Beyond Red-Black Planning: Limited-Memory State Variables

Patrick Speicher, Marcel Steinmetz, Daniel Gnäd, Jörg Hoffmann
Saarland University, Saarland Informatics Campus
Saarbrücken, Germany
speicher@uni-saarland.de;
steinmetz.gnaed.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 Westhal 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, hRB, 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 hRB, 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 \( \Pi = (V^B, V^R, A, I, G) \). \( V^B \) 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 \( D[v] \). The initial state \( I \) is a complete assignment to \( V \), the goal \( G \) is a partial assignment. Each action \( a \) is a pair \( \langle \text{pre}_a, \text{eff}_a \rangle \) 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 \( s[v] \subseteq D[v] \), where \( |s[v]| = 1 \) for all \( v \in V^B \). An action \( a \) is applicable in \( s \) if \( \text{pre}_a[v] \subseteq s[v] \) for all \( v \in V(\text{pre}_a) \). Applying \( a \) in \( s \) changes the value of \( v \in V(\text{eff}_a) \cap V^B \) to \( \text{eff}_a[v] \), and changes the value of \( v \in V(\text{eff}_a) \cap V^R \) to \( s[v] \cup \text{eff}_a[v] \). The resulting state is denoted \( s[a] \). A plan is an action sequence whose iterative application in \( I \) leads to a state \( s \) where \( G[v] \subseteq s[v] \) for all \( v \in V(G) \).

II 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 \( v \in V^R \) 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 \( R \) is defined through a function \( R[v] : D[v] \rightarrow 2^{P[v]} \) for each variable \( v \). For a given value history \( \delta \in D[v] \), \( v's \) value sequence on the current plan prefix), \( R[v](\delta) \) is the value subset that \( v \) will "remember" under \( R \). We apply the following two restrictions to \( R[v] \):
(1) For all \( v \in V \) and \( \delta = d_1, \ldots, d_n \in D[v]^+ \), \( \mathcal{R}[v](\delta) \subseteq \{ d_1, \ldots, d_n \} \), (2) the “lastest” value \( d_n \) is in \( \mathcal{R}[v](\delta) \). Given the state history \( I = s_0, a_1, \ldots, a_{m-1}, s_m = s \), we define \( s \cdot a_m \| v \) := \( \mathcal{R}[v](d_1, \ldots, d_n) \) where \( d_1, \ldots, d_n \) denote \( v \)'s value changes on this path. An \( \mathcal{R} \)-relaxed plan is a plan under these semantics.

The delete relaxation uses \( R[v](d_1, \ldots, d_n) := \{ d_1, \ldots, d_n \} \). Red-black relaxation uses the same for \( v \in V^R \), and uses \( R[v](d_1, \ldots, d_n) := \{ d_n \} \) for \( v \in V^B \). Our new limited-memory relaxation is parameterized by a memory size \( M[v] \), \( 1 \leq M[v] \leq |D[v]| \), for every \( v \in V \). It uses \( R[v](d_1, \ldots, d_n) := \{ d_k, \ldots, d_n \} \), where \( k = 1 \) if \( \{ d_1, \ldots, d_n \} \leq M[v] \), and otherwise \( k \) is s.t. \( (\{ d_k, \ldots, d_n \} \leq M[v] \). In words, each variable remembers as many of its recent values as fit into memory. Observe that \( v \in V^R \) is characterized by \( M[v] = |D[v]| \), 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 \( \delta \in D[v]^+ \), 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 \( \text{PSPACE} \), 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 \( \text{PSPACE} \)-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 \( |D[v]| = 2 \), 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 \( v, |D[v]| > 2 \) and \( M[v] > 1 \). Somewhat surprisingly perhaps, it turns out that this remains \( \text{PSPACE} \)-hard even in the maximally relaxed setting, where \( M[v] \geq |D[v]| - 1 \) for all \( v \), i.e., at most a single value per variable is forgotten:

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

**Proof Sketch:** Our proof is by reduction from FDR with binary variables, \( D[v] = \{ 0, 1 \} \) for all \( v \), and where \( \text{pre}_a[v] = 0 \) whenever \( \text{eff}_a[v] = 1 \), as well as \( \text{pre}_a[v] = 1 \) whenever \( \text{eff}_a[v] = 0 \). Bylander’s (1994) \( \text{PSPACE} \)-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 \( D[v] = \{ 1, 2, 3, 4 \} \) and \( M[v] = 3 \). The counter is arranged to behave as follows:

```
1: v = 0
2: dum1
3: dum2
4: v = 1
```

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 a technical report (Speicher et al. 2017).

**Extending h^RB with Non-Invertible Variables**

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

For \( h^RB \) to be tractable, solving \( \Pi^B \) 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.

---

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.
Figure 1: Domshlak et al.’s (2015) pseudocode for repair planning in the $h_{RB}$ heuristic, solving black-variable repair tasks $\Pi^B$. $v_1, \ldots, v_n$ 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 $d'$. 

Algorithm: REPAIRPLANNING($\Pi^B$)

\[ \pi^B = () \]

for $i = 0$ downto 1 do

// Denote $\pi^B = (a_1, \ldots, a_k)^*$/

$d' \leftarrow I^B[v_i]$

for $j = 1$ to $k$ do

$\pi_j \leftarrow ()$

if $\text{pre}_a[v_i]$ is defined then

$\pi_j \leftarrow \text{path}(v_i, d, \text{pre}_a[v_i])$

$d \leftarrow \text{pre}_a[v_i]$

$\pi_{k+1} \leftarrow ()$

if $G^B[v_i]$ is defined then

$\pi_{k+1} \leftarrow \text{path}(v_i, d, G^B[v_i])$

$\pi^B \leftarrow \pi_1 \circ (a_1) \circ \cdots \circ \pi_k \circ (a_k) \circ \pi_{k+1}$

return $\pi^B$

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 graphs over $V^B$ (Knoblock 1994; Brafman and Domshlak 2003; Helmert 2006). In the acyclic case, every action 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 $v_1 \in V(\text{pre}_a)$ and $v_2 \in V(\text{eff}_a)$.

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 $\pi^B$ 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 $\pi$ 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 $\Pi^B$. 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 $\Pi^B$, 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 $path(v, d, d')$, 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 $\mathcal{R}^{ON}_j[v_i]$, individually for each $j$. Denote by $(d_1, \ldots, d_m)$ the value sequence of $v_i$ along $\pi_1 \circ \cdots \circ \pi_j \circ …$ 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 $\mathcal{R}^{ON}_j[v_i](d_1, \ldots, d_m) = \{d_1, \ldots, d_m\}$ 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 $\pi^{ON}$ returned. Denote by $\mathcal{R}^{ON}$ the overall dynamic relaxation, i.e., $\mathcal{R}^{ON}$ 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 $\pi_1 \circ \cdots \circ \pi_j$ at iteration $j$, $\mathcal{R}^{ON}_j[v_i]$ constitutes our limited-memory relaxation with memory size $M_j[v_i]$. Then (ii) $\pi^{ON}$ is an $\mathcal{R}^{ON}$-relaxed plan, and (iii) $\mathcal{R}^{ON}$ 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 as returning the length of $\pi^{ON}$ – by $h_{\text{Gray}}$. In the sense of (ii) and (iii) put together, $h_{\text{Gray}}$ is more informed than $h_{RB}$.

Experiments

We implemented $h_{\text{Gray}}$ in FD (Helmert 2006). 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 $h_{\text{Gray}}$ to its direct predecessor $h_{RB}$, and to $h^B$ as a baseline. We use FD’s canonical search algorithm for satisfying 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.
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".

from disabling, as well as enabling FD’s preferred operators. For the latter, to enhance comparability, we use the same preferred operators, taken from $h^F$, for all three heuristics.  

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 $V_B$, (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), $h^G$ 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.

With preferred operators in use (left part of table), there isn’t much difference between $h^RB$ and $h^G$, except for Parprinter 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), $h^G$’s loss in Parprinter disappears; total coverage on IPC benchmarks is marginally better for $h^RB$ with preferred operators (+2), and is better for $h^G$ 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 $h^G$ is clearly superior to $h^RB$, as it does not have to paint key non-RSE-invertible variables red. With strategy (B), $h^G$ 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 $h^G$ 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


