Pushing the Limits of Partial Delete Relaxation: Red-Black DAG Heuristics

Michael Katz  
IBM Haifa Research Labs  
Haifa, Israel  
katzm@il.ibm.com

Jörg Hoffmann  
Saarland University  
Saarbrücken, Germany  
hoffmann@cs.uni-saarland.de

Abstract
Red-black planning is a systematic approach to partial delete relaxation, taking into account some of the delete effects: Red variables take the relaxed (value-accumulating) semantics, while black variables take the regular semantics. Prior work on red-black plan heuristics has identified a powerful tractable fragment requiring the black causal graph – the projection of the causal graph onto the black variables – to be a DAG; but all implementations so far use a much simpler fragment where the black causal graph is required to not contain any arcs at all. We close that gap here, and we design techniques aimed at making red-black plans executable, short-cutting the search. Our experiments show that these techniques can yield significant improvements on those IPC benchmarks where non-trivial DAG black causal graphs exist.

Introduction
The delete relaxation, where state variables accumulate their values rather than switching between them, has played a key role in the success of satisficing planning systems, e.g., (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010). Still, the delete relaxation has well-known pitfalls, for example the fundamental inability to account for moves back and forth (as done, e.g., by vehicles in transportation). It has thus been an actively researched question from the outset how to take some deletes into account, e.g., (Fox and Long 2001; Gerevini, Saetti, and Serina 2003; Helmert 2004; Helmert and Geffner 2008; Baier and Botea 2009; Cai, Hoffmann, and Helmert 2009; Haslum 2012; Keyder, Hoffmann, and Haslum 2012). Herein, we continue the most recent attempt, red-black planning (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013) where a subset of red state variables takes on the relaxed value-accumulating semantics, while the other black variables retain the regular semantics.

Katz et al. (2013b) introduced the red-black framework and conducted a theoretical investigation of tractability. Following up on this (2013a), they devised practical red-black plan heuristics, non-admissible heuristics generated by replacing fully delete-relaxed plans into red-black plans. Observing that this technique often suffers from dramatic overestimation incurred by following arbitrary decisions taken in delete-relaxed plans, Katz and Hoffmann (2013) refined the approach to rely less on such decisions, yielding a more flexible algorithm delivering better search guidance.

The black causal graph is the projection of the causal graph onto the black variables only. Both Katz et al. (2013a) and Katz and Hoffmann (2013) exploit, in theory, a tractable fragment characterized by DAG black causal graphs, but confine themselves to arc-empty black causal graphs – no arcs at all – in practice. Thus current red-black plan heuristics are based on a simplistic, almost trivial, tractable fragment of red-black planning. We herein close that gap, designing red-black DAG heuristics exploiting the full tractable fragment previously identified. To that end, we augment Katz and Hoffmann’s implementation with a DAG-planning algorithm (executed several times within every call to the heuristic function). We devise some enhancements targeted at making the resulting red-black plans executable in the real task, stopping the search if they succeed in reaching the goal. In experiments on the relevant IPC benchmarks, we find that the gained informativity often pays off, reducing search and improving overall performance.

Background
Our approach is placed in the finite-domain representation (FDR) framework. To save space, we introduce FDR and its delete-relaxation as special cases of red-black planning. A red-black (RB) planning task is a tuple $\langle V^B, V^R, A, I, G \rangle$. $V^B$ is a set of black state variables and $V^R$ is a set of red state variables, where $V^B \cap V^R = \emptyset$ and each $v \in V : = V^B \cup V^R$ is associated with a finite domain $D(v)$. The initial state $I$ is a complete assignment to $V$, the goal $G$ is a partial assignment to $V$. Each action $a$ is a pair $(\text{pre}(a), \text{eff}(a))$ of partial assignments to $V$ called precondition and effect. We often refer to (partial) assignments as sets of facts, i.e., variable-value pairs $v = d$. 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 state $s$ if $\text{pre}(a) \subseteq s[v]$ 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 $v \in V^R$ to $s[v] \cup \{\text{eff}(a)[v]\}$. By $s[\langle a_1, \ldots, a_k \rangle]$ we denote the state obtained from sequential application of $a_1, \ldots, a_k$. An action sequence $\langle a_1, \ldots, a_k \rangle$ is a plan if $G[v] \in I[\langle a_1, \ldots, a_k \rangle][v]$ for all $v \in V(G)$. 

\[\text{red-black DAG heuristics} \]
Π is a finite-domain representation (FDR) planning task if \( V^R = \emptyset \), and is a monotonic finite-domain representation (MFDR) planning task if \( V^B = \emptyset \). Plans for MFDR tasks (i.e., for delete-relaxed tasks) can be generated in polynomial time. A key part of many satisficing planning systems is based on exploiting this property for deriving heuristic estimates, via delete-relaxing the task at hand. Generalizing this to red-black planning, the red-black relaxation of an FDR task \( Π \) relative to \( V^R \) is the RB task \( Π^+_{v,R} = (V \setminus V^R, V^R, A, I, G) \). A plan for \( Π^+_{v,R} \) is a red-black plan for \( Π \), and the length of a shortest possible red-black plan is denoted \( h^+_{v,R}(Π) \). For arbitrary states \( s \), \( h^+_{v,R}(s) \) is defined via the RB task \( (V \setminus V^R, V^R, A, s, G) \). If \( V^R = V \), then red-black plans are relaxed plans, and \( h^+_{v,R} \) coincides with the optimal delete relaxation heuristic \( h^D \).

Algorithm: \textsc{RedBlackPlanning}(\Pi, R+)

\[
\pi ← ()
\]

\[
\text{while } R ⋈ R^+ \{
\begin{align*}
A' & = \{ a ∈ A | \text{pre}(a) ⊆ B \cup R, \text{eff}(a) ∩ (R^+ \setminus R) ≠ \emptyset \} \\
\text{Select } a ∈ A' \\
\text{do } \text{if } \text{pre}(a)[V^B] ⊈ I[\pi] \\
\text{then } π ← π ∪ \text{ACHIEVE}(\text{pre}(a)[V^B]) \\
π ← π ∪ \text{pre}(a)
\end{align*}
\]

\[
\text{if } G[V^B] ⊈ I[\pi] \text{ then } π ← π ∪ \text{ACHIEVE}(G[V^B])
\]

return \( π \)

Figure 2: Katz and Hoffmann’s (2013) red-black planning algorithm (abbreviated; for explanations see text).

much simpler fragment restricted to arc-empty black causal graphs in practice. In Figure 1, both \( T \) and \( F \) are RSE-invertible; if we paint only \( T \) black then the black causal graph is arc-empty, and if we paint both \( T \) and \( F \) black then the black causal graph is (not arc-empty but) a DAG.

Red-Black DAG Heuristics

As indicated, we augment Katz and Hoffmann’s (2013) implementation with a DAG-planning algorithm. To provide the context, Figure 2 shows (the main parts of) Katz and Hoffmann’s pseudo-code. The algorithm assumes as input the set \( R^+ \) of preconditions and goals on red variables in a fully delete-relaxed plan, i.e., \( R^+ = G[V^R] \cup \bigcup_{a ∈ A} \text{pre}(a)[V^R] \) where \( π^+ \) is a relaxed plan for \( Π \). It then successively selects achieving actions for \( R^+ \), until all these red facts are true. Throughout the algorithm, \( R \) denotes the set of red facts already achieved by the current red-black plan prefix \( π \); \( B \) denotes the set of black variable values that can be achieved using only red outside conditions from \( R \). We have omitted the (simple) maintenance of \( R \) and \( B \) here as it is not needed to understand the present paper.

For each action \( a ∈ A' \) selected to achieve new facts from \( R^+ \), and for the global goal condition at the end, there may be black variables that do not have the required values. For example, say we paint \( T \) and \( F \) black in Figure 1. Then \( R^+ \) will have the form \( \{ A = T, A = a, B = T, B = b, C = T, C = c, D = T, D = d \} \). In the initial state, \( A' \) contains only load actions. Say we execute \( a = \text{load}(A, \text{init}) \), entering \( A = T \) into \( R \) and thus including unload(A, a) into \( A' \) in the next iteration. Trying to execute that action, we find that its black precondition \( T = a \) is not satisfied. The call to \text{ACHIEVE}(\{T = a\}) is responsible for rectifying this.

\text{ACHIEVE}(g) creates a task \( Π^B \) over \( Π \)’s black variables, asking to achieve \( g \). As Katz and Hoffmann showed, \( Π^B \) is solvable, has a DAG causal graph, and has strongly connected DTGs (when restricting to actions \( a \) where \( \text{pre}(a) ⊆ I[\pi] \)). From this and Theorem 4.4 of Chen and Gimenez (2010), it directly follows that a plan for \( Π^B \), in a succinct

1Indeed, all optimal red-black plans (but not some non-optimal ones) then are real plans. We will get back to this below: As we shall see, the ability to increase red-black plan applicability is a main advantage of our red-black DAG heuristics over the simpler red-black plan heuristics devised in earlier work.
Algorithm: DAGPlanning(ΠB)

main
πB ← {}
for i = n downto 1
  // Denote πB = ⟨a1, . . . , ak⟩
  d ← I[v[i]]
  for j = 1 to k
    πj ← {}
    if pre(aj)<v[i] is defined
      then πj ← πv[i] · pre(aj)<v[i]
    d ← pre(aj)<v[i]
  πk+1 ← {}
  if G[v[i]] is defined
    then πk+1 ← πv[i] · d · G[v[i]]

πB ← π1 · ⟨a1⟩ · . . . · πk · ⟨ak⟩ · πk+1
return πB

Figure 3: Planning algorithm for FDR tasks ΠB with DAG causal graph CGΠB and strongly connected DTGs. v1, . . . , vn is an ordering of variables V consistent with the topology of CGΠB. πv(d, d′) denotes an action sequence constituting a shortest path in DTGv(d) from d to d′.

plan representation can be generated in polynomial time.

The “succinct plan representation” just mentioned consists of recursive macro actions for pairs of initial-value/other-value within each variable’s DTG; it is required as plans for ΠB may be exponentially long. Chen and Gimenez’ algorithm handling these macros involves the exhaustive enumeration of shortest paths for the mentioned value pairs in all DTGs, and it returns highly redundant plans moving precondition variables back to their initial value in between every two requests. For example, if a truck unloads two packages at the same location, then it is moved back to its start location in between the two unload actions.

Katz and Hoffmann (2013) shunned the complexity of DAG planning, and considered ΠB with arc-empty causal graphs, solving which is trivial. In our work, after exploring a few options, we decided to use the simple algorithm in Figure 3: Starting at the leaf variables and working up to the roots, the partial plan πB is augmented with plan fragments bringing the supporting variables into place (a similar algorithm was mentioned, but not used, by Helmer (2006)).

Proposition 1 The algorithm DAGPlanning(ΠB) is sound and complete, and its runtime is polynomial in the size of ΠB and the length of the plan πB returned.

Note here that the length of πB is worst-case exponential in the size of ΠB, and so is the runtime of DAGPlanning(ΠB). We trade the theoretical worst-case efficiency of Chen and Gimenez’ algorithm against the practical advantage of not having to rely on exhaustive computation of shortest paths – anew for every call of DAGPlanning, with “initial values” and DTGs from ΠB – for input tasks ΠB that typically have small plans (achieving the next action’s black preconditions) anyhow.2

Unlike the macro-based algorithm of Chen and Gimenez, our DAGPlanning algorithm does not superficially keep switching supporting variables back to their initial values. But it is not especially clever, either: If variable v0 supports two otherwise independent leaf variables v1 and v2, then the sub-plans for v1 and v2 will be inserted sequentially into πB, losing any potential for synergies in the values of v0 required. We developed a more flexible algorithm addressing that weakness through using a partially-ordered πB, but that algorithm resulted in significantly worse empirical performance, so we do not include it here.

Enhancing Red-Black Plan Applicability

One crucial advantage of red-black plans, over fully-delete relaxed plans, is that they have a much higher chance of actually working for the original planning task. This is especially so for the more powerful DAG red-black plans we generate here. In Figure 1, as already mentioned, if we paint just T black then the red-black plan might work: but if we paint both T and F black – moving to a non-trivial DAG black causal graph – then every optimal red-black plan definitely works. A simple possibility for exploiting this, already implemented in Katz and Hoffmann’s (2013) earlier work, is to stop search if the red-black plan generated for a search state s is a plan for s in the original task.

There is a catch here, though – the red-black plans we generate are not optimal and thus are not guaranteed to execute in Figure 1. In our experiments, we observed that the red-black plans often were not executable due to simple reasons. We fixed this by augmenting the algorithms with the two following applicability enhancements.

(1) Say that, as above, R+ = {A = T, A = a, B = T, B = b, C = T, C = c, D = T, D = d} and REDBLACKPlanning started by selecting load(A,init). Unload(A,a) might be next, but the algorithm might just as well select load(B,init). With T and F black, load(B,init) has the black precondition F = true. Calling ACHIEVE(⟨F = true⟩) will obtain that precondition using unload(A,init). Note here that variable A is red so the detrimental side effect is ignored. The same phenomenon may occur in any domain with renewable resources (like transportation capacity). We tackle it by giving a preference to actions a ∈ A getting whose black preconditions does not involve deleting R+ facts already achieved beforehand. To avoid excessive overhead, we approximate this by recording, in a pre-process, which red facts may be deleted by moving each black variable, and prefer an action if none of its black preconditions may incur any such side effects.

(2) Our second enhancement pertains to the DTG paths chosen for the black precondition variables in DAGPlanning (after REDBLACKPlanning has already selected the next action). The red outside conditions are by design all reached (contained in R), but we can prefer paths whose red outside conditions are “active”, i.e., true when executing the current red-black plan prefix in the real task. (E.g., if a capacity variable is red, then this will prefer loads/unloads that use the actual capacity instead of an arbitrary one.) In some special cases, non-active red outside conditions can be easily fixed by inserting additional supporting actions.

2One could estimate DAG plan length (e.g., using Helmer’s (2006) causal graph heuristic), computing a red-black plan length estimate only. But that would forgo the possibility to actually execute DAG red-black plans, which is a key advantage in practice.
Experiments

The experiments were run on Intel Xeon CPU E5-2660 machines, with time (memory) limits of 30 minutes (2 GB). We ran all IPC STRIPS benchmark instances whose causal graphs have at least one directed arc \((v,v')\) between RSE-invertible variables \(v\) and \(v'\), with no backwards arc \((v',v)\). These are exactly the tasks for which there exists a choice of black variables so that \((a)\) the resulting red-black planning task is inside the tractable fragment, and \((b)\) the black causal graph is a non-arc-empty DAG. The domains/instances where that happens are as shown in Table 1. For IPC'08 domains also used in IPC'11, we used only the IPC'11 version. For simplicity, we consider uniform costs throughout (i.e., we ignore action costs where specified).

We compare our DAG heuristics against Katz and Hoffmann’s (2013) arc-Empty ones, and against two variants of Keyder et al.’s (2012) partial delete relaxation heuristics: K12 is best in their published experiments, K13 is best in more recent (yet unpublished) experiments. S stops the search if a red-black plan works for the original planning task. Our baseline is the hFF heuristic implemented in Fast Downward. All configurations run greedy best-first search with lazy evaluation and a second open list for states resulting from preferred operators (Helmert 2006). All red-black heuristics return the same preferred operators as hFF: This enhances comparability; we found that changing the preferred operators was typically not beneficial anyway.

Katz and Hoffmann explored a variety of painting strategies, i.e., strategies for selecting the black variables. We kept this simple here because, as we noticed, there actually is little choice, at least when accepting the rationale that we should paint black as many variables as possible: In all our domains except Tidybot, there are at most 2 possible paintings per task. To illustrate, consider Figure 1: We can paint \(T\) and \(F\) black, or paint \(T\) and the packages black. All other paintings either do not yield a DAG black causal graph, or are not set-inclusion maximal among such paintings. We thus adopted only Katz and Hoffmann’s 3 basic strategies, ordering the variables either by incident arcs \((A)\), or by conflicts \((C)\), or by causal graph level \((L)\), and iteratively painting variables red until the black causal graph is a DAG (Katz and Hoffmann’s original strategies continue until that graph is arc-Empty). A works best for Katz and Hoffmann’s heuristics (both here and in their experiments), and L works best for ours, so we show data for these two.

Consider Table 1 from left to right. Our red-black DAG heuristics have the edge in coverage, thanks to excelling in Transport and being reliably good across these domains. For search space size (number of state evaluations, i.e., calls to the heuristic function), there are smallish differences in half of the domains, and huge differences in the other half: Under the L painting strategy, in Barman the arc-Empty heuristic already does well, and in Elevators, Gripper, and Transport our new DAG heuristic excels. A look at the “DLS Init” column, i.e., the number of instances solved by S in the initial state (without any search), shows that the latter 3 domains are exactly those where the superior applicability of DAG red-black plans makes the difference. Column “hFF/DLS Plan” shows that the plans found using DLS (even when constructed within the heuristic by S) are about as good as those found using hFF. Column “hFF/DLS Time” shows that the search space reductions do pay off in total runtime, except in Gripper (where all except K12 and K13 terminate in split seconds). The maximum speed-ups are 413 in Barman, 7.5 in Driverlog, 722 in Elevators, 12.9 in Tidybot, and 683 in Transport (none in the other domains); the maximum slow-down factors are 1.2 in Barman, 25 in Driverlog, 4.8 in Rovers, 4 in Tidybot, and 625 in Trucks.

The remainder of Table 1 sheds light on the contribution of stop search in DLS: -S switches stop search off, -(1,2) leaves it on but switches both applicability enhancements off, -(2) switches only enhancement (2) off.3 We see that stop search helps even in domains (Driverlog, Tidybot) not solved directly in the initial state, and we see that the superior coverage in Transport is half due to the more informed heuristic, and half due to stop search with enhancement (1).

To further investigate the effect of stop search, we generated additional instances of the three domains that are fully solved directly in the initial state. Table 2 summarizes the results for additional 100 instances of increasing size in each of the three domains, namely Elevators, Gripper, and Transport. Focusing on our best performer, we compare DLS to both switching stop search off and our base FF heuristic. Note that DLS still solves all, even extremely large4 instances directly in the initial state. Switching S off drastically reduces coverage without improving plan length. Comparing to FF, the picture is similar. Due to a large number of evaluations, even the much faster heuristic times out

---

3The data for -(1) is left out of Table 1 as it is equal to that for -(1,2). Enhancement (2) has more impact in other (non-DAG) domains, especially NoMystery and Zenotravel, where S solves 8 respectively 19 more tasks directly with (2) than without it.

4Largest Gripper instance has 3942 balls.
before finding a solution. Interestingly, in Gripper, DL has linear search space\(^5\). However, even in such case, due to the costly per node evaluation time of DL, large enough instances are not solved under the 30 minutes time bound.

### Conclusion

Our work provides one more step on the road towards systematic interpolation between delete-relaxed and non-relaxed (real) planning. Our experience, as reflected by the presented experiments, suggests that the key advantage of heavy interpolation may lie in the ability to produce approximate plans that are very close to real plans (or that already are real plans). We believe that further progress will be achieved by developing search methods exploiting that ability in a targeted manner, for example by using partially relaxed plans to initialize plan-space searches (e.g., (Nguyen and Kambhampati 2001; Gerevini, Saetti, and Serina 2003)).

### References


---

\(^5\) So does FF heuristic when using enhanced hill climbing search and helpful actions in FF planning system.