. AAAI Press, pp. 317–321.

Seipp, Jendrik and Malte Helmert (2018). "Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning." In: *Journal of Artificial Intelligence Research* 62, pp. 535–577.

Seipp, Jendrik, Thomas Keller, and Malte Helmert (2017a). "A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning." In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*. Ed. by Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith. AAAI Press, pp. 259–268.

Seipp, Jendrik, Thomas Keller, and Malte Helmert (2017b). "Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning." In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*. Ed. by Satinder Singh and Shaul Markovitch. AAAI Press, pp. 3651–3657.

Seipp, Jendrik, Thomas Keller, and Malte Helmert (2020). "Saturated Cost Partitioning for Optimal Classical Planning." In: *Journal of Artificial Intelligence Research* 67, pp. 129–167.

Seipp, Jendrik, Thomas Keller, and Malte Helmert (2021). "Saturated Post-hoc Optimization for Classical Planning." In: *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*. Ed. by Kevin Leyton-Brown and Mausam. AAAI Press, pp. 11947–11953.

Seipp, Jendrik, Florian Pommerening, and Malte Helmert (2015). "New Optimization Functions for Potential Heuristics." In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*. Ed. by Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein. AAAI Press, pp. 193–201.

Seipp, Jendrik, Florian Pommerening, Silvan Sievers, and Malte Helmert (2017). *Downward Lab*. https://doi.org/10.5281/zenodo.790461.

Sierksma, Gerard and Yori Zwols (2015). *Linear and Integer Optimization. Theory and Practice*. 3rd ed. Advances in Applied Mathematics. CRC Press.

Speck, David, Paul Höft, Daniel Gnad, and Jendrik Seipp (2023). "Finding Matrix Multiplication Algorithms with Classical Planning." In: *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. Ed. by Sven Koenig, Roni Stern, and Mauro Vallati. AAAI Press, pp. 411–416.

Spielman, Daniel A. and Shang-Hua Teng (2004). "Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time." In: *Journal of the ACM* 51.3.

Taitler, Ayal, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp (2024). "The 2023 International Planning Competition." In: *AI Magazine* 45.2, pp. 280–296.

Tarjan, Robert E. and Jan van Leeuwen (1984). "Worst-case Analysis of Set Union Algorithms." In: *Journal of the ACM* 31.2, pp. 245–281.

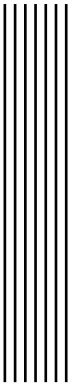
Tenth International Planning Competition (IPC-10): Planner Abstracts (2023).

Vanderbei, Robert J. (2013). *Linear Programming. Foundations and Extensions*. 4th. Springer-Verlag.

Wendell, Richard E. (1985). "The Tolerance Approach to Sensitivity Analysis in Linear Programming." In: *Management Science* 31.5, pp. 564–578.

Yang, Fan, Joseph Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner (2008). "A General Theory of Additive State Space Abstractions." In: *Journal of Artificial Intelligence Research* 32, pp. 631–662.

List of Figures



List of Tables

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology
Linköping Studies in Arts and Sciences
Linköping Studies in Statistics
Linköping Studies in Information Science

Linköping Studies in Science and Technology

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall**: Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91- 7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System, 1978, ISBN 91- 7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91- 7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372- 805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-X.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN 91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91-7373-207-9.

No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91-7373-208-7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91-7373-212-5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91-7373-258-3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X.

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing – An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5.

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8

No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.

No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.

No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.

No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.

No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.

No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.

No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.

No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.

No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.

No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.

No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.

No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.

No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.

No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for Satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.

No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.

No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.

No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.

No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.

No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.

No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.

No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.

No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.

No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.

No 1139 **Per Wikberg:** Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.

No 1143 **Mehdi Amirijoo:** QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.

No 1150 **Sanny Syberfeldt:** Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.

No 1155 **Beatrice Alenljung:** Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.

No 1156 **Artur Wilk:** Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.

No 1183 **Adrian Pop:** Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.

No 1185 **Jörgen Skågeby:** Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.

No 1187 **Imad-Eldin Ali Abugessaisa:** Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.

No 1204 **H. Joe Steinhauer:** A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.

No 1222 **Anders Larsson:** Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.

No 1238 **Andreas Borg:** Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.

No 1240 **Fredrik Heintz:** DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.

No 1241 **Birgitta Lindström:** Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.

No 1244 **Eva Blomqvist:** Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.

No 1249 **Rogier Woltjer:** Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.

No 1260 **Gianpaolo Conte:** Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.

No 1262 **AnnMarie Ericsson:** Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.

No 1266 **Jiri Trnka:** Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.

No 1268 **Bahlol Rahimi:** Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.

No 1274 **Fredrik Kuivinen:** Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.

No 1281 **Gunnar Mathiason:** Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.

No 1290 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.

No 1294 **Johan Thapper:** Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.

No 1306 **Susanna Nilsson:** Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.

No 1313 **Christer Thörn:** On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.

No 1321 **Zhiyuan He:** Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.

No 1333 **David Broman:** Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.

No 1337 **Alexander Siemers:** Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.

No 1354 **Mikael Asplund:** Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.

No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.

No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish, 2011, ISBN 978-91-7393-154-0.

No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.

No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.

No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control – Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.

No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.

No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.

No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.

No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.

No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.

No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.

No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.

No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.

No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.

No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.

No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.

No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.

No 1559 **Sergiu Rafiliu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.

No 1581 **Usman Dastgeer**: Performance-aware Component Composition for GPU-based Systems, 2014, ISBN 978-91-7519-383-0.

No 1602 **Cai Li**: Reinforcement Learning of Locomotion based on Central Pattern Generators, 2014, ISBN 978-91-7519-313-7.

No 1652 **Roland Samlaus**: An Integrated Development Environment with Enhanced Domain-Specific Interactive Model Validation, 2015, ISBN 978-91-7519-090-7.

No 1663 **Hannes Uppman**: On Some Combinatorial Optimization Problems: Algorithms and Complexity, 2015, ISBN 978-91-7519-072-3.

No 1664 **Martin Sjölund**: Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models, 2015, ISBN 978-91-7519-071-6.

No 1666 **Kristian Stavåker**: Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures, 2015, ISBN 978-91-7519-068-6.

No 1680 **Adrian Lifa**: Hardware/Software Codesign of Embedded Systems with Reconfigurable and Heterogeneous Platforms, 2015, ISBN 978-91-7519-040-2.

No 1685 **Bogdan Tanasa**: Timing Analysis of Distributed Embedded Systems with Stochastic Workload and Reliability Constraints, 2015, ISBN 978-91-7519-022-8.

No 1691 **Håkan Warnquist**: Troubleshooting Trucks – Automated Planning and Diagnosis, 2015, ISBN 978-91-7685-993-3.

No 1702 **Nima Aghaee**: Thermal Issues in Testing of Advanced Systems on Chip, 2015, ISBN 978-91-7685-949-0.

No 1715 **Maria Vasilevskaya**: Security in Embedded Systems: A Model-Based Approach with Risk Metrics, 2015, ISBN 978-91-7685-917-9.

No 1729 **Ke Jiang**: Security-Driven Design of Real-Time Embedded System, 2016, ISBN 978-91-7685-884-4.

No 1733 **Victor Lagerkvist**: Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications, 2016, ISBN 978-91-7685-856-1.

No 1734 **Chandan Roy**: An Informed System Development Approach to Tropical Cyclone Track and Intensity Forecasting, 2016, ISBN 978-91-7685-854-7.

No 1746 **Amir Aminifar**: Analysis, Design, and Optimization of Embedded Control Systems, 2016, ISBN 978-91-7685-826-4.

No 1747 **Ekhiotz Vergara**: Energy Modelling and Fairness for Efficient Mobile Communication, 2016, ISBN 978-91-7685-822-6.

No 1748 **Dag Sonntag**: Chain Graphs – Interpretations, Expressiveness and Learning Algorithms, 2016, ISBN 978-91-7685-818-9.

No 1768 **Anna Vapen**: Web Authentication using Third-Parties in Untrusted Environments, 2016, ISBN 978-91-7685-753-3.

No 1778 **Magnus Jandinger**: On a Need to Know Basis: A Conceptual and Methodological Framework for Modelling and Analysis of Information Demand in an Enterprise Context, 2016, ISBN 978-91-7685-713-7.

No 1798 **Rahul Hiran**: Collaborative Network Security: Targeting Wide-area Routing and Edge-network Attacks, 2016, ISBN 978-91-7685-662-8.

No 1813 **Nicolas Melot**: Algorithms and Framework for Energy Efficient Parallel Stream Computing on Many-Core Architectures, 2016, ISBN 978-91-7685-623-9.

No 1823 **Amy Rankin**: Making Sense of Adaptations: Resilience in High-Risk Work, 2017, ISBN 978-91-7685-596-6.

No 1831 **Lisa Malmberg**: Building Design Capability in the Public Sector: Expanding the Horizons of Development, 2017, ISBN 978-91-7685-585-0.

No 1851 **Marcus Bendtsen**: Gated Bayesian Networks, 2017, ISBN 978-91-7685-525-6.

No 1852 **Zlatan Dragisic**: Completion of Ontologies and Ontology Networks, 2017, ISBN 978-91-7685-522-5.

No 1854 **Meysam Aghighi**: Computational Complexity of some Optimization Problems in Planning, 2017, ISBN 978-91-7685-519-5.

No 1863 **Simon Ståhlberg**: Methods for Detecting Unsolvable Planning Instances using Variable Projection, 2017, ISBN 978-91-7685-498-3.

No 1879 **Karl Hammar**: Content Ontology Design Patterns: Qualities, Methods, and Tools, 2017, ISBN 978-91-7685-454-9.

No 1887 **Ivan Ukhov**: System-Level Analysis and Design under Uncertainty, 2017, ISBN 978-91-7685-426-6.

No 1891 **Valentina Ivanova**: Fostering User Involvement in Ontology Alignment and Alignment Evaluation, 2017, ISBN 978-91-7685-403-7.

No 1902 **Vengatanathan Krishnamoorthi**: Efficient HTTP-based Adaptive Streaming of Linear and Interactive Videos, 2018, ISBN 978-91-7685-371-9.

No 1903 **Lu Li**: Programming Abstractions and Optimization Techniques for GPU-based Heterogeneous Systems, 2018, ISBN 978-91-7685-370-2.

No 1913 **Jonas Rybing**: Studying Simulations with Distributed Cognition, 2018, ISBN 978-91-7685-348-1.

No 1936 **Leif Jonsson**: Machine Learning-Based Bug Handling in Large-Scale Software Development, 2018, ISBN 978-91-7685-306-1.

No 1964 **Arian Maghazeh**: System-Level Design of GPU-Based Embedded Systems, 2018, ISBN 978-91-7685-175-3.

No 1967 **Mahder Gebremedhin**: Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation, 2019, ISBN 978-91-7685-163-0.

No 1984 **Anders Andersson**: Distributed Moving Base Driving Simulators – Technology, Performance, and Requirements, 2019, ISBN 978-91-7685-090-9.

No 1993 **Ulf Kargén**: Scalable Dynamic Analysis of Binary Code, 2019, ISBN 978-91-7685-049-7.

No 2001 **Tim Overkamp**: How Service Ideas Are Implemented: Ways of Framing and Addressing Service Transformation, 2019, ISBN 978-91-7685-025-1.

No 2006 **Daniel de Leng**: Robust Stream Reasoning Under Uncertainty, 2019, ISBN 978-91-7685-013-8.

No 2048 **Biman Roy**: Applications of Partial Polymorphisms in (Fine-Grained) Complexity of Constraint Satisfaction Problems, 2020, ISBN 978-91-7929-898-2.

No 2051 **Olov Andersson**: Learning to Make Safe Real-Time Decisions Under Uncertainty for Autonomous Robots, 2020, ISBN 978-91-7929-889-0.

No 2065 **Vanessa Rodrigues**: Designing for Resilience: Navigating Change in Service Systems, 2020, ISBN 978-91-7929-867-8.

No 2082 **Robin Kurtz**: Contributions to Semantic Dependency Parsing: Search, Learning, and Application, 2020, ISBN 978-91-7929-822-7.

No 2108 **Shanai Ardi**: Vulnerability and Risk Analysis Methods and Application in Large Scale Development of Secure Systems, 2021, ISBN 978-91-7929-744-2.

No 2125 **Zeinab Ganjei**: Parameterized Verification of Synchronized Concurrent Programs, 2021, ISBN 978-91-7929-697-1.

No 2153 **Robin Keskisärkkä**: Complex Event Processing under Uncertainty in RDF Stream Processing, 2021, ISBN 978-91-7929-621-6.

No 2168 **Rouhollah Mahfouzi**: Security-Aware Design of Cyber-Physical Systems for Control Applications, 2021, ISBN 978-91-7929-021-4.

No 2205 **August Ernstsson**: Pattern-based Programming Abstractions for Heterogeneous Parallel Computing, 2022, ISBN 978-91-7929-195-2.

No 2218 **Huanyu Li**: Ontology-Driven Data Access and Data Integration with an Application in the Materials Design Domain, 2022, ISBN 978-91-7929-267-6.

No 2219 **Evelina Rennes**: Automatic Adaption of Swedish Text for Increased Inclusion, 2022, ISBN 978-91-7929-269-0.

No 2220 **Yuanbin Zhou**: Synthesis of Safety-Critical Real-Time Systems, 2022, ISBN 978-91-7929-271-3.

No 2247 **Azeem Ahmad**: Contributions to Improving Feedback and Trust in Automated Testing and Continuous Integration and Delivery, 2022, ISBN 978-91-7929-422-9.

No 2248 **Ana Kuštrak Korper**: Innovating Innovation: Understanding the Role of Service Design in Service Innovation, 2022, ISBN 978-91-7929-424-3.

No 2256 **Adrian Horga:** Performance and Security Analysis for GPU-Based Applications, 2022, ISBN 978-91-7929-487-8.

No 2262 **Mattias Tiger:** Safety-Aware Autonomous Systems: Preparing Robots for Life in the Real World 2022, ISBN 978-91-7929-501-1.

No 2266 **Chih-Yuan Lin:** Network-based Anomaly Detection for SCADA Systems: Traffic Generation and Modeling, 2022, ISBN 978-91-7929-517-2.

No 2280 **Filip Strömbäck:** Teaching and Learning Concurrent Programming in the Shared Memory Model, 2023, ISBN 978-91-8075-000-4.

No 2298 **Fiona Lambe:** Devising Capabilities: Service Design for Development Interventions, 2023, ISBN 978-91-8075-080-6.

No 2309 **Alachew Mengist:** Model-Based Tools Integration and Ontology-Driven Traceability in Model-Based Development Environments, 2023, ISBN 978-91-8075-143-8.

No 2322 **Mariusz Wzorek:** Selected Functionalities for Autonomous Intelligent Systems in Public Safety Scenarios, 2023, ISBN 978-91-8075-195-7.

No 2329 **Felipe Boeira:** Authentic Communication and Trustworthy Location in Mobile Networks, 2023, ISBN 978-91-8075-256-5.

No 2351 **Johan Källström:** Reinforcement Learning for Improved Utility of Simulation-Based Training, 2023, ISBN 978-91-8075-366-1.

No 2364 **Jenny Kunz:** Understanding Large Language Models: Towards Rigorous and Targeted Interpretability Using Probing Classifiers and Self-Rationalisation, 2024, ISBN 978-91-8075-470-5.

No 2366 **Sijin Cheng:** Query Processing over Heterogeneous Federations of Graph Data, 2024, ISBN 978-91-8075-488-0.

No 2368 **George Osipov:** On Infinite-Domain CSPs Parameterized by Solution Cost, 2024, ISBN 978-91-8075-496-5.

No 2382 **Fredrik Präntare:** Dividing the Indivisible: Algorithms, Empiricial Advances, and Complexity Results for Value-Maximizing Combinatorial Assignment Problems, 2024, ISBN 978-91-8075-600-6.

No 2383 **Alireza Mohammadinodooshan:** Data-driven Contributions to Understanding User Engagement Dynamics on Social Media, 2024, ISBN 978-91-8075-606-8.

No 2384 **Rodrigo Saar de Moraes:** Exploring Trade-offs in Concept Design of Integrated Modular Avionic Platform Configurations: Topology Generation, Resource Adequacy, and Dependability, 2024, ISBN 978-91-8075-13-16.

No 2392 **Minh Ha Le:** Beyond Recognition: Privacy Protections in a Surveilled World, 2024, ISBN 978-91-8075-675-4.

No 2403 **Klervie Toczé:** Orchestrating a Resource-aware Edge, 2024, ISBN 978-91-8075-747-8.

No 2264 **Robert Johansson:** Empirical Studies in Machine Psychology, 2024, ISBN 978-91-7929-505-9.

No 2429 **Mina Niknafs:** Prediction-Based Resource Management for Heterogeneous Multi-Core Embedded Systems, 2025, ISBN 978-91-8075-960-1.

No 2437 **Antonia Arvanitaki:** Performance Analysis of Wireless Systems with Security Constraints, 2025, ISBN 978-91-8118-015-2.

No 2439 **Dominik Drexler:** Learning and Exploiting Subgoal Structures in Classical Planning: Towards Reliable and Transparent Intelligent Agents that Learn to Plan on Multiple Levels, 2025, ISBN 978-91-8118-019-0.

No 2445 **Amirhossein Ahmadian:** Handling Novel and Out-Of-Distribution Data in Deep Learning: OOD Detection and Shortcut Mitigation, 2025, ISBN 978-91-8118-073-2.

No 2453 **Joel Oskarsson:** Modeling Spatio-Temporal Systems with Graph-based Machine Learning, 2025, ISBN 978-91-8118-116-6.

No 2454 **John Törnblom:** Efficient Formal Reasoning about the Trustworthiness of Tree Ensembles, 2025, ISBN 978-91-8118-128-9.

No 2461 **David Hasselquist:** Toward Secure and Privacy-Preserving Communication over Non-Trusted Networks, 2025, ISBN 978-91-8118-182-1.

No 2462 **Filip Ekström Kelvinius:** Deep Learning for the Atomic Scale, 2025, ISBN 978-91-8118-184-5.

No 2471 **Leif Eriksson:** Infinite-Domain CSPs and QBF: Fine-Grained and Parameterized Complexity, 2025, ISBN 978-91-8118-213-2.

No 2491 **John Tinnerholm:** Dynamic and Variable-Structured System Modeling for Equation-Based Languages, 2025, ISBN 978-91-8118-336-9.

No 2495 **Md Fahim Sikder:** Representative Synthetic Data for Fair Decision Making, 2026, ISBN 978-91-8118-374-0.

No 2502 **Ehsan Doostmohammadi:** Toward Understanding and Enhancing the Training and Evaluation of Language Models, 2026, ISBN 978-91-8118-443-3.

No 2503 **Dennis Granåsen:** Data-driven Team Development, 2026, ISBN 978-91-8118-450-1.

No 2504 **Paul Höft:** Computing Perfect Cost Partitioning Heuristics for Classical Planning, 2026, ISBN 978-91-8118-452-5.

Linköping Studies in Arts and Sciences

No 504 **Ing-Marie Jonsson:** Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.

No 586 **Fabian Segelström:** Stakeholder Engagement for Service Design: How service designers identify and communicate insights, 2013, ISBN 978-91-7519-554-4.

No 618 **Johan Blomkvist:** Representing Future Situations of Service: Prototyping in Service Design, 2014, ISBN 978-91-7519-343-4.

No 620 **Marcus Mast:** Human-Robot Interaction for Semi-Autonomous Assistive Robots, 2014, ISBN 978-91-7519-319-9.

No 677 **Peter Berggren:** Assessing Shared Strategic Understanding, 2016, ISBN 978-91-7685-786-1.

No 695 **Mattias Forsblad:** Distributed cognition in home environments: The prospective memory and cognitive practices of older adults, 2016, ISBN 978-91-7685-686-4.

No 787 **Sara Nygårdhs:** Adaptive behaviour in traffic: An individual road user perspective, 2020, ISBN 978-91-7929-857-9.

No 811 **Sam Thellman:** Social Robots as Intentional Agents, 2021, ISBN, 978-91-7929-008-5.

No 878 **Sofia Thunberg:** Companion Robots for Older Adults: A Mixed-Methods Approach to Deployments in Care Homes 2024, ISBN, 978-91-8075-573-3.

No 891 **Kajsa Weibull:** Emergency Vehicle Approaching: Warning Drivers using Cooperative Intelligent Transport Systems 2024, ISBN, 978-91-8075-804-8.

No 898 **Emma Mainza Chilufya:** Human-Centered Design of Socially Interactive Virtual Agents 2025, ISBN, 978-91-8075-933-5.

No 900 **Oscar Bjurling:** Designing Human-Swarm Interaction Systems 2025, ISBN, 978-91-8075-958-8.

Linköping Studies in Statistics

No 9 **Davood Shahsavani:** Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.

No 10 **Karl Wahlin:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN 978-91-7393-792-4.

No 11 **Oleg Sysoev:** Monotonic regression for large multivariate datasets, 2010, ISBN 978-91-7393-412-1.

No 13 **Agné Burauskaite-Harju:** Characterizing Temporal Change and Inter-Site Correlations in Daily and Sub-daily Precipitation Extremes, 2011, ISBN 978-91-7393-110-6.

No 14 **Måns Magnusson:** Scalable and Efficient Probabilistic Topic Model Inference for Textual Data, 2018, ISBN 978-91-7685-288-0.

No 15 **Per Sidén:** Scalable Bayesian spatial analysis with Gaussian Markov random fields, 2020, 978-91-7929-818-0.

No 16 **Caroline Svahn:** Prediction Methods for High Dimensional Data with Censored Covariates, 2022, 978-91-7929-398-7.

No 17 **Héctor Rodriguez Déniz:** Bayesian Models for Spatiotemporal Data from Transportation Networks, 2023, 978-91-8075-035-6.

No 18 **Amanda Olmin:** Perspectives on Predictive and Annotation Uncertainty in Probablistic Machine Learning 2024, 978-91-8075-798-0.

Linköping Studies in Information Science

No 1 **Karin Axelsson:** Metodisk systemstrukturering- att skapa samstämmighet mellan informationssystem-arkitektur och verksamhet, 1998. ISBN 9172-19-296-8.

No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998, ISBN 9172-19-299-2.

No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN 91-7219-606-8.

No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000, ISBN 91-7219-811-7.

No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X.

No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.

No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.

No 8 **Ulf Seigerroth:** Att förstå och förändra system-utvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN 91-7373-736-4.

No 9 **Karin Hedström:** Spår av datoriseringens värden – Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.

No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.

No 11 **Fredrik Karlsson:** Method Configuration method and computerized tool support, 2005, ISBN 91-85297-48-8.

No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent

- mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.

FACULTY OF SCIENCE AND ENGINEERING

Linköping Studies in Science and Technology. Dissertation No. 2504, 2026
Department of Computer and Information Science

Linköping University
SE-581 83 Linköping, Sweden

www.liu.se