Traps, Invariants, and Dead-ends

Nir Lipovetzky
University of Melbourne
Melbourne, Australia
@unimelb.edu.au*

Christian Muise
MIT CSAIL
Massachusetts, USA
cjmuise@mit.edu

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

We consider the problem of deriving formulas that capture traps, invariants, and dead-ends in classical planning through polynomial forms of preprocessing. An invariant is a formula that is true in the initial state and in all reachable states. A trap is a conditional invariant: once a state is reached that makes the trap true, all the states that are reachable from it will satisfy the trap formula as well. Finally, dead-ends are formulas that are satisfied in states that make the goal unreachable. We introduce a preprocessing algorithm that computes traps in k-DNF form that is exponential in the k parameter, and show how the algorithm can be used to precompute invariants and dead-ends. We report also preliminary tests that illustrate the effectiveness of the preprocessing algorithm for identifying dead-end states, and compare it with the identification that follows from the use of the h^1 and h^2 heuristics that cannot be preprocessed, and must be computed at run time.

1 Introduction

The detection of dead-ends, states from which the goal is not reachable, is a crucial ability in many forms of planning, including classical, non-deterministic, and probabilistic planning (Hoffmann 2011; Junghanns and Schaeffer 1998; Kolobov et al. 2011; Muise, McIlraith, and Beck 2012). Indeed, a plan or policy that must reach the goal with certainty must definitely avoid reaching dead-end states. In spite of the importance of dead-ends in planning, however, there has not been much work in the area. State-of-the-art classical planners do not feature specific mechanisms for detecting dead-ends, and recognize dead-ends implicitly, in the delete relaxation only, when heuristic estimators based on the delete relaxation yield infinite values. Heuristics that are not based on the delete relaxation, like the h^m heuristics (Haslum and Geffner 2000) and variations for $m > 1$, can detect a larger class of dead-ends, but these heuristics are expensive and cannot be computed at run time for every new node in the (forward) search, and thus such heuristics have been used mainly for detecting unsolvability, that is, when the initial problem state is a dead-end (Helmert 2006; Lipovetzky and Geffner 2011; Bäckström, Jonsson, and Ståhlberg 2013; Hoffmann, Kissmann, and Torralba 2014).

*firstname.lastname

The situation is different for invariants (Rintanen 2000; Gerevini and Schubert 2001). Invariants are formulas that are true in all reachable states, and they are usually computed at preprocessing. Two types of invariants that are commonly used are mutex constraints (Blum and Furst 1997), i.e., invariants of the form $\neg(p_1 \wedge p_2)$ for atoms p_1 and p_2, and disjunctive constraints (Helmert 2006), i.e., invariants of the form $p_1 \vee p_2 \vee \cdots \vee p_n$. Together such invariants yield the multivalued variables that are extracted from boolean problem representations (Helmert 2006).

In this paper, we establish a relation between invariants and dead-ends, and develop a preprocessing algorithm for computing them both. The key concept in our analysis will be the notion of *traps*. Syntactically, traps are formulas that stand for *conditional invariants*; namely, once a state is reached that satisfies a trap formula, all the states that can be reached from that state will satisfy the formula as well. A standard invariant is thus a trap that is true in the initial state, while a trap that is mutually exclusive with the goal denotes a dead-end formula; a formula that is true in dead-end states only. As a result, the proposed algorithm for computing traps is also an algorithm for computing invariants and dead-ends. We propose an algorithm that computes traps in k-DNF form: trap formulas in disjunctive normal form whose terms have at most k atoms. The algorithm is simple and clean, operates as a preprocessing step, and runs in time that is exponential in the k parameter. We report also preliminary experimental tests that illustrate its effectiveness for identifying dead-end states, and compare it with the identification that follows from the use of the h^1 and h^2 heuristics that cannot be preprocessed, and must be computed at run time.

2 Logic

The above notions are all simple, but it is worth making them explicit before presenting the algorithm. We assume a classical planning problem P where states provide an interpretation to the formulas in that language. A state s is reachable from another state s' (in the problem P) if there is an action sequence (possibly empty) that maps s into s'. A state is reachable if it is reachable from the initial state s_0 of P.

Definition 1 (Invariant) *An invariant is a formula that is true in all reachable states.*

Definition 2 (Dead-end) *A dead-end formula is a formula that is true only in states from which no goal states are reachable.*

Definition 3 (Trap) *A trap is a formula such that if a state s satisfies the formula, all states reachable from s satisfy the formula as well.*

The notion of a trap generalizes the notion of invariant, and connects it with dead-end formulas:

Theorem 4 (Invariant Trap) *A formula is an invariant if and only if it is a trap that is true in the initial state.*

Theorem 5 (Dead-end Trap) *If the terms t_i in a DNF trap $t_1 \vee t_2 \vee \cdots \vee t_n$ are* mutually exclusive *with the (conjunctive) goal, then the trap is a dead-end formula.*

Recall that two atoms p_1 and p_2 are said to be mutually exclusive (mutex) when $\neg(p_1 \vee p_2)$ is an invariant, and there are effective algorithms for computing such mutexes.

These simple theorems indicate how the computation of traps can be used for deriving invariants and dead-end formulas. For obtaining dead-end formulas, we compute DNF traps whose terms are mutex with the goal. For obtaining invariants, select the traps that are true in the initial state. Furthermore, if a trap is both an invariant and a dead-end formula, it means that the problem is not solvable.

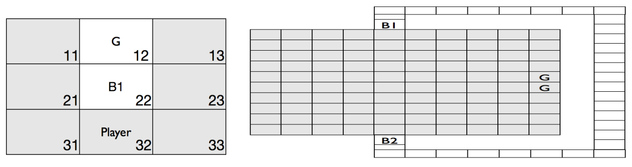
3 Algorithm

The algorithm for computing traps in k-DNF is conceptually simple and runs in time that is exponential in the parameter k. We call it *Trapper*. Trapper builds a graph whose nodes are the tuples B_i of at most k atoms in the problem, along with a dummy node D. For computing dead-end formulas, the only tuples B_i considered are the ones that are mutex with the goal. Otherwise, neither the goal or the initial state of the problem play a role in the construction of the graph that follows exclusively from the action descriptions. We assume STRIPS actions a for simplicity with precondition, add, and delete lists $Pre(a)$, $Add(a)$, $Del(a)$. The k-trap graph, or simply trap graph, is constructed as:

1. Initialize the k-trap graph whose nodes are the tuples B_i of at most k atoms not mutex with one another, along with the dummy node D. For computing dead-end formulas, the only tuples considered are those mutex with the goal.
2. Consider action a applicable in node B_i if $Pre(a)$ is not mutex with B_i
3. For an action action a applicable in B_i define the a-children of B_i as the nodes B_j such that B_j is contained in the progression of B_i through the action a defined as $B_j \subseteq ((B_i \cup Pre(a)) \setminus Del(a)) \cup Add(a)$.
4. For an action a applicable in B_i, if the set of a-children of B_i is empty, set D as the only a-child of B_i.

Once the trap graph is built, its nodes are marked starting with the dummy node D after the following propagation:

- Mark the dummy node D.
- Mark node B_i if there is an action a applicable in B_i such that *all* the a-children of B_i are marked.



(a) (b)

Figure 1: The left sub-Figure illustrates a Sokoban instance where block B1 has to be pushed towards the goal location G. If the block is pushed towards any of the grey locations the problem becomes unsolvable. In the right Subfigure blocks B1 and B2 need to be pushed towards the goal positions G. The player can be located in any grey location.

- Terminate when no more nodes can be marked.

If non-empty, the set of unmarked nodes represents a trap:

Theorem 6 (Computation of Traps) *Let T stand for the k-DNF formula whose terms are the tuples B_i associated with the nodes that have not been marked at the end of the above process. Then T is a trap.*

Proof. If all nodes are marked, T is empty (i.e., false), and is trivially a trap as no states satisfies it. Otherwise, we show that the following conditional invariant must hold: if a state entails a disjunct $B_i \in T$, then for any action a applicable in s, the progression s' of s through a entails some $B_j \in T$. Indeed, B_i is not marked as $B_i \in T$, from the marking procedure, there must be an a-child of B_i, call it B_k, that is not marked either and belongs to T as well. But if B_k is an a-child of B_i, from the definition of a-children (cf. step 3) and the fact that action a must be applicable in B_i if it is applicable in s (cf. step 2), it follows that B_k must hold in s'. This means that the conditional invariant holds for $j = k$ and hence that T is a trap. □

The application of this result to the computation of invariants and dead-end traps is direct, from the discussion above. In the experiments below we focus on the computation of dead-end formulas only:

Theorem 7 (Computation of Dead-end Traps) *Let the tuples B_i in the k-trap graph be restricted to those that are mutex with the goal, and let T be as in Theorem 6. Then T is a dead-end trap.*

In problems where there is an atom representing the negation of each fluent, the algorithm will be complete when k is set to the number of fluents.

4 Example

We illustrate the traps derived in the Sokoban domain. Sokoban consists of a player pushing each block towards a goal location. Given that blocks can only be pushed, not pulled back, it is easy to fall into dead-ends.

Consider the example of Figure 1a, where a single block is located in the middle (2,2) of a 3x3 grid. The goal is to push the block up into location (1,2) from location (2,2), but if the player pushes the block towards any other location, the problem becomes unsolvable. To capture all the dead-end

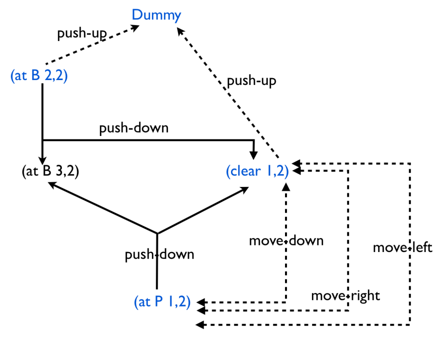


Figure 2: Partial k-trap graph of instance shown in Figure 1a. Dashed-edges propagate the mark from $Dummy$. The B_i in blue are marked at the end of the propagation.

states we just need to consider B_i tuples of size 1 and derive the 1-trap $\vee_{x \in Pos \setminus \{(1,2),(2,2)\}}\{(at\ block\ x)\}$, where Pos is the set of locations in the grid. This example shows how in preprocessing our approach can detect the same dead-end states that h^1 and h^2 would detect if they are computed at every state in the search. To illustrate how we can capture dead-end states that are not detected by either h^1 or h^2, consider the example in Figure 1b. In this example a large narrow passage connects the bottom area of the main grid with its top area. Two blocks are considered; one located initially at the entrance and the other at the exit of the passage. The goal is to move these blocks into the middle-right area of the main grid. If the player is initially located at any location of the grid, marked in grey in the Figure, the problem becomes unsolvable. The solvable instances need to place the player in the passage between the blocks. h^1 and h^2 would not detect any of the unsolvable instances, as the entrance and exit positions become free in their relaxation once the blocks are pushed. In order to capture all the unsolvable instances we need the 3-DNF formula $\{\vee_{x \in Grid;\ y_1,y_2 \in Psg;\ y_1 \neq y_2}\{(at_p x),(at_{s1} y_1),(at_{s2} y_2)\}\quad\}$ where Psg stands for the passage and $Grid$ for the main area. Note that the 2-DNF formula $\{\vee_{y_1,y_2 \in Psg;\ adj(y_1,y_2)}\{(at_{s1} y_1),(at_{s2} y_2)\}\quad\}$ would capture only the dead-ends when the two blocks are in adjacent positions within the passage. These dead-ends would be captured by h^2, but not by h^1.

We are going to show how we derive the 1-DNF formulae for the first example by first identifying the set of B_i's of size 1 that are mutex with the goal (Theorem 5). We will then build the k-graph and perform the propagation from the $Dummy$ node to finally identify the B_i's that *have not been marked as part of the trap*. The set of B_i mutex with the block being at position (1,2) (i.e., the goal location) are $(at\ block\ x)$ for every position that is not (1,2), $(at\ player\ 1,2)$ as the player cannot be in the goal location at the same time as the block, and $(clear 1,2)$ as it states that no block is located at the goal position.

The k-Graph will contain the aforementioned set of B_i and hyperedges connecting each B_i to a set of B_j's, labeled by a, if the B_j's are the a-children of B_i in the k-Graph. That is, we now have to add a hyper-edge from each B_i to each B_j entailed by the progression of B_i through every applicable action a. A partial trap graph is shown in Figure 2. Let's consider the $B_i = \{(at\ player\ 1,2)\}$ (at goal location). The applicable actions are *move down,right or left* and *push-down block*. The progression through *move-down* is entailed by only one B_j, namely $\{(clear\ 1,2)\}$. Therefore a hyper-edge from $\{(at\ player\ 1,2)\}$ to $\{(clear\ 1,2)\}$ will be added. A hyper-edge towards $\{(clear\ 1,2)\}$ is added as well through the progression of action *move-right* and *move-left*. Consider now the progression through action *push-down*. The resulting partial state is $\{(at\ block\ 3,2),(at\ player\ 2,2),(clear\ 1,2)\}$. This partial state is entailed by two B_j's, $\{(clear\ 1,2)\}$ and $\{(at\ block\ 3,2)\}$, so a hyper-edge from $\{(at\ player\ 1,2)\}$ to $\{(clear\ 1,2)\}$ and $\{(at\ block\ 3,2)\}$ will be added to the graph. The only two B_i's that have an edge to the $Dummy$ node among other edges to other B_j are $\{(at\ block\ 2,2)\}$ and $\{(clear\ 1,2)\}$. The progression through action *push-up block* of both B_i's result in partial state $\{(at\ block\ 1,2),(at\ player\ 2,2),(clear\ 2,3)\}$, which is not entailed by any other B_j, as all the single fluents are not mutex with the goal. The other edges relevant to the propagation phase are the ones from $\{(clear\ 1,2)\}$ to $\{(at\ player\ 1,2)\}$. These edges result from the other applicable actions on $\{(clear\ 1,2)\}$, besides *push-up block* leading to $dummy$. These actions are *move down, right or left* towards $\{(clear\ 1,2)\}$.

Once the k-Graph is fully generated, the propagation starts marking the $Dummy$ node, and then marking nodes that have a hyper-edge with all its children marked. This process is performed until a fix point. Figure 2 shows the relevant hyper-edges from the graph that propagate the mark and the resulting B_i that end-up marked. The tuples $\{(at\ block\ 2,2)\}$ and $\{(clear\ 1,2)\}$ are marked as they have a hyper-edge to the $dummy$ node. The next B_i marked is $\{(at\ player\ 1,2)\}$ as either of the hyper-edges from action *move down,right or left* have all their children marked, namely $\{(clear\ 1,2)\}$. No other node is marked. The final trap in $k - DNF$ will be made by the remaining unmarked B_i, i.e. $\vee_{x \in Pos \setminus \{(1,2),(2,2)\}}\{(at\ block\ x)\}$.

5 Preliminary Evaluation

We evaluated the algorithm for deriving traps over a subset of benchmarks known to contain dead-ends. All algorithms are implemented using the LAPKT toolkit (Ramirez, Lipovetzky, and Muise 2015). In theses experiments we consider only 1-DNF and 2-DNF formulas, named 1-traps and 2-traps respectively. Processes are limited to 1h of running time (including trap construction) and 2GB of RAM, and all experiments were conducted on a Linux desktop with 2.5GHz Intel Processors. We derive mutex fluents computing h^2 once from the initial state. Averages per domain are computed among commonly solved instances by all approaches that solve at least 1 problem. The first experiment, summarized in Table 1, aims to assess empirically the power of the k-DNF formulae for detecting dead-ends. We expand the same 10,000 states in a Breadth-First search (BRFS) for all methods and compare the number of dead-ends captured by the traps, derived in preprocessing, with respect to the number of dead-ends captured by h^1 and h^2 computed at ev-

	h^1		1-trap		h^2		2-trap	
	T	Prn	T	Prn	T	Prn	T	Prn
Airport (50)	1.6	640	**0.8**	195	152.9	**683**	481.7	665
Floortile-sat14 (20)	4.7	7278	**0.9**	6295	177.5	7777	27.5	**9549**
Mystery (30)	8.9	921	**1.1**	2399	720.3	**3378**	236.9	2526
Parc-sat11 (20)	6.4	5874	**1.3**	8695	1362.9	8154	233.52	**9155**
Pegsol-sat11 (20)	0.9	461	**0.4**	0	42.5	**1155**	7.3	0
Sokoban-sat11 (20)	3.9	2681	**0.9**	2472	589.6	**2776**	168.5	2683
Trucks (30)	13.3	18663	**0.8**	18650	346.3	**18688**	3.9	18663
Wood-sat11 (20)	2297.0	**299820**	**20.0**	**299820**	-	-	-	-
Avg. Time	584.2		6.5	(0.1)	848.2		289.9	(286.7)

Table 1: Times (sec.) and number of nodes generated that are pruned in a BrFS search cut after expanding 10,000 nodes using h1, 1-traps, h2, and 2-traps. Numbers are Blue when x-trap outperforms h^x. In Bold the best performers.

LM-Cut	LM-Cut-1	LM-Cut-2	BRFS	1-trap		2-trap	
#S/T	#S/T	#S/T	#S/T	#S/T	G	#S/T	G
24 / 1.7	**24** / 3.2	16 / 382.8	18 / 1.8	19 / 1.2	1.86	15 / 375.6	3.97
2 / 1965.5	2 / 1944.1	5 / 1839.7	0 / 0	0 / 0	–	**8** / 86.6	–
15 /6.6	**26** / 11.0	22 / 125.5	13 / 10.1	24 / 5.0	1	18 / 99.5	1
3 / 56.4	**3** / 45.2	**3** / 95.3	0 / 0	0 / 0	–	1 /113.3	–
15 / 68.0	**15** / 72.1	**15** / 79.0	**15** / 6.0	**15** / 4.5	1	**15** / 10.07	1
7 / 220.3	**7** / 184.8	6 / 174.4	2 / 14.4	**7** / 3.8	3.83	**7** / 26.7	4.98
6 / 0.3	**7** / 0.2	**7** / 0.5	2 / 9.4	4 / 0.2	101.7	5 / 0.7	733.44
1 / 0.1	**1** / 0.1	**1** / 0.2	**1** / 1.0	**1** / 0.4	1	**1** / 0.3	1
73 / 386.5	**86** / 376.9	76 / 449.6	51 / 10.7	70 / **3.8**		70 / 178.2	

Table 2: Performance of LM-Cut (1,2 traps), plain BRFS, BRFS + 1-trap or 2-trap pruning. #S stands for the number of solved instances, T as avg. time in sec., and G as the reduction factor in terms of generated states w.r.t plain BRFS (e.g., half of the nodes are generated w.r.t BRFS when $G = 2$).

ery state. 1-traps detects more dead-end states than h^1 in 2 out of 8 domains, 2-traps in 3 domains compared to h^2, and 6 domains compared to h^1. $\{1,2\}$-traps are empty in Pegsol, and both 2-traps and h^2 run out of time in Woodworking. 1-traps is by far the fastest approach expanding the 10,000 states, spending on average 0.1 out of 6 seconds computing traps, while 2-traps spends 286.7 out of 289.9 seconds.

As it can be seen from the table, the precomputed k-traps and the h^k heuristics computed at run-time do not identify the same dead-end states, even if they both reason with tuples of k atoms. We can see the difference with a simple example. Let's assume that we have a STRIPS problem encoding two multivalued variables X and Y that can take three values 1,2,3, starting with values 1. The goal is for both variables to have value 3. The STRIPS actions allow to increase the value of a variable by 1, but the action that increases X to 3 resets Y to 1, and similarly, the action that increases Y to 3, resets X to 1. The problem is thus unsolvable, yet the heuristic h^1 will ignore the deletes and hence the resets, and yield a value of 2 for the heuristic at the initial state. On the the other hand, it's easy to see that the 1-DNF formula whose terms are the atoms X=1, X=2, Y=1, and Y=2 form a 1-traps. This is because all the terms (atoms) are mutex with the goal, and in addition, every action that deletes one atom in this set, adds another atom in the set as well, including the actions that add X=3 or Y=3.

The power of 2-traps in Floortile has a substantial impact in our second experiment, summarized in Table 2. We explore a preliminary use of the k-DNF formulae in optimal planning, and evaluate how a plain BRFS benefits from the traps pruning dead-ends as well as the optimal planner A^* + LM-Cut heuristic (Helmert and Domshlak 2009), as a reference of a high performance optimal planner[1]. 1-traps and 2-traps improve the coverage over BRFS or LM-CUT across all the benchmarks but Woodworking and Pegsol. In the former, even if we trap dead-ends, it is not enough to increase coverage, while in the latter we derive only the empty trap, false. In Airport, while 1-traps solves 1 more instance than BRFS, 2-traps solves less due to the sheer number of B_i's, timing out during the 2-traps construction. In Trucks we witness the biggest pruning impact, 1-traps reduces the search by a factor of 101.70, and 2-traps by 733.44, solving 2 and 3 more instances than BRFS respectively. Another domain where 2-traps has a huge impact is in Floortile where it is the only approach that manages to solve 8 instances, while BRFS and 1-traps solves none, and LM-Cut solves 2. The only unsolvable tasks were 9 instances of Mystery, 2 instances detected by the FD-parser in preprocessing. Among the remaining 7 unsolvable instances 1-traps directly prunes all the children of the initial state, while 2-traps prunes 4 and times-out in 3. Overall, LM-Cut with 1-traps pruning has higher coverage, while BFRS with 1-traps is the fastest.

[1]Costs are ignored in LM-Cut as we compare with BRFS.

It remains as future work to explore how to enhance other classical and non classical planners. One avenue is to use the traps $B_1 \ldots B_n$ to make the negation of these traps an invariant of the classical problem, E.g. if the B_i's have size 2, one adds the negation of one of the 2 atoms to the preconditions of actions that add the other atom. These DNFs can be compiled into classical problems to enforce their negation in all reachable states.

6 Summary

The notion of traps has been explored before in the context of non-deterministic and probabilistic planning (Ramírez and Sardiña 2014; Kolobov et al. 2011; Keyder and Geffner 2008), but key differences are that our approach is purely syntactic, executed in preprocessing only (unlike heuristics h1 and h2), and polynomial for fixed k.

For the future, we would like to explore the relation between traps, dead-ends, and different heuristics such as critical path heuristics like h^m, and pattern database heuristics. For example, it is trivial to show that if s is a dead-end in the delete relaxation (i.e., $h^1(s) = \infty$), then the conjunction made up of the negations of the atoms not reachable from s in the relaxation that includes some goal atom is a trap, although not necessarily a k-trap for a bounded k. Also, if the projection of a problem on a subset DB of k multivalued variables (Edelkamp 2001), has optimal cost $h^*_{DB}(\hat{s}) = \infty$, where $\hat{s}$ is the projection of the state s over such variables, then clearly s is a dead-end that can be identified by a k-trap defined over such variables. There is thus much to be learned about these relationships, and much to be gained computationally given the importance of dead-end detection in a variety of planning models.

References

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013.*

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial intelligence* 90(1):281–300.

Edelkamp, S. 2001. Planning with pattern databases. In *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*.

Gerevini, A. E., and Schubert, L. 2001. Discoplan: an efficient on-line system for computing planning domain invariants. In *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 140–149.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In *21st European Conference on Artificial Intelligence (ECAI 2014)*, 441–446.

Hoffmann, J. 2011. Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal of Artificial Intelligence Research (JAIR)* 155–229.

Junghanns, A., and Schaeffer, J. 1998. Single-agent search in the presence of deadlocks. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 1998)*, 419–425.

Keyder, E., and Geffner, H. 2008. The hmdpp planner for planning with probabilities. *Sixth International Planning Competition at (ICAPS 2008)*.

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H. 2011. Heuristic search for generalized stochastic shortest path mdps. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 154–161.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Ramírez, M., and Sardiña, S. 2014. Directed fixed-point regression-based planning for non-deterministic domains. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. `http://lapkt.org/`. Accessed: 2016-03-11.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, 806–811.