Beyond Red-Black Planning: Limited-Memory State Variables (Technical Report)

Patrick Speicher Marcel Steinmetz Daniel Gnad Jörg Hoffmann
Saarland Informatics Campus, Saarland University
Saarbrücken, Germany
speicher@uni-saarland.de;{steinmetz,gnad,hoffmann}@cs.uni-saarland.de

Alfonso Gerevini
University of Brescia
Brescia, Italy
gerevini@ing.unibs.it

Abstract

Red-black planning delete-relaxes only some of the state variables. This is coarse-grained in that, for each variable, it either remembers all past values (red), or remembers only the most recent one (black). We herein introduce *limited-memory* state variables, that remember a subset of their most recent values. It turns out that planning is still **PSPACE**-complete even when the memory is large enough to store all but a single value. Nevertheless, limited memory can be used to substantially broaden a known tractable fragment of red-black planning, yielding better heuristic functions in some domains.

Introduction

The delete relaxation has been instrumental for scalability in satisficing classical planning (e. g., (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Gerevini, Saetti, and Serina 2003; Richter and Westphal 2010)), but it also has weaknesses. *Partial delete relaxation* methods interpolate between delete-relaxed planning and real planning (Keyder, Hoffmann, and Haslum 2012; 2014; Katz, Hoffmann, and Domshlak 2013; Domshlak, Hoffmann, and Katz 2015). We herein focus on red-black planning, which applies the delete-relaxed semantics to a subset of state variables (the "red" ones), letting them accumulate their values, while keeping the real semantics for the others (the "black" ones). Red-black planning is tractable if the dependencies between black variables are acyclic, and each black variable is *invertible*. The heuristic function based on that tractable fragment, h^{RB}, can be quite useful; it was a key part of the Mercury system that performed well in IPC'14.

Yet distinctions at the level of entire state variables are very coarse-grained: either we remember all past values of a variable (red), or only the most recent one (black). We herein introduce *limited-memory* state variables, that allow more fine-grained relaxations through remembering a subset of their most recent values. As we show, planning is still **PSPACE**-complete even when the memory is large enough to store all but a single value. Nevertheless, limited memory can be used to substantially extend the abovementioned tractable fragment. In h^{RB}, non-invertible v cannot be painted black, because v may not be able to go back to a previous value when required. Our idea is, instead of fully delete-relax such v, to give v "just enough" memory to ensure this property. The resulting heuristic function proves to be superior in some domains.

Preliminaries

We use the *finite-domain representation (FDR)* framework. We introduce FDR and its delete relaxation as special cases of red-black planning. A *red-black (RB)* planning task is a tuple $\Pi = \langle V^B, V^R, A, I, G\rangle$. V^B is a set of *black variables* and V^R is a set of *red variables*, where $V^B \cap V^R = \emptyset$ and each $v \in V := V^B \cup V^R$ has a finite domain $\mathcal{D}[v]$. The *initial state* I is a complete assignment to V, the *goal* G is a partial assignment. Each action a is a pair $\langle \mathit{pre}_a, \mathit{eff}_a\rangle$ of partial assignments called *precondition* and *effect*. For a partial assignment p, $V(p)$ denotes the subset of V instantiated by p. For $V' \subseteq V(p)$, $p[V']$ denotes the value of V' in p.

A state s assigns each $v \in V$ a non-empty subset $s[v] \subseteq \mathcal{D}[v]$, where $|s[v]| = 1$ for all $v \in V^B$. An action a is *applicable* in s if $\mathit{pre}_a[v] \in s[v]$ for all $v \in V(\mathit{pre}_a)$. Applying a in s changes the value of $v \in V(\mathit{eff}_a) \cap V^B$ to $\{\mathit{eff}_a[v]\}$, and changes the value of $v \in V(\mathit{eff}_a) \cap V^R$ to $s[v] \cup \{\mathit{eff}_a[v]\}$. The resulting state is denoted $s[\![a]\!]$. A *plan* is an action sequence whose iterative application in I leads to a state s where $G[v] \in s[v]$ for all $v \in V(G)$.

Π is an FDR planning task if $V = V^B$, and is a *monotonic (MFDR)* planning task if $V = V^R$. The delete relaxation uses MFDR to approximate FDR; *red-black relaxation* uses the more general RB instead. The state-of-the-art red-black heuristic, h^{RB}, does so via choosing a variable *painting* – a partitioning of V into V^B and V^R – such that the *causal graph* over V^B is acyclic, and each $v \in V^B$ is *relaxed side effects invertible*. We will explain these notions later on when considering our extended heuristic function.

Limited-Memory State Variables

Our notion of limited-memory variables is an instance of what we baptize *trace-memory relaxation*. In its most general form, such a relaxation $\mathcal{R}$ is defined through a function $\mathcal{R}[v] : \mathcal{D}[v]^+ \mapsto 2^{\mathcal{D}[v]}$ for each variable v: For a given value history $\delta \in \mathcal{D}[v]^+$ (v's value sequence on the current plan prefix), $\mathcal{R}[v](\delta)$ is the value subset that v will "remember" under $\mathcal{R}$. We apply the following two restrictions to $\mathcal{R}[v]$: (1) For all $v \in V$ and $\delta = d_1, \dots, d_n \in \mathcal{D}[v]^+$, $\mathcal{R}[v](\delta) \subseteq \{d_1, \dots, d_n\}$, (2) the "latest" value d_n is in $\mathcal{R}[v](\delta)$. Given the state history $I = s_0, a_1, \dots, a_{m-1}, s_{m-1} = s$, we define $s[\![a_m]\!][v] := \mathcal{R}[v](d_1, \dots, d_n)$ where $d_1, \dots, d_n$ denote v's value changes on this path. An $\mathcal{R}$-*relaxed plan* is a plan under these semantics.

The delete relaxation uses $\mathcal{R}[v](d_1, \dots, d_n) := \{d_1, \dots, d_n\}$. Red-black relaxation uses the same for $v \in V^R$, and uses $\mathcal{R}[v](d_1, \dots, d_n) := \{d_n\}$ for $v \in V^B$. Our new *limited-memory* relaxation is parameterized by a *memory size* $M[v]$, $1 \leq M[v] \leq |\mathcal{D}[v]|$, for every $v \in V$. It uses $\mathcal{R}[v](d_1, \dots, d_n) := \{d_k, \dots, d_n\}$, where $k = 1$ if $|\{d_1, \dots, d_n\}| \leq M[v]$, and otherwise k is s.t. $|\{d_k, \dots, d_n\}| = M[v]$. In words, each variable remembers as many of its recent values as fit into memory. Observe that $v \in V^R$ is characterized by $M[v] = |\mathcal{D}[v]|$, and $v \in V^B$ is characterized by $M[v] = 1$.

The space of trace-memory relaxations is a refinement hierarchy, where $\mathcal{R}$ *refines* $\mathcal{R}'$ if, for every $v \in V$ and $\delta \in \mathcal{D}[v]^+$, we have $\mathcal{R}[v](\delta) \subseteq \mathcal{R}'[v](\delta)$. If $\mathcal{R}$ refines $\mathcal{R}'$, then $\mathcal{R}$ is more informed than $\mathcal{R}'$: Any $\mathcal{R}$-relaxed plan is an $\mathcal{R}'$-relaxed plan, but not vice versa (unless $\mathcal{R} = \mathcal{R}'$). The unique coarsest element in the refinement hierarchy is the delete relaxation, $\mathcal{R}^+$, as every $\mathcal{R}$ refines $\mathcal{R}^+$. The unique most refined element is the standard FDR semantics, $\mathcal{R}^*$, as $\mathcal{R}^*$ refines every $\mathcal{R}'$. A limited-memory relaxation $\mathcal{R}$ refines a red-black relaxation iff $M[v] = 1$ for all $v \in V^B$.

Worst-Case Complexity

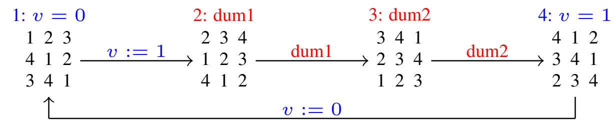
We consider the complexity of $\mathcal{R}$-relaxed plan existence for limited-memory relaxations $\mathcal{R}$. Observe first that $\mathcal{R}$-relaxed plan existence remains a member of **PSPACE**, thanks to a similar non-deterministic polynomial space algorithm as given for $\mathcal{R}^*$ by Bylander (1994) (it suffices to remember the most recent $M[v]$ values for each v). Observe further that, likewise easily, $\mathcal{R}$-relaxed plan existence in full generality remains **PSPACE**-hard, simply as it includes the case where $\mathcal{R} = \mathcal{R}^*$ and we do not actually relax anything. That case is, of course, not interesting, so we will exclude it. Observe finally that limited-memory relaxation is pointless for binary variables v, i. e., for $|\mathcal{D}[v]| = 2$, as one can only either set $M[v] = 2$ (delete relaxation), or $M[v] = 1$ (no relaxation).

We hence focus on the case where, for all v, $|\mathcal{D}[v]| > 2$ and $M[v] > 1$. Somewhat surprisingly perhaps, it turns out that this remains **PSPACE**-hard even in the maximally relaxed setting, where $M[v] \geq |\mathcal{D}[v]| - 1$ for all v, i. e., at most a single value per variable is forgotten:

Theorem 1 *For FDR tasks where $|\mathcal{D}[v]| > 2$ for all v, and limited-memory relaxations $\mathcal{R}$ where $M[v] \geq |\mathcal{D}[v]| - 1$ for all v, deciding $\mathcal{R}$-relaxed plan existence is **PSPACE**-hard.*

Proof Sketch: Our proof is by reduction from FDR with binary variables, $\mathcal{D}[v] = \{0, 1\}$ for all v, and where $pre_a[v] = 0$ whenever $\mathit{eff}_a[v] = 1$, as well as $pre_a[v] = 1$ whenever $\mathit{eff}_a[v] = 0$. Bylander's (1994) **PSPACE**-hardness proof (his Theorem 3.1) is easy to adapt to this FDR fragment.

Our key observation is that any binary variable v can be encoded into a *counter*, consisting of three *counter variables* v_1, v_2, v_3 with domain $\mathcal{D}[v_i] = \{1, 2, 3, 4\}$ and $M[v_i] = 3$. The counter is arranged to behave as follows:



The counter has a life cycle through 4 *counter states* – memory contents of the counter variables – of which state 1 encodes $v = 0$ and state 4 encodes $v = 1$. State i is characterized by the single value i all counter variables have in common (left-top to right-bottom diagonal in the figure). To achieve this behavior, the action moving the counter from state j to state $j + 1$ has precondition $\{v_1 = j, v_2 = j, v_3 = j\}$, and adds a new value to each v_i corresponding to the new rightmost column in the figure. Any precondition/goal $v = 0$ can then be replaced by $\{v_1 = 1, v_2 = 1, v_3 = 1\}$, and any precondition/goal $v = 1$ can be replaced by $\{v_1 = 4, v_2 = 4, v_3 = 4\}$. To move from counter state 1 ($v = 0$) to counter state 4 ($v = 1$), we insert two dummy actions which must be applied, and will be applicable only, after an effect $v := 1$ (which by construction necessitates a precondition $v = 0$ so will happen only in counter state 1).

The construction assumes that, at the start of the lifecycle, the counter is in state 1, which cannot be specified in FDR as each v_i has only one initial value. But that issue can be solved through an additional counter-initialization phase. ■

A full proof of this theorem is available in the appendix.

Extending h^{RB} with Non-Invertible Variables

Towards explaining our extension of h^{RB}, let us summarize the workings of that heuristic function. To evaluate a state s, $h^{\mathrm{RB}}(s)$ first computes a fully delete relaxed plan π^+ for s, and then refines π^+ to treat the variables V^B accordingly. The refinement process goes through $\pi^+ = \langle a_1^+, \dots, a_k^+ \rangle$ from front to back, executes each a_i^+ using the RB semantics, and inserts a *repair* sequence π^B for $v \in V^B$ whenever the precondition of a_{i+1}^+ on v is not satisfied.[1] That sequence π^B is found by solving a planning task Π^B over the black variables V^B. The initial state of Π^B is the state of V^B before executing a_{i+1}^+; the goal is the precondition of a_{i+1}^+ on v. We denote these by I^B and G^B respectively.

For h^{RB} to be tractable, solving Π^B must be tractable. To that end, h^{RB} relies on a known tractability result for planning, restricting (a) cross-variable dependencies to be acyclic, and restricting (b) each variable to be able to move from any value d to any other value d' (Knoblock 1994; Williams and Nayak 1997; Brafman and Domshlak 2003; Helmert 2006; Chen and Giménez 2010). Figure 1 shows the algorithm, *repair planning*, that h^{RB} uses in that setting.

Repair planning sequentializes the handling of black variables, from "clients" to "servants" according to prerequisite (a) above. Formally, (a) is captured in terms of the *causal graph* over V^B (Knoblock 1994; Brafman and Domshlak 2003; Helmert 2006). In the acyclic case, every action

[1] Actually, this is what Domshlak et al. (2015) call *relaxed plan repair*, a simpler and empirically worse variant of the *red facts following* algorithm h^{RB} uses. We explain relaxed plan repair as red facts following is complicated, and our extensions are identical for both algorithms. Our implementation extends red facts following.

```
Algorithm: REPAIRPLANNING(Π^B)
  π^B ← ⟨⟩
  for i = n downto 1 do
    /* Denote π^B = ⟨a_1, ..., a_k⟩*/
    d ← I^B[v_i]
    for j = 1 to k do
      π_j ← ⟨⟩
      if pre_{a_j}[v_i] is defined then
        π_j ← path(v_i, d, pre_{a_j}[v_i])
        d ← pre_{a_j}[v_i]
    π_{k+1} ← ⟨⟩
    if G^B[v_i] is defined then
      π_{k+1} ← path(v_i, d, G^B[v_i])
    π^B ← π_1 ∘ ⟨a_1⟩ ∘ ··· ∘ π_k ∘ ⟨a_k⟩ ∘ π_{k+1}
  return π^B
```

Figure 1: Domshlak et al.'s (2015) pseudocode for repair planning in the h^{RB} heuristic, solving black-variable repair tasks Π^B. $v_1, \dots, v_n$ is an ordering of V^B consistent with the acyclic V^B causal graph. $path(v_i, d, d')$ denotes an action sequence inducing a path moving v_i from d to d'.

affects exactly one $v \in V^B$; and the arcs in the causal graph over V^B are exactly the servant-client relations, i. e., an arc $v_1 \rightarrow v_2$ indicates that there is an action a where $v_1 \in V(pre_a)$ and $v_2 \in V(eff_a)$.

Repair planning fixes the sub-plans for each variable v_i one-by-one, processing v_i after all its potential clients are already processed. The sub-plan created for v_i then provides the values requested by the clients, as manifested in the action sequence π^B constructed so far; and it achieves v_i's own goal (if any) at the end. Prerequisite (b) is required so that the requested paths $path(v_i, d, d')$ always exist. To ensure the latter, h^{RB} requires every value transition $d_1 \rightarrow d_2$ of v_i to be *relaxed side effects (RSE-)invertible*. The exact definition of RSE-invertibility is not needed to understand our contribution. The important aspect is: RSE-invertibility entails that v_i can always get from d to its start value $I^B[v_i]$, back along the path it came from. From $I^B[v_i]$, v_i can get to d' in the same way the input delete-relaxed plan π^+ does.

Our extension consists in a more elaborate handling of the case where some value transitions of v_i are *not* RSE-invertible. Currently, h^{RB} has to paint v_i red, relaxing it completely and removing it from Π^B. But this is an overkill if, for example, only a few instances of $path(v_i, d, d')$ will fail, or if the required paths all exist yet are via transitions not captured by RSE-invertibility. Our idea is to relax v_i "just enough" to guarantee prerequisite (b).

Our relaxation now distinguishes three kinds of variables: fully relaxed (red) and unrelaxed (black) variables as before; and partially relaxed variables, that we call *gray variables*, V^G. For prerequisite (a), as the gray variables are members of Π^B, i. e., are part of repair planning, we require the causal graph over $V^B \cup V^G$ to be acyclic. For prerequisite (b), we design a suitable limited-memory relaxation $\mathcal{R}$ on $v \in V^G$.

The relaxation $\mathcal{R}[v]$ of $v \in V^G$ needs to be such that, whenever a path $path(v, d, d')$ is required, v remembers at least one value d_0 from which a path to d' exists. Such $\mathcal{R}[v]$ can, in principle, be obtained through limited-memory relaxation with a suitable memory size. For example, one can set $M[v]$ so that $I^B[v]$ can always be reached. Yet, for most instances of $path(v_i, d, d')$, this memory size will be much larger than actually needed to make d' reachable. This leads us to the idea of not fixing the memory size once a-priori (*offline*), but setting it on a per-need basis (*online*) instead.

Our online limited-memory relaxation adjusts the memory size dynamically during heuristic function computation. The method is thus specific to the particular computation addressed, i. e., repair planning as per Figure 1. The memory size is set individually for each relevant time point during the run of that algorithm. Namely, say that $v_i \in V^G$. Consider the sub-plan construction for v_i, i. e., the inner **for** loop of Figure 1, and consider any iteration j of that loop. We define a memory size $M_j[v_i]$, and therewith a limited-memory relaxation $\mathcal{R}^{\text{ON}}_j[v_i]$, individually for each j. Denote by $\langle d_1, \dots, d_m \rangle$ the value sequence of v_i along $\pi_1 \circ \dots \circ \pi_{j-1}$ constructed so far. If no request of the form $path(v_i, d_m, d')$ is made, we set $M_j[v_i] := 1$. Otherwise, if a request of the form $path(v_i, d_m, d')$ is made, we set $M_j[v_i]$ such that $\mathcal{R}^{\text{ON}}_j[v_i](d_1, \dots, d_m) = \{d_l, \dots, d_m\}$ where $l \leq m$ is the highest index for which a path from d_l to d' exists. In words, the online relaxation relaxes v_i only where needed during repair planning, and only to the extent required for success. We force the repair planner to be as unrelaxed as possible (without necessitating backtracking) in the handling of v_i.

The basic properties of this construction follow easily by extending Domshlak et al.'s (2015) observations. Say we run the modified h^{RB} as above on a state s, using a partition of V into V^B, V^G, and V^R. Then *(i) the algorithm terminates in time polynomial in the length of the action sequence π^{ON} returned*. Denote by $\mathcal{R}^{\text{ON}}$ the overall dynamic relaxation, i. e., $\mathcal{R}^{\text{ON}}$ fully relaxes $v \in V^R$, it does not relax $v \in V^B$, and for $v_i \in V^G$ and v_i's value sequence along $\langle a_1, \dots, a_j \rangle$ at iteration j, $\mathcal{R}^{\text{ON}}[v_i]$ constitutes our limited-memory relaxation with memory size $M_j[v_i]$. Then *(ii) π^{ON} is an $\mathcal{R}^{\text{ON}}$-relaxed plan*, and *(iii) $\mathcal{R}^{\text{ON}}$ is a refinement of the red-black relaxation with black variables V^B and red variables $V^G \cup V^R$*. We denote the resulting heuristic function – returning the length of π^{ON} – by h^{Gray}. In the sense of (ii) and (iii) put together, h^{Gray} is more informed than h^{RB}.

Experiments

We implemented h^{Gray} in FD (Helmert 2006), extending Domshlak et al.'s (2015) implementation of h^{RB}. We adopt Domshlak et al.'s *stop search* technique, which tests whether the relaxed plan is actually a real plan, and if so, stops the search. We use this in all configurations tested.

We compare h^{Gray} to its direct predecessor h^{RB}, and to h^{FF} as a baseline. We use FD's canonical search algorithm for satisficing planning, greedy best-first search (GBFS). To get an unbiased comparison between the different heuristics, respectively to compare to state-of-the-art, we report results from disabling, as well as enabling FD's preferred operators. For the latter, to enhance comparability, we use the same preferred operators, taken from h^{FF}, for all three heuristics.[2]

[2]Taking preferred operators from the individual heuristics does not significantly affect the results.

Domain	#	w/ p.o. h^{FF}	w/ p.o. h^{RB}	w/ p.o. h^{Gray}	stop search at I h^{FF}	stop search at I h^{RB}	stop search at I h^{Gray}	w/o p.o. h^{FF}	w/o p.o. h^{RB}	w/o p.o. h^{Gray}
Airport	50	**37**	**37**	36	0	0	0	36	36	**37**
Barman	40	35	**39**	**39**	0	0	0	31	**32**	**32**
Blocksworld	35	35	35	35	0	**1**	**1**	35	35	35
Childsnack	20	3	**6**	**6**	0	0	0	0	0	0
Depots	22	18	18	18	0	**1**	**1**	14	**16**	**16**
Elevators	50	50	50	50	0	**50**	**50**	48	**50**	**50**
Floortile	40	8	**9**	8	0	0	0	8	**9**	8
Freecell	80	80	80	80	0	0	0	**79**	**79**	78
Gripper	20	20	20	20	0	**20**	**20**	20	20	20
Hiking	20	**19**	12	12	0	0	0	**16**	11	11
Logistics	63	63	63	63	0	**63**	**63**	54	**63**	**63**
Miconic	150	150	150	150	0	**150**	**150**	150	150	150
Mprime	35	35	35	35	0	0	**9**	31	31	**34**
Mystery	30	**16**	**16**	15	0	0	**7**	18	18	**19**
NoMystery	20	10	**15**	**15**	0	0	0	9	**14**	**14**
Openstacks	100	100	100	100	0	0	**77**	100	100	100
Parcprinter	51	39	**51**	39	0	0	0	39	**51**	39
Parking	40	**37**	36	36	0	0	0	**29**	28	**29**
Pathways	30	20	20	20	0	0	0	11	11	**12**
PipesNoTank	50	**44**	**44**	43	0	0	0	30	**32**	30
PipesTank	50	39	39	39	0	0	0	23	23	**25**
Rovers	40	40	40	40	0	3	**40**	23	27	**40**
Satellite	36	36	36	36	0	**10**	**10**	30	**36**	**36**
Scanalyzer	50	46	**48**	**48**	0	0	0	44	**46**	**46**
Sokoban	50	48	48	48	0	0	0	**48**	46	**48**
Storage	30	20	20	20	0	**3**	**3**	18	18	18
Tetris	20	**16**	14	15	0	0	0	7	**8**	**8**
Thoughtful	20	**14**	12	12	0	0	0	9	**11**	9
Tidybot	20	15	14	**16**	0	0	0	14	15	**16**
TPP	30	30	30	30	0	**5**	**5**	23	**24**	19
Transport	70	45	**70**	**70**	0	**70**	**70**	16	**70**	**70**
Trucks	30	**20**	18	18	0	0	0	**17**	16	14
Visitall	40	4	**40**	**40**	0	**40**	**40**	4	**40**	**40**
Woodworking	50	50	50	50	0	**1**	**1**	49	**50**	49
Zenotravel	20	20	20	20	1	**20**	9	**20**	**20**	19
OTHERS	145	145	145	145	1	1	1	142	142	142
Log'98-Battery	35	17	25	**30**	0	0	**20**	7	7	**23**
$\sum$	1682	1424	**1505**	1497	2	438	**577**	1252	1385	**1399**

Table 1: Coverage with respectively without preferred operators ("p.o."), and number of instances solved without search because stop search fires at the initial state. Best results within each category shown in **boldface**. Domains where all three heuristic functions perform equally within every category are summarized in "OTHERS".

To choose the partitioning of V into V^B, V^G, and V^R, we extend Domshlak et al.'s *painting strategy*. That strategy starts with $V^B = V$. It then paints red, i. e., moves to V^R, (1) all causal graph leaf variables, and (2) all non-RSE-invertible variables. From the remaining variables V^B, (3) iteratively one v is picked and painted red, until the causal graph over V^B is acyclic. We use the same strategy, except that we omit step (2) and, upon termination, paint all non-RSE-invertible $v \in V^B$ gray (moving them to V^G).

To pick v in (3), Domshlak et al. experiment with many prioritization options. Here we use their canonical (simple and performant) option, always selecting the v with highest index in Helmert's (2004) *level* ordering. More precisely, we experiment with two different strategies, (A) as just described, (B) preferring non-RSE-invertible v and breaking ties by the level ordering (so that RSE-invertible v are more likely to end up black). With (B), h^{Gray} is very close to h^{RB}, with differences only where some variables can be painted gray on top of the black ones in h^{RB}. With (A), the paintings tend to differ more. For space reasons, we show data only for (A) and briefly summarize the differences for (B).

We run all IPC STRIPS benchmarks. We assume unit action costs throughout. As a showcase for the strengths of our method, we also run a modification of IPC'98 Logistics, where load/unload consumes battery power, and in every city there is one location where the battery can be recharged. The battery variable is not RSE-invertible so h^{RB} must paint it red, whereas we can paint it gray. If that is done – which typically happens with strategy (A) though never with (B) – then stop search succeeds in the initial state, i. e., our relaxed plan for I is a real plan. All experiments were run on a cluster of Intel Xeon E5-2650v3 machines, with runtime (memory) limits of 30 minutes (4 GB). Consider Table 1.

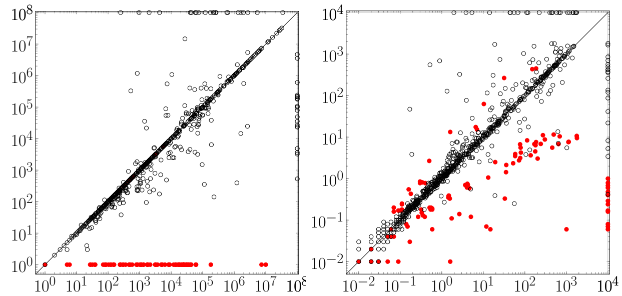


Figure 2: Search space size (left) and total runtime (right), for h^{RB} (x-axis) vs. h^{Gray} (y-axis), without preferred operators. Red data points for domains where h^{Gray} stops search at the initial state in more instances than h^{RB}.

With preferred operators in use (left part of table), there isn't much difference between h^{RB} and h^{Gray}, except for Parcprinter and Logistics'98-Battery. As we see in the right part of the table, this is partly because the differences are hidden beneath the preferred operators. With strategy (B), h^{Gray}'s loss in Parcprinter disappears; total coverage on IPC benchmarks is marginally better for h^{RB} with preferred operators (+2), and is better for h^{Gray} without them (+9).

The middle part of the table shows the number of times stop search solves the task directly at the initial state, the main advantage of this kind of heuristic (Domshlak, Hoffmann, and Katz 2015). Here h^{Gray} is clearly superior to h^{RB}, as it does not have to paint key non-RSE-invertible variables red. With strategy (B), h^{Gray} stop search solves all Zenotravel instances, but does not solve any Logistics'98-Battery instances because it uses the same paintings as h^{RB}.

Figure 2 gives a search space and runtime comparison. The main advantage of h^{Gray} over h^{RB} lies in the additional stop-search prowess, thus adding another increment to that same main advantage of h^{RB} over h^{FF}.

Conclusion

While red-black planning has opened a huge space of relaxations in between the delete relaxation and real (unrelaxed) planning, that space is still limited in having to take decisions at the level of entire state variables. Our contribution lies in pointing this out, putting forward the much richer concept of trace-memory relaxation, and examining limited-memory variables as a first instantiation. The empirical results are not overwhelming but certainly show promise, and an entire universe of future research lies in exploring the space of trace-memory relaxations in more detail.

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”, and was partially supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA, grant no. 16KIS0656).

References

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

Brafman, R., and Domshlak, C. 2003. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research* 18:315–349.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Chen, H., and Giménez, O. 2010. Causal graphs and structurally restricted planning. *Journal of Computer and System Sciences* 76(7):579–592.

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

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

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Koenig, S.; Zilberstein, S.; and Koehler, J., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 161–170. Whistler, Canada: Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013. Who said we need to relax *all* variables? In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 126–134. Rome, Italy: AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 128–136. AAAI Press.

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

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

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

Williams, B. C., and Nayak, P. P. 1997. A reactive planner for a model-based executive. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1178–1185. Nagoya, Japan: Morgan Kaufmann.

Proofs

Theorem 2 *For FDR tasks where $|\mathcal{D}[v]| > 2$ for all v, and limited-memory relaxations $\mathcal{R}$ where $M[v] \geq |\mathcal{D}[v]| - 1$ for all v, deciding $\mathcal{R}$-relaxed plan existence is* ***PSPACE**-hard.*

Proof: Our proof is by reduction from FDR with binary variables, $\mathcal{D}[v] = \{0, 1\}$ for all v, and where $pre_a[v] = 0$ whenever $eff_a[v] = 1$, as well as $pre_a[v] = 1$ whenever $eff_a[v] = 0$. Bylander's (1994) **PSPACE**-hardness proof (his Theorem 3.1) is easy to adapt to this FDR fragment.

Our key observation is that any binary variable v can be encoded into a *counter*, consisting of three *counter variables* v_1, v_2, v_3 with domain $\mathcal{D}[v_i] = \{1, 2, 3, 4\}$ and $M[v_i] = 3$. The counter is arranged to behave as follows:

$$\begin{array}{ccccccc}
1\text{: } v=0 & & 2\text{: dum1} & & 3\text{: dum2} & & 4\text{: } v=1 \\
\begin{matrix}1&2&3\\4&1&2\\3&4&1\end{matrix} & \xrightarrow{v:=1} & \begin{matrix}2&3&4\\1&2&3\\4&1&2\end{matrix} & \xrightarrow{\text{dum1}} & \begin{matrix}3&4&1\\2&3&4\\1&2&3\end{matrix} & \xrightarrow{\text{dum2}} & \begin{matrix}4&1&2\\3&4&1\\2&3&4\end{matrix} \\
\uparrow & & & v:=0 & & & \lrcorner
\end{array}$$

The counter has a life cycle through 4 *counter states* – memory contents of the counter variables – of which state 1 encodes $v = 0$ and state 4 encodes $v = 1$. State i is characterized by the single value i all counter variables have in common (left-top to right-bottom diagonal in the figure).

To achieve this behavior, the action moving the counter from state j to state $j + 1$ has precondition $\{v_1 = j, v_2 = j, v_3 = j\}$, and adds a new value to each v_i corresponding to the new rightmost column in the figure. Any precondition/goal $v = 0$ can then be replaced by $\{v_1 = 1, v_2 = 1, v_3 = 1\}$, and any precondition/goal $v = 1$ can be replaced by $\{v_1 = 4, v_2 = 4, v_3 = 4\}$. To move from counter state 1 ($v = 0$) to counter state 4 ($v = 1$), we insert two dummy actions which must be applied, and will be applicable only, after an effect $v := 1$ (which by construction necessitates a precondition $v = 0$ so will happen only in counter state 1).

In detail, there are four types of actions interacting with the counter: (1) ones that change the value of v from 0 to 1, corresponding to the counter state change from 1 to 2; (2) ones that change the value of v from 1 to 0, corresponding to the counter state change from 4 to 1; (3) the dummy action changing the counter state from 2 to 3; (4) the dummy action changing the counter state from 3 to 4. For this setup, we modify the action set as follows:

(1) Any action a with $pre_a[v] = 0$ and $eff_a[v] = 1$ is replaced by a' with $pre_{a'} = \{v_1 = 1, v_2 = 1, v_3 = 1\} \cup pre_a[V \setminus \{v\}]$ and $eff_{a'} = \{v_1 = 4, v_2 = 3, v_3 = 2\} \cup eff_a[V \setminus \{v\}]$.

(2) Any action a with $pre_a[v] = 1$ and $eff_a[v] = 0$ is replaced by a' with $pre_{a'} = \{v_1 = 4, v_2 = 4, v_3 = 4\} \cup pre_a[V \setminus \{v\}]$ and $eff_{a'} = \{v_1 = 3, v_2 = 2, v_3 = 1\} \cup eff_a[V \setminus \{v\}]$.

(3) We introduce a new dummy action dum1 with $pre_{dum1} = \{v_1 = 2, v_2 = 2, v_3 = 2\}$ and $\mathit{eff}_{dum1} = \{v_1 = 1, v_2 = 4, v_3 = 3\}$.

(4) We introduce a new dummy action dum2 with $pre_{dum2} = \{v_1 = 3, v_2 = 3, v_3 = 3\}$ and $\mathit{eff}_{dum2} = \{v_1 = 2, v_2 = 1, v_3 = 4\}$.

Observe that, beside the respective dummy action, no other action affecting v_1, v_2, v_3 is applicable if the counter is in state 2 or 3; and no dummy action is applicable if the counter is in state 1 or 4. Thus, the counter as a whole behaves exactly like v, if we abstract away the dummy actions.

The above specification is for encoding a single variable v. To encode the overall set of variables, we simply process each the variables one at a time. Observe that, for each variable v in this process, the encoding takes time and space $O(k)$ where k is the number of actions affecting v. Hence, in particular, the overall encoding time is polynomial in the size of the input task.

The construction so far assumes that, at the start of the lifecycle, the counter is in state 1. Yet that cannot be specified in FDR as each v_i has only one initial value. But that issue can be solved through an additional counter-initialization phase.

Namely, we set the leftmost column (143) in I, and introduce two initializing actions av_1^I and av_2^I adding the columns 214 and 321 respectively, where av_2^I has precondition 214 to enforce the correct initialization order. To enforce a strict separation between the initializing and lifecycle phases, we introduce a guard variable v^g with $\mathcal{D}[v^g] = \{1, 2, 3\}$ and $M[v^g] = 2$. We set $I(v^g) = 1$, give av_1^I precondition $v^g = 1$ and effect $v^g = 2$, give av_2^I precondition $v^g = 1$ and effect $v^g = 3$, and introduce $v^g = 3$ into the precondition of every other action. That is, we modify the action set as follows:

- For any action a, we set $pre_a := pre_a \cup \{v^g = 3\}$.
- We introduce a new dummy action av_1^I with $pre_{av_1^I} = \{v_1 = 1, v_2 = 4, v_3 = 3, v^g = 1\}$ and $\mathit{eff}_{av_1^I} = \{v_1 = 2, v_2 = 1, v_3 = 4, v^g = 2\}$.
- We introduce a new dummy action av_2^I with $pre_{av_2^I} = \{v_1 = 2, v_2 = 1, v_3 = 4, v^g = 1\}$ and $\mathit{eff}_{av_2^I} = \{v_1 = 3, v_2 = 2, v_3 = 1, v^g = 3\}$.

Clearly, av_1^I and av_2^I are only applicable in the intended order. As, after applying av_2^I, $v^g = 1$ disappears from v^g's memory and can never be reachieved, av_1^I and av_2^I are deactivated forever after applying them once. All other actions are only applicable after the initialization, i. e., after av_2^I, due to the new precondition $v^g = 3$. This concludes the proof. ■