Partial Delete Relaxation, Unchained: On Intractable Red-Black Planning and Its Applications

Daniel Gnad and **Marcel Steinmetz** and **Mathäus Jany** and **Jörg Hoffmann**
Saarland University, Saarbrücken, Germany
{gnad,steinmetz,hoffmann}@cs.uni-saarland.de, s9majany@stud.uni-saarland.de

Ivan Serina and **Alfonso Gerevini**
University of Brescia, Brescia, Italy
{ivan.serina,gerevini}@ing.unibs.it

Abstract

Partial delete relaxation methods, like red-black planning, are extremely powerful, allowing in principle to force relaxed plans to behave like real plans in the limit. Alas, that power has so far been chained down by the computational overhead of the use as heuristic functions, necessitating to compute a relaxed plan on every search state. For red-black planning in particular, this has entailed an exclusive focus on tractable fragments. We herein unleash the power of red-black planning on two applications not necessitating such a restriction: *(i) generating seed plans for plan repair*, and *(ii) proving planning task unsolvability*. We introduce a method allowing to generate red-black plans for arbitrary inputs – intractable red-black planning – and we evaluate its use for (i) and (ii). With (i), our results show promise and outperform standard baselines in several domains. With (ii), we obtain substantial, in some domains dramatic, improvements over the state of the art.

Introduction

The use of *relaxations* (aka abstractions, over-approximations) has been extremely successful in AI Planning, for the generation of heuristic functions guiding the search (e. g. (Haslum and Geffner 2000; Bonet and Geffner 2001; Hoffmann and Nebel 2001; Edelkamp 2001; Gerevini, Saetti, and Serina 2003; Helmert 2006; Haslum et al. 2007; Helmert and Domshlak 2009; Richter and Westphal 2010; Helmert et al. 2014)). The *delete relaxation* in particular is prominent, with so-called *relaxed plan heuristics* playing a major role in almost all successful satisficing (non-optimal) planning systems since more than 15 years (e. g. (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010)). The delete relaxation pretends that "what was once true will remain true forever", in the optimistic sense where, e. g., after driving from A to B a car is at both A and B simultaneously.

The downsides of this relaxation – being unable to account for to-and-fro, ignoring resource consumption– are evident, and it has been an active area from the outset to devise heuristic functions "taking some deletes into account" (e. g. (Fox and Long 2001; Helmert 2004; Keyder and Geffner 2008; Helmert and Geffner 2008; Cai, Hoffmann, and Helmert 2009; Coles et al. 2013)). Recent *partial delete relaxation* methods take this idea to the extreme: they allow, in principle, to force relaxed plans to behave like real plans in the limit, thus interpolating all the way between delete-relaxed planning and real planning. Two such methods are known, namely *explicit conjunctions* (Haslum 2012; Keyder, Hoffmann, and Haslum 2012; 2014), which forces relaxed plans to more accurately handle a given set C of conjunctions; and *red-black planning* (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013; Domshlak, Hoffmann, and Katz 2015), which delete-relaxes only a subset of the state variables (the "red" ones), keeping the original semantics of the other ("black") variables.

Alas, partial delete relaxation of course becomes costly as it approaches real planning, and a heuristic function computes a (partially delete-) relaxed plan anew for every search state. Hence the power to compute almost-real relaxed plans has been chained down by the computational overhead incurred. Explicit-conjunction heuristics are time-effective only with small C, and red-black planning heuristics have exclusively been based on tractable fragments. *Are there alternate ways of using partial delete relaxation, employing it more sparsely and hence unchaining its power? More generally, how to best employ accurate, information-rich yet computationally expensive, relaxations in planning?*

Focusing on red-black planning, we consider *(i) generating seed plans for plan repair*, and *(ii) proving planning task unsolvability*. Neither of these are, per se, new applications of relaxation. Research on the LPG system (Gerevini, Saetti, and Serina 2003), which conducts a local search in plan space, has already considered (i) the use of fully delete-relaxed plans as seed plans. Research on state-space abstractions has considered (ii) their use for proving unreachability as in Verification (Domshlak, Hoffmann, and Sabharwal 2009). The empirical results haven't been very promising for either. What is new in our work is the very different relaxation given by red-black planning.

Red-black planning is promising for (i) because red-black plans will be much closer to real plans, and hence a better starting point for plan-space search/plan repair. It is promising for (ii) because the red variables still carry the information "what needs to be done", while avoiding full enumeration across these variables. Consider, for example, a truck

with restricted fuel having to transport some packages. If we delete-relax ("paint red") the packages, they still need to be transported, to the effect that (i) any non-redundant red-black plan is a real plan, and (ii) if there is insufficient fuel then the red-black relaxation is unsolvable. (Contrast the latter with projections, recently suggested for proving unsolvability (Bäckström, Jonsson, and Ståhlberg 2013): projecting away the packages, the task becomes trivially solvable as there is no goal anymore.)

As prior work on red-black planning was concerned exclusively with tractable fragments, to realize our aims first of all we require a red-black planning method applicable to arbitrary inputs, i. e., arbitrary planning tasks and red/black "paintings" of its variables. This profile generalizes classical planning, and is **PSPACE**-complete, so a search is required. We introduce *red-black state space search*, which mixes standard forward state space search with standard delete-relaxed planning methods (Hoffmann and Nebel 2001), essentially by searching over black-variable states and augmenting each state transition with a delete-relaxed planning step over the red variables. If all variables are black, this defaults to forward search. If all variables are red, it defaults to delete-relaxed planning. In between, we have a hybrid. We evaluate the use of red-black plans generated this way for applications (i) and (ii). For (i), our results show promise, consistently improving the efficiency of plan repair as the number of black variables increases, and outperforming standard baseline planners (LPG, FD with h^{FF} and greedy best-first search) in several domains. For (ii), we obtain substantial, partly dramatic, improvements over the state of the art.

Preliminaries

Our approach is placed in the *finite-domain representation (FDR)* framework (initially introduced by Bäckström and Nebel (1995) under the less intuitive name "SAS$^+$"). An FDR *planning task* is a 4-tuple $\Pi = (V, A, I, G)$. V is a finite set of *state variables*, short *variables* v, each associated with a finite domain D_v. A complete assignment to V is a *state*. I is the *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*, each $a \in A$ being a triple $(\mathsf{pre}_a, \mathsf{eff}_a, c_a)$, where the *precondition* pre_a and the *effect* eff_a are partial assignments to V, and c_a is the action's real-valued *cost*. We often refer to (partial) assignments as sets of *facts*, i. e., variable-value pairs $v = d$. For a partial assignment p, $\mathcal{V}(p)$ denotes the subset of V instantiated by p. For $V' \subseteq \mathcal{V}(p)$, $p[V']$ denotes the value of V' in p.

The semantics of a planning task Π is defined in terms of its *state space*, which is a (labeled) *transition system*. Such a system is a 5-tuple $\Theta = (S, T, s_0, S_G)$, where S is a finite set of states, L is a finite set of *labels*, $T \subseteq S \times L \times S$ is a set of (labeled) *transitions*, $s_0 \in S$ is the *start state*, and $S_G \subseteq S$ is the set of *goal states*. We will usually write transitions $(s, l, s') \in T$ as $s \xrightarrow{l} s'$, or $s \rightarrow s'$ if the label does not matter. Assuming that the transition system is deterministic, i. e., for every state s and action a there exists at most one outgoing transition labeled with a, we define a *solution* for Θ to be a transition path from s_0 to a state in S_G.

The state space of a planning task Π is the transition system Θ_Π where: S is the set of all states in Π; the labels $L = A$ are the task's actions; $s \in S_G$ if $G \subseteq s$; and $s \xrightarrow{a} s'$ if a is *applicable* to s and s' is the *outcome state* of applying a to s. Here, a is applicable to s if $s[\mathcal{V}(\mathsf{pre}_a)] = \mathsf{pre}_a$, i. e., if $s[v] = \mathsf{pre}_a[v]$ for all $v \in \mathcal{V}(\mathsf{pre}_a)$. The outcome state, denoted $s[\![a]\!]$, is obtained by changing the value of $v \in \mathcal{V}(\mathsf{eff}_a)$ to $\mathsf{eff}_a[v]$. A solution for Θ_Π is called a *plan* for Π.

Red-Black Planning

The delete relaxation, originally defined for STRIPS planning, can be captured in FDR in terms of state variables that accumulate, rather than switch between, their values. Red-black planning is the partial delete relaxation resulting from doing so only for a subset of the state variables (the "red" ones), keeping the original value-switching semantics for the others (the "black" ones) (Katz, Hoffmann, and Domshlak 2013b; Domshlak, Hoffmann, and Katz 2015).

Formally, a *red-black planning task*, short *RB planning task*, is a 5-tuple $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$. Here, V^{B} are the black variables, and V^{R} are the red ones. We require that $V^{\mathsf{B}} \cap V^{\mathsf{R}} = \emptyset$, and given the overall set of variables $V := V^{\mathsf{B}} \cup V^{\mathsf{R}}$, the remainder of the task syntax is defined exactly as before. The major change lies in the semantics, i. e., the definition of the state space Θ_Π.[1]

We refer to variable/value pairs over V^{B} as *black facts*, and to variable/value pairs over V^{R} as *red facts*. A *red-black state*, short *RB state* and denoted s^{RB}, assigns non-empty value subsets, rather than values, to the state variables, where $|s^{\mathsf{RB}}(v)| = 1$ for $v \in V^{\mathsf{B}}$. An action a is applicable in s^{RB} if $\mathsf{pre}_a[v] \in s^{\mathsf{RB}}[v]$ for all $v \in \mathcal{V}(\mathsf{pre}_a)$. Applying a in s^{RB} changes the value of $v \in \mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{B}}$ to $\{\mathsf{eff}_a[v]\}$, and changes the value of $v \in \mathcal{V}(\mathsf{eff}_a) \cap V^{\mathsf{R}}$ to $s^{\mathsf{RB}}[v] \cup \{\mathsf{eff}_a[v]\}$. The outcome state is denoted $s^{\mathsf{RB}}[\![a]\!]$.

The *red-black initial state*, short *RB initial state* and denoted s_0^{RB}, is the one where every variable contains just its initial value as prescribed by I. *Red-black goal states*, short *RB goal states*, are those s^{RB} where $G[v] \in s^{\mathsf{RB}}[v]$ for all $v \in \mathcal{V}(G)$. A solution for Θ_Π is called a *red-black plan*, short *RB plan*, for Π.

Given an FDR task $\Pi = (V, A, I, G)$, a *painting* is a partition of the variables V into two subsets, V^{B} and V^{R}. Given a painting, a plan for the red-black planning task $(V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ is called a red-black plan for Π.

Example 1. *As a running example, we consider the IPC NoMystery domain, where a truck has to transport packages subject to fuel restrictions. We will be using simple examples with two packages and locations, as illustrated in Figure 1.*

Our examples will differ only in the initial amount of fuel. The FDR encoding for all examples has four state variables: the truck position $t \in \{A, B\}$, the current amount of fuel $f \in \{0, 1, 2\}$, and the positions of the packages $p_1, p_2 \in \{T, A, B\}$ (T meaning that the package is currently in the truck). There are actions to $drive$ the truck

[1] We intentionally do not refer to Θ_Π as the "red-black state space", because we will use that name for the transition system introduced in the next section, which branches only over black state transitions. The definition of Θ_Π here is impractical, and is relevant only to the semantics definition.

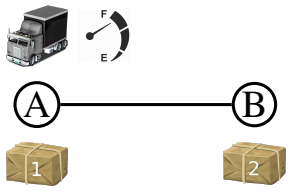

Figure 1: Running example: transportation with limited fuel.

from location X to location Y, assuming remaining fuel Z, and consuming one fuel unit; to load package X at location Y; and to unload package X at location Y. For example, $drive(A, B, 2)$ has precondition $\{t = A, f = 2\}$ and effect $\{t = B, f = 1\}$. In the initial state, shown in Figure 1, the truck and p_1 are at A, and p_2 is at B. The goal is to swap the positions of the packages, i. e., $G = \{p_1 = B, p_2 = A\}$. Throughout, we will use the painting $V^{\mathsf{B}} = \{t, f\}$, $V^{\mathsf{R}} = \{p_1, p_2\}$, i. e., truck and fuel are black, packages are red.

For an initial fuel amount of $f = 2$, an RB plan is $\langle load(p_1, A), drive(A, B, 2), unload(p_1, B), load(p_2, B), drive(B, A, 1), unload(p_2, A)\rangle$. Note that this RB plan does not "cheat" – it is a plan for the original input plan Π. For t and f, this is necessarily so as they are painted black. For the packages, the red-black plan could in principle insert invalid additional actions (like loading p_1 twice at A), but a non-redundant red-black plan won't do that.[2]

Say now that instead $f = 1$ in the initial state. Then, obviously, a real plan for the task does not exist. Remarkably, a red-black plan does not exist either: (a) the package goals still obligate the truck to be at A, B, A in this sequence, while (b) this cannot be done given that both t and f are black. Note here the contrast to projection onto the black variables t and f (Bäckström, Jonsson, and Ståhlberg 2013) which foregoes (a), and delete-relaxed planning/$h^{\max}$ (Bonet and Geffner 2001; Hoffmann and Nebel 2001) which foregoes (b): these known methods cannot detect the dead-end here. Combining their virtues, red-black planning can.

If $V^{\mathsf{B}} = V$, then the red-black plans for Π coincide with the (real) plans for Π. If $V^{\mathsf{R}} = V$, then the red-black plans for Π coincide with the (fully) delete-relaxed ones. In this sense, red-black planning is a partial delete relaxation method, allowing to interpolate between delete-relaxed and real planning. Yet, as outlined in the introduction, prior work hardly made use of that power, limiting itself to tractable fragments of red-black planning (more precisely, tractable fragments for red-black plan generation). We next introduce a generally applicable red-black planning method, allowing to handle arbitrary input FDR tasks and paintings.

Red-Black State Space Search

In principle, one could of course generate red-black plans simply by search in the state space of a red-black planning task, as defined above. This would, however, not benefit much from the monotonic behavior of red variables. In particular, if all variables are red, then this would be a search in the state space over delete-relaxed actions, as opposed to the known efficient algorithm for that case, which just performs a forward fixed point operation applying *all* actions in parallel layers (Blum and Furst 1997; Hoffmann and Nebel 2001). Our key observation here is that the latter algorithm can be combined with state space search into a hybrid, *red-black state space search*, which branches only over those actions affecting black variables, while handling the other actions through red forward fixed points associated with individual state transitions.

The *forward phase* of red-black state space search chains forward until reaching the goal (for readers familiar with relaxed planning: "state space search with a relaxed planning graph at each transition"), at which point a *backward phase* extracts a red-black plan ("extracting the solution path with a relaxed plan extraction step at each transition"). In what follows, we specify first the former, then the latter. We conclude the section by examining the algorithm's runtime behavior, as a function of the fraction of black variables.

Forward Phase: States & Transitions

As stated, we distinguish two kinds of actions, those affecting black variables vs. those that don't. More precisely actually, our definition of *red actions* is dependent on the RB state s^{RB} in question, collecting all those whose black effects do not affect *that state*. These are exactly the actions we do not need to branch over in s^{RB}. We project these actions onto the red variables, for the purpose of the relaxed (the "red") fixed point in s^{RB}:

Definition 1 (Red Actions)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task, and let s^{RB} be an RB state. The set of* red actions *in s^{RB}, denoted $A^{\mathsf{R}}(s^{\mathsf{RB}})$, contains an action a^{R} for every $a \in A$ where $\mathsf{pre}_a[V^{\mathsf{B}}] \subseteq s^{\mathsf{RB}}$ and $\mathsf{eff}_a[V^{\mathsf{B}}] \subseteq s^{\mathsf{RB}}$. a^{R} is defined as $\mathsf{pre}_{a^{\mathsf{R}}} := \mathsf{pre}_a[V^{\mathsf{R}}]$ and $\mathsf{eff}_{a^{\mathsf{R}}} := \mathsf{eff}_a[V^{\mathsf{R}}]$.*

The red forward fixed point in s^{RB}, and the red plan extraction later on, are easiest to formulate via a fully delete-relaxed planning task. We formalize this here as a red-black planning task with red variables only:

Definition 2 (Red Relaxation)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. Let s^{RB} be an RB state, and let g be a set of red facts. The* red relaxation *of s^{RB} given g, denoted $\Pi^+(s^{\mathsf{RB}}, g)$, is the RB task $\Pi^+(s^{\mathsf{RB}}, g) := (\emptyset, V^{\mathsf{R}}, A^{\mathsf{R}}(s^{\mathsf{RB}}), s^{\mathsf{RB}}[V^{\mathsf{R}}], g)$. If the set g is not relevant, we will just use $\Pi^+(s^{\mathsf{RB}})$.*

This task is defined as one would expect, using the red variables V^{R} only, using the red actions $A^{\mathsf{R}}(s^{\mathsf{RB}})$ as per above, using the red facts $s^{\mathsf{RB}}[V^{\mathsf{R}}]$ of s^{RB} as the initial state. The set g here will play a role only during the red plan extraction step later on (it will be the red-fact subgoals propagated into this step from the postfix of the red-black plan extraction). For the forward step, we are interested merely in what red facts are reachable given $\Pi^+(s^{\mathsf{RB}})$ ("the relaxed planning graph fixed point layer"). It will be convenient to perceive these red facts as an extension of the RB state s^{RB}

[2] Indeed, in our painting here, only "causal graph leaf variables" are painted red, in which case every non-rendundant red-black plan is guaranteed to be a real plan (Katz, Hoffmann, and Domshlak 2013b). This is not so in general of course. Our examples here are merely intended as simple illustrations of our main points.

we started out with, enriching s^{RB} with the additional red facts reachable when applying red actions only:

Definition 3 (Red Fixed Point)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task, and let s^{RB} be an RB state. The* red fixed point *of s^{RB}, denoted $\mathcal{F}^+(s^{\mathsf{RB}})$, is the RB state t^{RB} where $t^{\mathsf{RB}}[V^{\mathsf{B}}] = s^{\mathsf{RB}}[V^{\mathsf{B}}]$, and $t^{\mathsf{RB}}[V^{\mathsf{R}}]$ is the set of all red facts reachable from $s^{\mathsf{RB}}[V^{\mathsf{R}}]$ in $\Pi^+(s^{\mathsf{RB}})$.*

The red-black state space can now be specified very easily, as a labeled transition system interleaving black-variable transitions with red-variable fixed points:

Definition 4 (RB State Space)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. The* red-black state space *of Π, short* RB state space *and denoted Θ^{RB}_Π, is the transition system $\Theta^{\mathsf{RB}} = (S^{\mathsf{RB}}, T^{\mathsf{RB}}, s^{\mathsf{RB}}_0, S^{\mathsf{RB}}_G)$ where:*

- *S^{RB} is the set of all RB states.*
- *s^{RB}_0 is the RB initial state.*
- *S^{RB}_G contains the RB states s^{RB} where $\mathcal{F}^+(s^{\mathsf{RB}})$ is an RB goal state.*
- *T^{RB} is the set of transitions $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$ where $\mathsf{eff}(a)[V^{\mathsf{B}}] \not\subseteq s^{\mathsf{RB}}[V^{\mathsf{B}}]$, a is applicable to $\mathcal{F}^+(s^{\mathsf{RB}})$, and $t^{\mathsf{RB}} = \mathcal{F}^+(s^{\mathsf{RB}})[\![a]\!]$.*

Note that this definition is modular, in the sense that it is independent of how exactly red fixed points are being computed. Standard methods, like *relaxed planning graphs* (Hoffmann and Nebel 2001), take low-order polynomial runtime. To illustrate the definition, consider again our running example:

Example 2. *We consider first the solvable variant where we have two fuel units, $f = 2$, in the initial state. Figure 2 shows part of the red-black state space.*

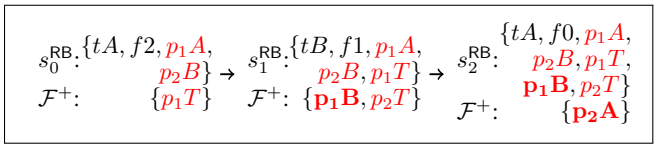

Figure 2: Part of the RB state space of our solvable running example. Variable-value pairs $v = d$ (facts) are abbreviated by omitting the "=". Black facts shown in black, red facts in red, goal facts in **boldface**. $\mathcal{F}^+$ gives only those red facts not already contained in the state itself.

We show here the part of the RB state space corresponding to the solution path – the solution of the labeled transition system Θ^{RB}_Π – given by $\langle s^{\mathsf{RB}}_0, drive(A,B,2), s^{\mathsf{RB}}_1, drive(B,A,1), s^{\mathsf{RB}}_2\rangle$. Observe how, in the transitions, e. g. $s^{\mathsf{RB}}_0 \xrightarrow{drive(A,B,2)} s^{\mathsf{RB}}_1$, the target state ($s^{\mathsf{RB}}_1$) accumulates the red fixed point of the source state (s^{RB}_0), and then in its own red fixed point accumulates the now additionally reachable red facts. For s^{RB}_1, the latter facts are $p_1 = B$ and $p_2 = T$, which were not reachable in s^{RB}_0 as they rely on the black precondition $t = B$. Note that the solution path does not specify what exactly to do about the red variables; this is the task of the backward step described in the next subsection.

Importantly, in each of s^{RB}_0 and s^{RB}_1, the action we apply labels the only *outgoing transition in Θ^{RB}_Π: the only black-affecting actions are the $drive$ ones, and these are the only applicable such actions. In other words,* Figure 2 actually shows the entire RB state space up to the first goal state encountered. *The same phenomenon leads to an even more compact RB state space in the unsolvable example variant, depicted in Figure 3.*

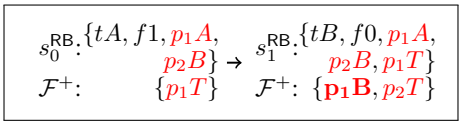

Figure 3: The (entire) RB state space of our unsolvable running example. Notations as in Figure 2.

It is easy to see that the red-black state space preserves red-black plans, in the following sense:

Theorem 1 (Completeness)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. Let $\pi = \langle a_0, \dots, a_{n-1}\rangle$ be an RB plan for Π traversing RB states $s^{\mathsf{RB}}_0, \dots, s^{\mathsf{RB}}_n$. Let π^B be the subsequence of a_i in π where $\mathsf{eff}_{a_i}[V^{\mathsf{B}}] \not\subseteq s^{\mathsf{RB}}_i[V^{\mathsf{B}}]$. Then π^B labels a solution for Θ^{RB}.*

This is true simply because the red-black state space branches over all actions that change the black part of the state, while the red fixed point in each state captures everything that can be done without changing that part of the state.

Note that the red-black state space is a labeled transition system, just like the standard FDR state space. To find solutions, we can apply any search algorithm. Indeed, our implementation in Fast Downward (FD) (Helmert 2006) merely exchanges the state and state transition data structures, preserving all search algorithms. We can also use any classical-planning heuristic function that can be defined on red-black states s^{RB}. In our implementation, we realized this for the standard delete-relaxed plan heuristic h^{FF} (Hoffmann and Nebel 2001), simply by applying h^{FF} to the delete-relaxed initial state containing all facts (red or black) true in s^{RB}.

If all variables are black, then the RB state space is just the standard state space, i. e., *for the special case of fully un-relaxed planning, the RB state space defaults to standard forward search.* More importantly, if all variables are red, then the RB state space has a single state only, whose red fixed point contains exactly those facts delete-relaxed reachable from the task's initial state. In other words, as desired, *for the special case of fully delete-relaxed planning, the RB state space defaults to a standard polynomial-time delete-relaxed forward reachability check.* In between, we obtain a hybrid between these two. In particular, if there are red variables, then an additional RB plan extraction step is needed.

Backward Phase: Red-Black Plan Extraction

To extract an RB plan, where the forward phase built a relaxed planning graph at every transition $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$, we now perform a corresponding relaxed plan extraction step. The only difference to standard relaxed plan extraction lies in the "initial state" and "goal" of this relaxed plan extraction. These are instantiated here with the red-black source

state s^{RB} of the transition, respectively with the red subgoals propagated to the transition's target state t^{RB}, from the previous relaxed plan extraction steps at the postfix behind $s^{\mathsf{RB}} \xrightarrow{a} t^{\mathsf{RB}}$.

To formalize this in a modular way – independently of how exactly relaxed plans are being extracted – we specify in what follows only the relaxed planning tasks to be solved, not how they are being solved. Standard methods for the latter are readily available (e. g. (Hoffmann and Nebel 2001; Keyder and Geffner 2008)). To specify the propagation of red subgoals without an explicit relaxed/red plan extraction mechanism, we rely on regression: new red subgoals will result from previous ones by regression over the red plan.

Definition 5 (Red Regression)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. Let g be a set of red facts, and let $a \in A$ be an action. The* red regression *of g over a is defined as $\mathsf{Regress}^{\mathsf{R}}(g, a) := \mathsf{pre}_a[V^{\mathsf{R}}] \cup (g \setminus \mathsf{eff}_a[V^{\mathsf{R}}])$. The red regression of g by a sequence of actions $a_1, \dots, a_n \in A^n$ is recursively defined as $\mathsf{Regress}^{\mathsf{R}}(g, \langle a_1, \dots, a_n \rangle) := \mathsf{Regress}^{\mathsf{R}}(\mathsf{Regress}^{\mathsf{R}}(g, a_n), \langle a_1, \dots, a_{n-1} \rangle)$.*

Red-black plan extraction can now conveniently be specified in a recursive manner, assuming in each step that the postfix has already been dealt with, and pre-pending the plan extracted so far with an additional red plan extraction step:

Definition 6 (RB Plan Extraction)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. Let $\pi = \langle s_0^{\mathsf{RB}}, a_0, s_1^{\mathsf{RB}}, \dots, a_{n-1}, s_n^{\mathsf{RB}} \rangle$ be a solution for Θ_Π^{RB}. The* red-black plan *extracted from π, short* RB plan *and denoted π^{RB}, is constructed as follows:*

- *(ii) If $n = 0$, then π^{RB} is defined as a plan for the task $\Pi^+(s_0^{\mathsf{RB}}, G[V^{\mathsf{R}}])$.*
- *(iiii) Otherwise, let π_1^{RB} be the RB plan for the postfix of π, i. e., for $\langle s_1^{\mathsf{RB}}, a_1, \dots, a_{n-1}, s_n^{\mathsf{RB}} \rangle$. Let $g := \mathsf{Regress}^{\mathsf{R}}(G[V^{\mathsf{R}}], \langle a_0 \rangle \circ \pi_1^{\mathsf{RB}})$ be the regression of the red goal facts over the concatenation of a_0 with π_1^{RB}. Finally, let π_0^+ be a plan for the task $\Pi^+(s_0^{\mathsf{RB}}, g)$. Then π^{RB} is defined as $\pi_0^+ \circ \langle a_0 \rangle \circ \pi_1^{\mathsf{RB}}$.*

To illustrate red-black plan extraction, consider the solvable variant of our running example:

Example 3. *Reconsider the RB state space shown in Figure 2, and the solution path $\pi = \langle s_0^{\mathsf{RB}}, drive(A, B, 2), s_1^{\mathsf{RB}}, drive(B, A, 1), s_2^{\mathsf{RB}} \rangle$. The recursion in Definition 6 continues until $n = 0$, i. e., until we consider just the trivial postfix $\langle s_2^{\mathsf{RB}} \rangle$. At this point, case (i) applies and we merely have to construct a plan for $\Pi^+(s_2^{\mathsf{RB}}, G[V^{\mathsf{R}}])$, i. e., the red-black planning task which considers only the red package position variables, starts from the red facts in s_2^{RB}, and whose goal is the original goal $\{p_1 = B, p_2 = A\}$. Say we extract the (obvious/only optimal) plan for $\Pi^+(s_2^{\mathsf{RB}}, G[V^{\mathsf{R}}])$, $\langle unload(p_2, A) \rangle$.*

Going back up in the recursion, we consider next the last transition on π, in the form of the postfix $\langle s_1^{\mathsf{RB}}, drive(B, A, 1), s_2^{\mathsf{RB}} \rangle$. Definition 6 case (ii) applies. In the notation of that case, we have $\pi_1^{\mathsf{RB}} = \langle unload(p_2, A) \rangle$, and we have $g = \mathsf{Regress}^{\mathsf{R}}(G[V^{\mathsf{R}}], \langle drive(B, A, 1) \rangle \circ \langle unload(p_2, A) \rangle) = \{p_2 = T, p_1 = B\}$. In other words, the red-black plan postfix is to unload p_2 at A, and the corresponding subgoal propagated here is for p_2 to be in the truck, plus the other top-level goal $p_1 = B$, which has not been addressed in the postfix so far. We need to extract a plan for the task starting from the red facts in s_1^{RB} and achieving this subgoal. Say we extract the plan $\langle load(p_2, B), unload(p_1, B) \rangle$. Then at this point our red-black plan postfix is $\langle load(p_2, B), unload(p_1, B) \rangle \circ \langle drive(B, A, 1) \rangle \circ \langle unload(p_2, A) \rangle$.

The next step of the recursion considers the whole path π, with π_1^{RB} as just specified, and with the regressed red subgoal $g = \{p_2 = B, p_1 = T\}$. Starting from the red facts in s_0^{RB}, this subgoal can be achieved by $\langle load(p_1, A) \rangle$. Thus the final red-black plan π^{RB} constructed is $\langle load(p_1, A), drive(A, B, 2), load(p_2, B), unload(p_1, B), drive(B, A, 1), unload(p_2, A) \rangle$.

By a simple inductive argument using the same recursion as Definition 6, we get soundness:

Theorem 2 (Soundness)**.** *Let $\Pi = (V^{\mathsf{B}}, V^{\mathsf{R}}, A, I, G)$ be an RB planning task. Let π be a solution for Θ^{RB}, and let π^{RB} be the RB plan extracted from π. Then π^{RB} is an RB plan for Π.*

Performance Profile

Red-black state space search allows to interpolate between (easy) delete-relaxed planning and (hard) real planning. *But how many variables can we paint black without making that search prohibitively costly to conduct?* The answer is: typically, more than you would expect. Have a look at Figure 4.

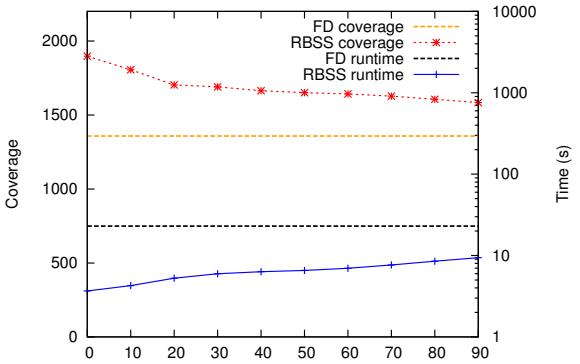

Figure 4: Coverage and average runtime of red-black state space search, compared to Fast Downward, as a function of the fraction of black variables. Explanations see text.

The "FD" part of Figure 4 refers to the canonical configuration of Fast Downward for satisficing planning, i. e., greedy best-first dual-queue search with h^{FF} and preferred operators. For red-black state space search, we use the exact same search algorithm and implementation, based on our aforementioned adaptation of the underlying state-space data structures and h^{FF}. Thus the *only* difference between the configurations tested are the red variables.

We run all satisficing-planning STRIPS benchmarks from the International Planning Competitions (IPC) 1998–2014, as well as the resource-constrained benchmarks of Nakhost et al. (2012), for a total of 1897 benchmark instances. We

use a cluster of Intel E5-2660 machines running at 2.20 GHz, with a runtime limit of 30 minutes, and a memory limit of 4 GB (these same machines and limits will be used throughout). The x-axis in Figure 4 shows the fraction of black variables, i. e. $|V^{\mathsf{B}}|/|V|$, in percent. The underlying paintings are obtained by fixing an ordering of the variables – namely the "SCC-DFS" ordering introduced in the next section – and painting the first $x\%$ of the variables black. The runtime shown is the geometric average over the benchmark instances solved by at least one configuration (i. e., either by FD, or by red-black state space search for some value of x), and solving which takes non-trivial runtime (> 1 second) for at least one configuration. For unsolved instances, we insert the time-out of 1800 seconds into the average.

As Figure 4 shows quite clearly, we can typically paint a large fraction of variables black while still being able to solve the red-black task quickly. Even if we paint 90% of the variables black, the average runtime is still less than half that of the fully black (FD) search. We do lose coverage substantially as x grows, from the full 1897 instances for $x = 0$ to 1584 instances for $x = 90$, but we consistently remain far above the 1358 instances solved by FD.

In summary, it is feasible to obtain highly informed red-black plans. We next proceed to exploring two possible uses of such plans, namely *(i) generating seed plans for plan repair*, and *(ii) proving planning task unsolvability*. We consider these applications in this order.

Red-Black Plan Repair

The idea here is to generate a red-black plan π^{RB}, then hand π^{RB} over to a plan repair method to turn it into a real plan. We baptize this approach *red-black plan repair*.

There are two basic design decisions to be made, namely how to obtain the painting, and which plan repair mechanism to use. Regarding the former, the fraction of black variables encompasses a trade-off between the effort for red-black state space search vs. that for plan repair. A natural choice thus is to make the fraction of black variables an input parameter, which we will denote by F^B.

The question of choosing a painting now boils down to choosing the given fraction of most preferred variables. Following Domshlak et al.'s (2015) work on painting strategies for red-black planning heuristic functions, we formulate such preferences in terms of *variable orderings*. The $\lfloor F^B * |V| \rfloor$ first variables will be painted black.

It remains to specify the variable ordering. Intuitively, variables "close to the causal graph root" should preferrably be black. This suggests to use the *level* ordering, originally introduced by Helmert (2004) and also used by Domshlak et al. (2015). On the other hand though, intuitively the plan repair mechanism will have an easier task if the part of the problem it still needs to tackle is "coherent", i. e., if the painting does not put tightly linked parts of the problem on different sides of the red/black border. To illustrate with an extreme case: if the input task has two completely independent parts, then intuitively each part should be completely black or completely red, rather than a combination thereof. We hence design a variant of the level ordering, where the causal graph strongly connected components (SCC) are assigned increasing levels from root SCCs to leaf SCCs. Starting at a level-0 SCC, we include successor SCCs in a depth-first manner. We will refer to this ordering as *SCC-DFS*.

For repairing the red-black plan, we considered two methods. First, the aforementioned LPG planner – local search in the space of partial plans – run in plan-adaptation mode (Fox et al. 2006; Gerevini, Saetti, and Serina 2003). Second, the plan-adaptation system ADJ (Gerevini and Serina 2000; 2010), which revises a flawed input plan by replacing incrementally growing flawed plan portions, called "replanning windows", by new (sub)plans having incrementally bounded lengths.[3] We will refer to the first of these methods as *LPG-RB*, and to the second one as *ADJ-RB*.

As before, we run all IPC satisficing-planning STRIPS benchmarks, and Nakhost et al.'s (2012) resource-constrained benchmarks. Figure 5 gives an overview of performance for LPG-RB, aggregated over these benchmarks.

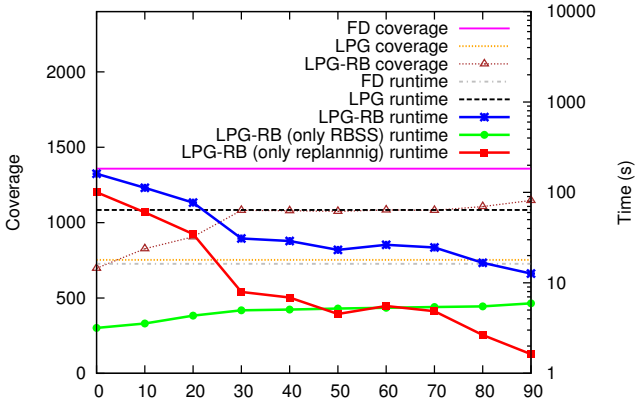

Figure 5: Coverage and average runtime of red-black plan repair with LPG-RB, compared to the FD and LPG baselines, as a function of the fraction of black variables. Explanations see text.

FD in Figure 5 is the same canonical satisficing configuration as before, LPG is the default configuration starting from an empty seed plan. Like in Figure 4, the runtime averages are geometric, and are taken over those instances solved by at least one configuration, and solving which takes non-trivial runtime for at least one configuration. The time-out of 1800 seconds is inserted in unsolved cases.

In terms of overall coverage, FD is best by far. We will see below that this is not so in some individual domains. LPG-RB is slightly worse than LPG for $x = 0$, reflecting a known observation about LPG, namely that seeding it with a fully delete-relaxed plan tends to result in worse performance than using an empty/no seed plan. However, this changes as we increase the number of black variables, showing that, as expected, using red-black plans is better for plan repair. In particular, LPG-RB is much more effective than LPG. Compared to FD (whose runtime is the horizontal line directly below that for LPG coverage), LPG-RB's runtime is clearly worse up to $x = 70$, and thereafter is marginally better.

[3]The (sub)plans of ADJ can be generated using any base planner. In our experiments here, we use the default incorporated planner Metric-FF (Hoffmann 2003).

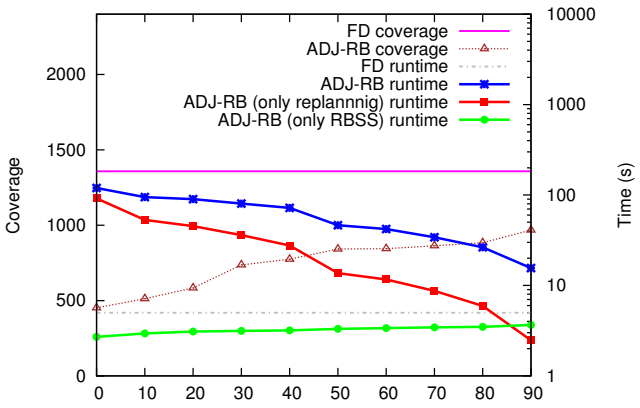

Figure 6: Overview data like Figure 5, for ADJ-RB.

Figure 6 shows the data for ADJ-RB. All settings are as in Figure 5. The overall conclusions are similar as for LPG-RB. The FD baseline dominates overall performance. ADJ-RB profits substantially and consistently, in both coverage and runtime, from a growing fraction of black variables.

There is, of course, a lot of variance across domains. *In particular, in several domains, performance over the baseline(s) is strictly, sometimes vastly, improved.*[4]

For space reasons, we don't include detailed data for our 55 test suites. For LPG-RB, in 13 test suites coverage becomes worse relative to LPG; in 6 suites, coverage is not affected; in 13 suites, coverage increases but remains worse than that of FD; in 15 suites, coverage increases up to that of FD; and in 7 suites, coverage for LPG-RB exceeds that of both baselines. The latter suites are Airport (coverage LPG 35, coverage for the best LPG-RB configuration 46, coverage FD 36); 3 NoMystery suites (overall LPG 9, best LPG-RB 228, FD 103); 2 resource-constrained Rovers suites (overall LPG 13, best LPG-RB 60, FD 33); and Tidybot (LPG 10, best LPG-RB 17, FD 15).

For ADJ-RB, in 9 suites coverage becomes worse as x grows; in 13 suites, it is not affected; in 14 suites, it increases but remains worse than that of FD; in 11 suites, it increases up to that of FD; in 8 suites, coverage for ADJ-RB exceeds that of FD. The latter suites are Childsnack (best ADJ-RB 4, FD 3); 3 NoMystery suites (overall best ADJ-RB 228, FD 103); Pathways (best ADJ-RB 21, FD 20); 2 resource-constrained Rovers suites (overall best ADJ-RB 60, FD 33); and Tidybot (best ADJ-RB 19, FD 15).

In resource-constrained NoMystery, LPG-RB and ADJ-RB are state of the art. Together with *decoupled search* (Gnad and Hoffmann 2015a; 2015b), they are the only planners that can solve all 175 instances of the "small" test suite. The previously best planner, as per Nakhost et al.'s (2012) extensive experiments, was FD-AutoTune which solves 152.

[4]Note also that we are somewhat "abusing" plan repair mechanisms here. LPG in adaptation mode, and ADJ, are designed and intended to be run in an execution & monitoring setting, where failed plans may require repair, and these repairs can be expected to be small, and/or the plans satisfy assumptions ("plan stability" (Fox et al. 2006) or the "regular world" assumption" (Borrajo et al. 2015)) not considered here. In such settings, plan repair has been shown to be much more efficient than planning from scratch.

Proving Unsolvability

We now turn our attention to proving the input planning task Π unsolvable, *within* the red-black relaxation, i. e., by fully building the red-black state space Θ^{RB} for a suitable painting. If there is no solution for Θ^{RB}, then by Theorem 1 there is no red-black plan, hence no real plan for Π either. As illustrated in Example 2, this is promising because painting variables red preserves their need to reach their goals, yet allows to avoid enumerating their value combinations.

But what is a "suitable painting"? Similarly as before, we simplify this question here, to a decision about in which order to paint variables black. Instead of fixing a ratio of black variables, however, here we can use an *incremental* approach. We start with the empty set V^{B} of black variables. We then build Θ^{RB}. We terminate if Θ^{RB} has no solution. Otherwise, we add the next variable into V^{B}, and iterate. This way, we don't need to guess a priori how many black variables we will need. As red-black tasks with few black variables are typically solved quickly, the overhead from unsuccessful iterations can be expected to be small – provided of course that Π can be proved unsolvable with few black variables. But if this is not so, then red-black state space search is not a good approach anyhow.

So, how to order the variables? On Π that *can* be proved unsolvable with few black variables, how to find those variables? Intuitively, as before, variables "close to the causal graph root" should preferrably be black. In difference to red-black plan repair, there is no second solver to which we want to leave a coherent part of the problem, so instead of a depth-first strategy we here use a breadth-first one, *SCC-BFS*, which paints SCCs black by increasing level.

We also explore an alternative, more "conflict-directed", source of guidance, taking inspiration from the "C" painting strategy Domshlak et al. (2015) design for their heuristic function. That strategy considers a fully delete-relaxed plan π^+, and counts for each variable v the number of action preconditions that would be violated along π^+ when painting v black. Variables with a higher number of such *conflicts* are painted black. We adopt this here by using, instead of a fully delete-relaxed plan, the red-black plan π^{RB} found in the previous iteration of the incremental process: the next variable is one with the maximal number of conflicts in π^{RB}. This provides a more dynamic form of guidance, honing in on the "holes" left over by the previous insufficient painting. We refer to this strategy as *Conf*. We also experiment with a simple combined strategy *SCC-BFS-Conf*, which breaks ties in SCC-BFS, within each SCC, by the number of conflicts.

We use the suite of unsolvable benchmarks underlying the current state-of-the-art experiments by Hoffmann et al. (2014). Table 1 shows coverage data, i. e., the number of instances proved unsolvable.

Our three painting strategies are shown as "RBb" for SCC-BFS, "RBc" for Conf, and "RBbc" for SCC-BFS-Conf. We compare against a selection of approaches from Hoffmann et al.'s (2014) extensive experiments, namely blind search ("Bli") and search with $h^{\max}$ as canonical simple methods; exhaustive testing of small projections ("SP") as per Bäckström et al. (2013) to compare against this recently proposed method; constrained BDDs (Torralba and

Domain	#	Bli	$h^{\max}$	SP	BDD	MS1	MS2	RBb	RBc	RBbc	BP	DS
Bottleneck	25	10	21	10	15	10	21	12	**25**	**25**	5	0
3UNSAT	30	**15**	**15**	0	**15**	**15**	**15**	**15**	**15**	**15**	5	0
Mystery	9	2	2	6	**9**	**9**	6	7	2	2	0	0
NoMystery	25	0	0	8	14	**25**	**25**	24	24	24	14	24
PegSol	24	**24**	**24**	0	**24**	**24**	**24**	12	22	22	8	0
Rovers	25	0	1	3	10	17	9	**25**	11	**25**	0	0
Tiles	20	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	0
TPP	25	5	5	2	1	**9**	**9**	2	1	1	0	0
$\sum$	183	66	78	39	98	119	119	107	110	**124**	42	24

Table 1: Number of instances proved unsolvable. Best values highlighted in **boldface**. Left part: state of the art as per Hoffmann et al. (2014). Middle part: red-black state space search. Right part: particular comparisons. Explanations and abbreviations see text.

Alcázar 2013) ("BDD") as a competitive symbolic method (named "BDD H^2 in (Hoffmann, Kissmann, and Torralba 2014)); as well as the two most competitive variants of merge-and-shrink by Hoffmann et al., namely their "Own+A H^2" (here: "MS1") and their "Own+K N100k M100k $h^{\max}$" (here: "MS2"). This selection of planners represents the state of the art in proving unsolvability in planning.

Our best configuration, RBbc, beats the state of the art in overall coverage. It excels in Bottleneck and Rovers, where red-black state space search is the only method able to solve all instances. In NoMystery, together with merge-and-shrink and DS (regarding which: see below), it performs way better than all other planners. In the remaining domains, the performance of red-black state space search is not as remarkable, about in the mid-range in Mystery, PegSol, and TPP, and on par with other planners in 3UNSAT and Tiles where no planner seems to manage to do something interesting.

The "BP" and "DS" columns stand for *black-projection*, respectively *decoupled search* (Gnad and Hoffmann 2015a; 2015b). BP is like our incremental RBbc method but considering the black variables only. It follows RBbc's variable ordering, SCC-BFS-Conf, until RBbc terminates; if the projection onto the black variables is at this point still solvable, then BP continues with the SCC-BFS variable ordering. BP has not been previously explored, and is included here to show the benefit of considering red variables in addition to the black ones. The data clearly attests to that benefit.

DS identifies a partition of the variables inducing a "star topology", then searches only over the "center" component of the star, enumerating the possible moves for each "leaf" component separately. We include it here because, like red-black state space search, it can avoid the enumeration across packages in NoMystery (each package is a leaf component). DS is, however, limited to tasks with a useful star topology that can be identified with the current variable partitioning methods. The latter is rare on this benchmark set, and the data clearly shows the benefit of not having that limitation.

In Rovers, red-black state space search clearly outperforms all competitors even when looking at coverage only. In its other strongest domain, Bottleneck, the coverage differences are less pronounced. Figure 7 shows runtime data to point out the substantial runtime advantage. (The behavior of MS2 arises from having similar pruning power as $h^{\max}$ on its own, but incurring a large runtime overhead from building the merge-and-shrink abstraction.) We remark that, in NoMystery, all RB variants as well as DS have coverage close to MS, yet lag far behind in runtime.

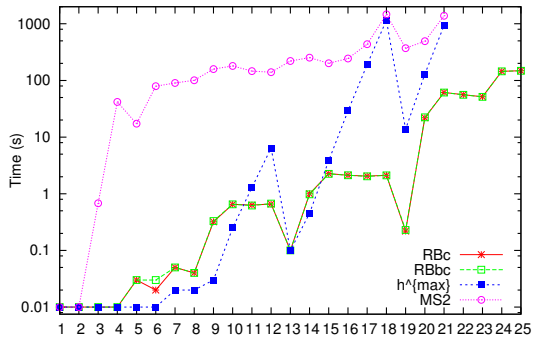

Figure 7: Runtime of the strongest planners in Bottleneck.

Consider finally Figure 8, a direct comparison between red-black state space search and black-projection, as the set of black variables V^{B} grows. This provides an in-depth view of the advantages of taking into account the remaining variables $V \setminus V^{\mathsf{B}}$ as red ones, rather than ignoring them completely. The coverage advantage is dramatic, as red-black state space search can make do with *much* smaller sets V^{B}. The runtime averages are taken over the commonly solved instances for each value of x. We see that, as expected, red-black state space search incurs a substantial overhead for those tasks tackled also by projection with small V^{B}. Yet as the V^{B} required in projection grows larger, that disadvantage becomes smaller and finally disappears completely.

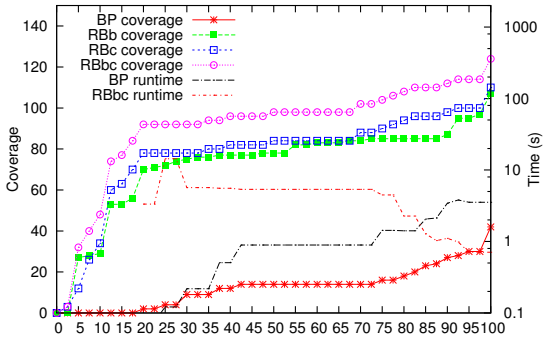

Figure 8: Coverage and average runtime of red-black state space search, compared to black-projection, as a function of the fraction of black variables. Explanations see text.

Conclusion

Partial delete relaxation is powerful, yet has been chained to the ground by its use in heuristic functions. Our investigation shows that other, intractable, applications may be very useful. Red-black plan repair still requires further research; promising options are, e. g., reordering steps to reduce conflicts, or using only parts (applicable prefixes, only black actions, . . .) of the red-black plan. An interesting alternative is to keep painting variables black until a real plan is found, exploiting in each iteration the information from the previous one. Further directions are the generation of preferred operators, meta-heuristics intelligently selecting which method to use (à la (Domshlak, Karpas, and Markovitch 2012)), etc.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, "Critically Constrained Planning via Partial Delete Relaxation".

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. *SOCS'13*.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *AIJ* 90(1-2):279–298.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1–2):5–33.

Borrajo, Daniel and Roubíčková, Anna and Serina, Ivan 2015. Progress in Case-Based Planning. *ACM Computing Surveys* 47(2):35:1–35:39.

Cai, D.; Hoffmann, J.; and Helmert, M. 2009. Enhancing the context-enhanced additive heuristic with precedence constraints. *ICAPS'09*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2013. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *JAIR* 46:343–412.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *AIJ* 221:73–114.

Domshlak, C.; Hoffmann, J.; and Sabharwal, A. 2009. Friends or foes? on planning as satisfiability and abstract cnf encodings. *JAIR* 36:415–469.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2012. Online speedup learning for optimal planning. *JAIR* 44:709–755.

Edelkamp, S. 2001. Planning with pattern databases. *ECP'01*.

Fox, M., and Long, D. 2001. Stan4: A hybrid planning strategy based on subproblem abstraction. *The AI Magazine* 22(3):81–84.

Fox, M.; Gerevini, A. E.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. *ICAPS'06*.

Gerevini, A. E., and Serina, I. 2000. Fast plan adaptation through planning graphs: Local and systematic search techniques. *AIPS'00*.

Gerevini, A., and Serina, I. 2010. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae* 102(3-4):287–323.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *JAIR* 20:239–290.

Gnad, D., and Hoffmann, J. 2015a. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. *ICAPS'15*.

Gnad, D., and Hoffmann, J. 2015b. Red-black planning: A new tractability analysis and heuristic function. *SOCS'15*.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. *AIPS'00*.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. *AAAI'07*.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. *ICAPS'12*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? *ICAPS'09*.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. *ICAPS'08*.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *JACM* 61(3).

Helmert, M. 2004. A planning heuristic based on causal graph analysis. *ICAPS'04*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. *ECAI'14*.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. *SOCS'13*.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. *AAAI'13*.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax *all* variables? *ICAPS'13*.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. *ECAI'08*.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. *ICAPS'12*.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *JAIR* 50:487–533.

Leake, D. B. 1996. Case-Based Reasoning. The MIT Press.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A monte carlo random walk approach. *ICAPS'12*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Torralba, A., and Alcázar, V. 2013. Constrained symbolic search: On mutexes, BDD minimization and more. *SOCS'13*.