Django: Unchaining the Power of Red-Black Planning

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

Abstract

Red-black planning is a powerful method, allowing in principle to interpolate smoothly between fully delete relaxed planning, and real (completely unrelaxed) planning. Alas, the method has been chained to the use as a heuristic function, necessitating to compute a red-black plan on every search state, entailing an exclusive focus on tractable fragments. The Django system unleashes red-black planning on the problem of proving unsolvability *within the red-black relaxation*. We introduce red-black state space search that can solve arbitary red-black planning problems, and we prove unsolvability by iteratively painting more and more red variables black.

Introduction

Red-black planning (Katz, Hoffmann, and Domshlak 2013b) interpolates between fully delete relaxed planning, and real (completely unrelaxed) planning, by selecting a subset of state variables – the "red" ones – which take the delete-relaxed semantics, accumulating their values; while the remaining state variables – the "black" ones – retain the original value-switching semantics. If all variables are red, we have a delete relaxation, if all variables are black, we have the original planning task. In between we have a hybrid *red-black relaxation* more informed than the delete relaxation.

The method has so far been used for the design of heuristic functions (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013; Gnad and Hoffmann 2015b; Domshlak, Hoffmann, and Katz 2015), computing a red-black plan on every search state akin to the wide-spread relaxed plan heuristic (Hoffmann and Nebel 2001). Naturally, this entails an exclusive focus on tractable fragments of red-black plan generation. Our key observation in the Django system is that red-black relaxation can be useful also for proving unsolvability *within the relaxation*. This is promising 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, 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.

Django therefore unleashes the power of red-black planning, through *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. Given this hybrid, we can prove unsolvability by fixing a variable order, and then, starting with all variables being red, painting more and more variables black until the red-black relaxation is unsolvable.

On the unsolvable benchmarks introduced by Hoffmann et al. (2014), this method excels in 3 domains and thus, overall, substantially improves the state of the art, at least when using our new better variable ordering strategy, not the old one that we had designed at IPC planner submission time. The authors are curious to see how much luck Django will have with whatever benchmarks will be used in the Unsolvability IPC 2016. But whatever happens, Django, remember: After the showers, the sun will be shining . . . [1]

Django

Framework

Django is implemented on top of FD (Helmert 2006) (who would have guessed!). It uses the mutex-optimized preprocessor by Alcazar and Torralba (2015) to get an optimized finite-domain variable encoding.

[1] For the reader looking for an algorithm description fitting "Django" as an acronym: there is none. We just like the movie.

Obviously we're not going to go into tremendous detail here, but let it be said that we use the *finite-domain representation (FDR)* framework, notating *planning tasks* as $\Pi = (V, A, I, G)$. V is a set of finite-domain *state 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 pair $(\mathsf{pre}_a, \mathsf{eff}_a)$ of the action's *precondition* pre_a and *effect* eff_a, each a partial assignment to V.

The semantics of a planning task Π is defined in terms of its *state space*, which is a (labeled) *transition system* $\Theta_\Pi = (S, T, s_0, S_G)$ defined in the usual manner, S being the set of all states, T being the transitions given by the actions A, s_0 being the initial state, and S_G being the goal states. A plan is a path from s_0 to some state in S_G. We want to prove that no plan exists.

Red-Black Planning

The delete relaxation 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* is a 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. *Red-black states* s^RB assign each variable v a subset $s^\mathsf{RB}(v) \subseteq D_v$ of its possible values. Initially, in the *red-black initial state*, the value subset contains the single value $I(v)$. If v is a black variable, then action effects on v overwrite v's previous value, so that $s^\mathsf{RB}(v)$ always contains exactly one element; if v is a red variable, then action effects on v are accumulated into the previous value subset. A *red-black goal state* is one where, for every goal variable v, $G(v) \in s^\mathsf{RB}(v)$.

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.

Red-Black State Space Search

Red-black planning generalizes both, delete-relaxed planning and real planning, so in particular deciding red-black plan existence is, in general, **PSPACE**-hard. To solve arbitrary red-black planning problems, we need a search algorithm: red-black state space search.

Essentially, the search branches only over those actions affecting black variables, while handling the other actions through red forward fixed points associated with individual state transitions. To keep this paper crisp, we give an outline only, and we refer the masochistic and/or interested reader to our SOCS'16 paper for the details (Gnad et al. 2015).

Like typical relaxed planning algorithms, red-black state space search consists of a forward phase, followed by a backward phase. The forward phase chains forward until reaching the goal ("state space search with a relaxed planning graph at each transition"), and the backward phase extracts a red-black plan ("extracting the solution path with a relaxed plan extraction step at each transition").

It is cumbersome to spell this out formally. But it should be possible to get an intuition across. Without actually introducing the notations, consider this (slightly simplified) definition from our SOCS'16 paper:

Definition 1 (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, 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:*

- *(i) S^RB is the set of all red-black states.*
- *(ii) s^RB_0 is the red-black initial state.*
- *(iii) S^RB_G contains the red-black states s^RB where $\mathcal{F}^+(s^\mathsf{RB})$ is a red-black goal state.*
- *(iv) T^RB is the set of transitions $s^\mathsf{RB} \xrightarrow{a} t^\mathsf{RB}$ where a affects at least one black variable, a is applicable to $\mathcal{F}^+(s^\mathsf{RB})$, and $t^\mathsf{RB} = \mathsf{outcomeState}(\mathcal{F}^+(s^\mathsf{RB}), a)$.*

The notation "$\mathcal{F}^+(s^\mathsf{RB})$" denotes the extension of the red-black state s^RB with all those values of red variables that can be reached from s^RB by applying actions with red effects only. In other words, $\mathcal{F}^+(s^\mathsf{RB})$ adds, into the value subsets $s^\mathsf{RB}(v)$ of the red variables v, the red-planning fixed point ("delete-relaxed fixed-point layer in a relaxed planning graph"), when considering only those red-effect actions whose black preconditions are satisfied in s^RB.

Given this, item (iii) just says that we can stop at s^RB if its red fixed point contains the goal. Item (iv) says that, to transition from one red-black state s^RB to another t^RB via action a, we first execute the red fixed point on s^RB, to obtain $\mathcal{F}^+(s^\mathsf{RB})$; then we check whether a is applicable to $\mathcal{F}^+(s^\mathsf{RB})$; and if so, we simply apply a to that fixed point, treating $\mathcal{F}^+(s^\mathsf{RB})$ like any other red-black state.

Say now that the forward phase has found a path to the goal, i. e., a path $\pi = \langle s^\mathsf{RB}_0, a_0, s^\mathsf{RB}_1, \ldots, a_{n-1}, s^\mathsf{RB}_n \rangle$ in Θ^RB_Π, where $s^\mathsf{RB}_n \in S^\mathsf{RB}_G$. In standard state space search, we would simply return the actions $a_0, \ldots, a_{n-1}$ labeling the path. But in our case here, that would account only for the black-affecting actions. To collect the red-affecting actions, at each transition $s^\mathsf{RB}_i \xrightarrow{a} s^\mathsf{RB}_{i+1}$ along π, we need to extract a red plan supporting the subgoals needed at time $i+1$, propagating new needed subgoals to time i. The subgoals needed at time n are simply the red goals; each red-plan extraction step is a standard relaxed plan extraction step on the red fixed point leading from i to $i+1$; once we reach time 0, we can schedule all the red plans along $a_0, \ldots, a_{n-1}$ and have a red-black plan.

The reader might have noticed that the author just got carried away – this paper, competition, and planning system being exclusively about proving unsolvability, we will never actually get to the backward red-black plan extraction phase, or if we do, then we know that the relaxation is not informed enough and we need to paint more variables black. Apologies for the inconvenience; then again, the backward phase is part of red-black state space search, and that search also

has other possible uses (cf. our SOCS'16 paper), so the author is right now choosing to just leave this in.

In any case, coming back to what does matter for our purpose here: it is easy to see that, if the goal cannot be reached in $\Theta^{\mathsf{RB}}_{\Pi}$, then Π is unsolvable. This is simply because red-black relaxation preserves plans, and goal reachability in $\Theta^{\mathsf{RB}}_{\Pi}$ is equivalent to red-black plan existence.

Wrapping it Up with a Variable Ordering Strategy

To turn the above into an actual automatic planner, we need to decide how to actually paint the variables – which ones are to be red, which ones are to be black?

Previous work designed such *painting strategies* for the purpose of heuristic functions. For the purpose of proving unsolvability, matters are different in that it makes a lot of sense to merely *try* a painting, and, if it does not succeed, try another one. The simplest possible way to do this – or at least these authors could not think of a simpler one – is to start with all variables being red, then iteratively check whether there is a red-black plan; if no, stop (unsolvability proved); else, pick a red variable v, paint it black, and iterate. The question then just remains how to pick the next variable.

At the time of planner submission, the authors simplified even this simple question, fixing a variable order a priori, not taking into account any new information found during the process. Specifically, we used a variable ordering strategy that we denote as *RBb*, the "b" standing for breadth-first (we leave it to the reader's imagination what the "RB" may be for). The strategy builds the DAG of strongly connected components (SCC) of the input task's causal graph, and processes these (i. e., orders the variables) in a breadth-first manner, from root SCCs to leaf SCCs.

We later on realized that it is actually a good idea to take information found during the process into account, specifically *conflicts* in the red-black plan found in the previous iteration. The notion of conflicts is inspired by painting strategies underlying heuristic functions (Domshlak, Hoffmann, and Katz 2015). Given a red variable v, a conflict on v is an action in the red-black plan whose precondition on v would not be satisfied when painting v black. The idea is to select, as the next red v to be painted black, one with a maximal number of conflicts. We denote this by *RBc*, and we denote by *RBbc* the strategy that applies RBb and breaks ties, for inclusion of the next variable within an SCC, by the maximal number of conflicts.

And this is all there is to say about Django ...

... except, catering for the unlikely case where Django does not work on the benchmarks wisely chosen by the IPC organizers, let us show off a little bit with our results on the previous benchmarks by Hoffmann et al. (2014):

Own Experiments

Table 1 shows coverage data, i. e., the number of instances proved unsolvable. 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

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.

BDDs (Torralba and 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 – we should really say: represent*ed* the state of the art at planner submission time – 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, until RBbc terminates; if the projection onto the black variables is at this point still solvable, then BP continues with the RBb 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.[2]

Consider finally Figure 1, a direct comparison between red-black state space search and black-projection, as the set

[2]Remark by the author: Isn't it great how one can bash one's own work in one's own papers?

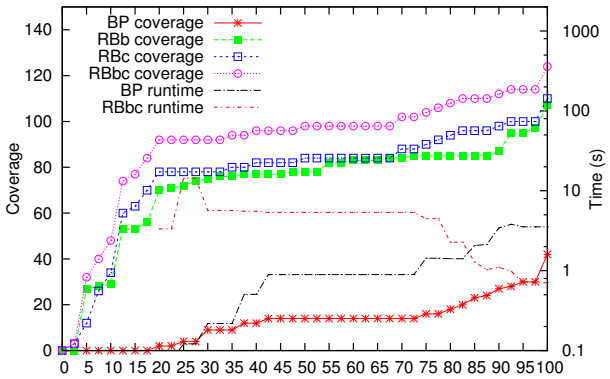

Figure 1: Coverage and average runtime of red-black state space search, compared to black-projection, as a function of the fraction of black variables. Explanations see text.

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.

Conclusion

Django is unchained! Red-black planning has finally escaped the cage of computational tractability! What more is there to say?

Well, let us say that this is the beginning, not the end, of the story (oops I almost said "movie" here). Django can still be improved in a gazillion ways, including but not limited to: better variables ordering strategies; re-using state space information (e. g. dead-end regions) from previous iterations; adaptive paintings choosing red/black variables depending on state; etc. It should also be noted that Django is not doomed to just prove unsolvability – if the red-black plan in some iteration happens to be a real plan, then we can also stop. The question then is how to fruitfully interleave both purposes, choosing the next black variable, perhaps, based on the current hypothesis whether the task will turn out to be solvable or unsolvable.

Figure 2: THE END.

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

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press.

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 29–37. AAAI Press.

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

Gnad, D., and Hoffmann, J. 2015a. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press.

Gnad, D., and Hoffmann, J. 2015b. Red-black planning: A new tractability analysis and heuristic function. In Lelis, L., and Stern, R., eds., *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*. AAAI Press.

Gnad, D.; Steinmetz, M.; Jany, M.; Hoffmann, J.; Serina, I.; and Gerevini, A. 2015. Partial delete relaxation, unchained: On intractable red-black planning and its applications. In Baier, J., and Botea, A., eds., *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS'16)*. AAAI Press.

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.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T., ed., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*. Prague, Czech Republic: IOS Press.

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 105–113. AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In desJardins, M., and Littman, M., eds., *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*, 489–495. Bellevue, WA, USA: AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. 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.

Torralba, A., and Alcázar, V. 2013. Constrained symbolic search: On mutexes, BDD minimization and more. In Helmert, M., and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 175–183. AAAI Press.