Minimal Landmarks for Optimal Delete-Free Planning

Article *in* Proceedings of the International Conference on Automated Planning and Scheduling · May 2012

DOI: 10.1609/icaps.v22i1.13534

3 authors:

Patrik Haslum
Australian National University
67 PUBLICATIONS **2,178** CITATIONS
SEE PROFILE

John K. Slaney
Australian National University
133 PUBLICATIONS **2,191** CITATIONS
SEE PROFILE

Sylvie Thiebaux
Australian National University
137 PUBLICATIONS **2,561** CITATIONS
SEE PROFILE

Minimal Landmarks for Optimal Delete-Free Planning

Patrik Haslum, John Slaney and **Sylvie Thiébaux**
Optimisation Research Group, NICTA
Research School of Computer Science, Australian National University
firstname.lastname@anu.edu.au

Abstract

We present a simple and efficient algorithm to solve delete-free planning problems optimally and calculate the h^+ heuristic. The algorithm efficiently computes a minimum-cost hitting set for a complete set of disjunctive action landmarks generated on the fly. Unlike other recent approaches, the landmarks it generates are guaranteed to be set-inclusion minimal. In almost all delete-relaxed IPC domains, this leads to a significant coverage and runtime improvement.

Introduction

Problems without delete lists play an important role in the planning literature. Most planning heuristics, including the admissible heuristics $h^{\max}$, additive $h^{\max}$, LM-cut, and variants (Bonet and Geffner 2001; Haslum, Bonet, and Geffner 2005; Coles et al. 2008; Helmert and Domshlak 2009; Bonet and Helmert 2010) are based on the *delete relaxation* of the planning problem which ignores delete lists. These heuristics attempt to find good lower bounds on the cost h^+ of the optimal delete-relaxed plan, which is NP-equivalent to compute and hard to approximate (Bylander 1994; Betz and Helmert 2009). Hence practical methods for computing h^+ are crucial to assess how close such planning heuristics are able to get to the holy grail they seek.

Delete-free problems of interest in their own right are also starting to appear. An example originating in systems biology is the minimal seed-set problem (Gefen and Brafman 2011), which challenges both classical optimisation methods and optimal planners. Moreover, recent work on translating NP problems into an NP fragment of classical planning opens the possibility that optimal delete-free planning could directly be used to solve a wide range of NP-hard problems (Porco, Machado, and Bonet 2011). Hence practical methods for solving delete-free problems are also important to extend the reach of planning technology.

A possible approach to solving such problems is to treat them as any other planning problem and use any cost-optimal planning algorithm. However, Helmert and Domshlak (2009, Table 1) show that this approach, or even resorting to domain-specific procedures, still leaves many cases where h^+ cannot be computed. Besides two papers in this conference (Gefen and Brafman 2012; Pommerening and Helmert 2012), there is no other published work on optimal planners designed *specifically* for general delete-free planning problems. Bonet and Helmert (2010) established that determining h^+ amounts to computing a minimum-cost hitting set for a complete set of action landmarks for the relaxed problem, and Bonet and Castillo (2011) described an algorithm for computing such a complete set on the fly. However, neither tried to compute h^+, for fear that solving the general hitting set problem optimally would be too hard, settling instead for computing improvements on LM-cut.

Bonet and Castillo (2011) state that "in the future [they] would like to like to develop an effective algorithm for computing exact h^+ values". This is exactly what we do here. We present a simple algorithm which efficiently computes a minimum-cost hitting set of a complete set of landmarks generated on the fly. Unlike in previous work, the landmarks it generates are set-inclusion minimal. We show that this results in simpler hitting set problems and significant improvements in coverage and runtime.

Background

We adopt the standard definition of a propositional STRIPS planning problem, without negation in action preconditions or the goal (see, e.g., Ghallab, Nau, and Traverso 2004, chapter 2). Each action a has a non-negative cost, $\mathrm{cost}(a)$, and the cost of a plan is the sum of the cost of actions in it. The initial state is denoted by s_I and the goal by G.

The Delete Relaxation

The delete relaxation P^+ of a planning problem P is a problem exactly like P except that $\mathrm{del}(a) = \emptyset$ for each a, i.e., no action makes any atom false. The delete relaxation heuristic, h^+ is defined as the minimum cost of any plan for P^+.

Let A be a set of actions in P. We denote by $R^+(A)$ the set of all atoms that are reachable in P^+, starting from the initial state, using only actions in A. We say a set of actions A is a *relaxed plan* iff the goal $G \subseteq R^+(A)$. An actual plan for the delete-relaxed problem is of course a sequence of actions. That $G \subseteq R^+(A)$ means there is at least one sequencing of the actions in A that reaches a goal state. Such a sequencing can be generated in linear time, starting from s_I and selecting actions greedily as they become applicable.

We assume throughout that $G \subseteq R^+(A)$ for some set of actions A (which may be the set of all actions in P^+), i.e., that the goal is relaxed reachable. If it is not, the problem is unsolvable and $h^+ = \infty$.

Relevance Analysis

For optimal delete-free planning, we only need to consider actions that can achieve a relevant atom for the first time. This enables more aggressive, yet still optimality-preserving, pruning of irrelevant actions. The standard relevance analysis method in planning is to backchain from the goal: goal atoms are relevant, an action that achieves a relevant atom is relevant, and atoms in the precondition of a relevant action are relevant. We strengthen this analysis in the delete-free case, by considering as relevant only actions that are possible *first achievers* of a relevant atom. Action a is a possible first achiever of atom p if $p \in \mathrm{add}(a)$ and a is applicable in a state that is relaxed reachable with only actions that do not add p, i.e., if $\mathrm{pre}(a) \subseteq R^+(\{a \mid p \notin \mathrm{add}(a)\})$.

Iterative Minimal Landmark Generation

A *disjunctive action landmark* (landmark, for short) of a problem P is a set of actions such that at least one action in the set must be included in any valid plan for P. Hence, for any collection of landmarks L for P, the set of actions in any valid plan for P is a hitting set for L, i.e., contains an action from each set in L.

For any set of actions A such that $G \not\subseteq R^+(A)$, the complement $\overline{A}$ of A (w.r.t. the whole set of actions) is a disjunctive action landmark for P^+. If A is an inclusion-maximal such "relaxed non-plan", $\overline{A}$ is an inclusion-minimal landmark, since if some proper subset of $\overline{A}$ were also a landmark, A would not be maximal. This leads to the following algorithm for computing h^+:

Initialise the landmark collection L to $\emptyset$. Repeatedly (1) find a minimum-cost hitting set A for L; (2) test if $G \subseteq R^+(A)$; and if not, (3) extend A, by adding actions, to a maximal set A' such that $G \not\subseteq R^+(A')$, and add the complement of A' to the landmark collection.

The algorithm terminates when the hitting set A contains enough actions to reach the goal and is thus a relaxed plan. There is no lower-cost relaxed plan, because any plan for P^+ must contain an action from every landmark in L, and A is a minimum-cost hitting set. Finally, the algorithm eventually terminates, because every iteration adds a new landmark, which cannot already be in L as it is not hit by A, and there is only a finite set of minimal landmarks for P^+.

The algorithm is given in Figure 1. Details about the procedures NEWLANDMARK (which implements step (3) above) and MINCOSTHITTINGSET are given below in the sections on 'Testing Relaxed Reachability' and 'Finding a Minimum-Cost Hitting Set' respectively.

```
1: $L = \emptyset$
2: $A = \emptyset$
3: while $G \not\subseteq R^+(A)$ do
4:     $L = L \cup \mathrm{NEWLANDMARK}(A)$
5:     $A = \mathrm{MINCOSTHITTINGSET}(L)$
6: return $A$
```

Figure 1: Basic Iterative Landmark Algorithm

```
1: $L = \emptyset$
2: while $G \not\subseteq R^+(\bigcup_{l \in L} l)$ do
3:     $L = L \cup \mathrm{NEWLANDMARK}(\bigcup_{l \in L} l)$
4: $A = \mathrm{MINCOSTHITTINGSET}(L)$
5: while $G \not\subseteq R^+(A)$ do
6:     $L = L \cup \mathrm{NEWLANDMARK}(A)$
7:     $H = \mathrm{APXHITTINGSET}(L)$
8:     if $G \subseteq R^+(A \cup H)$ then $A = H$
9:     else $A = A \cup H$.
10:    if $G \subseteq R^+(A)$ then $A = \mathrm{MINCOSTHITTINGSET}(L)$
11: return $A$
```

Figure 2: Iterative Landmark Algorithm with improvements

Improvements to the Algorithm

In fact, any set of actions that hits every landmark in L but is not a relaxed plan can be used as the starting point to generate a new landmark, not in L. This observation is the basis for two improvements to the algorithm: First, it is not necessary to find an optimal hitting set in every iteration. Only when the hitting set A is a relaxed plan do we need to verify that its cost is minimal. Thus, we use a fast approximate algorithm to generate hitting sets H, updating A as we go (lines 8–9 of Figure 2: see also next paragraph below), until one that reaches the relaxed goal is found. Only then do we apply the optimal branch-and-bound algorithm, detailed below, using the cost of the non-optimal A as an initial upper bound. After this, A is a cost-minimal hitting set. If it is also a relaxed plan, it is the solution; if not then the algorithm continues. One candidate for H is the hitting set from the previous iteration extended with the cheapest action in the new (unhit) landmark. Another is the set found by the standard greedy algorithm using the weighted degree heuristic (Chvatal 1979), which greedily adds actions in increasing order of the ratio of their cost to the number of landmarks they hit. We take whichever of these two has lower cost.

Second, we collect the union of recently found hitting sets, and first try using this in place of only the last hitting set. If this larger set is also not a relaxed plan, it is used as the starting point to generate a new landmark. If it is a relaxed plan, earlier hitting sets are forgotten and the collection reset to the last hitting set only. The effect of using a larger relaxed non-plan is similar to that of "saturation" used by Bonet & Castillo (2011): it makes new landmarks have less in common with those already in L, thus creating easier hitting set problems and faster convergence to the optimal h^+ value. Taking this idea a step further, as long as the set of *all* actions appearing in current landmarks is not a relaxed plan, we can use that to generate the next landmark, which will be disjoint from all previous ones. We do this (lines 2 and 3 of Figure 2) to seed L and A before the main loop.

Two steps in this algorithm are frequently repeated, and therefore important to do efficiently: the first is testing if the goal is relaxed reachable with a given set of actions, and the second is finding a hitting set with minimum cost. Their implementations are detailed in the following sections.

Testing Relaxed Reachability

$R^+(A)$, the set of atoms that are relaxed reachable with actions A, can be computed in linear time by a variant of Dijkstra's algorithm: Keep track of the number of unreached preconditions of each action, and keep a queue of newly reached atoms, initialised with atoms true in s_I. Dequeue one atom at a time until the queue is empty; when dequeueing an atom, decrement the precondition counter for the actions that have this atom as a precondition, and if the counter reaches zero, mark atoms added by the action as reached and place any previously unmarked ones on the queue.[1]

When generating a new landmark, we perform a series of reachability computations, with mostly increasing sets of actions, i.e. H, $H \cup \{a_1\}$, $H \cup \{a_1, a_2\}$, etc. Therefore, each reachability test can be done incrementally. Suppose we have computed $R^+(A)$, by the algorithm above, and now wish to compute $R^+(A \cup \{a\})$. If $\text{pre}(a) \not\subseteq R^+(A)$, $R^+(A \cup \{a\})$ equals $R^+(A)$, and the only thing that must be done is to initialise a's counter of unreached preconditions. If not, mark and enqueue any previously unreached atoms in $\text{add}(a)$, and resume the main loop until the queue is again empty. If the goal becomes reachable, we must remove the last added action (a) from the set, and thus must restore the earlier state of reachability. This is done by saving the state of $R^+(A)$ (including precondition counters) before computing $R^+(A \cup \{a\})$, and copying it back if needed.

Finding a Minimum-Cost Hitting Set

Finding a minimum-cost hitting set over the set of landmarks L is an NP-hard problem. We solve it using a recursive branch-and-bound algorithm, with some improvements.

When finding a hitting set for $\{l_1, \ldots, l_m\}$, we already have an optimal hitting set for $\{l_1, \ldots, l_{m-1}\}$. $H^\star(\{l_1, \ldots, l_{m-1}\})$ is clearly a lower bound on $H^\star(\{l_1, \ldots, l_m\})$, and an initial upper bound can be found by taking $H^\star(\{l_1, \ldots, l_{m-1}\}) + \min_{a \in l_m} \text{cost}(a)$. These bounds are often very tight, which limits the amount of search. E.g. if l_m contains any zero-cost action, the initial upper bound is the lower bound, and is thus optimal.

Given a set $L = \{l_1, \ldots, l_m\}$ of landmarks to hit, we pick a landmark $l_i \in L$; the minimum cost of a hitting set for L is $H^\star(L) = \min_{a \in l_i} H^\star(L - \{l \mid a \in l\}) + \text{cost}(a)$. In our implementation, we pick the landmark whose cheapest action has the highest cost, using landmark size to break ties (preferring smaller). We branch on which action in l_i to include in the hitting set, cheapest action first. Next, we describe the lower bounds, and three improvements to the basic branch-and-bound scheme, that we use.

Lower Bounds We use the maximum of two lower bounds on $H^\star(L)$. The first is obtained by selecting a subset $L' \subset L$ s.t. $l \cap l' = \emptyset$ for any $l, l' \in L'$, i.e., a set of pair-wise disjoint landmarks, and summing the costs of their cheapest actions, i.e., $\sum_{l \in L'} \min_{a \in l} \text{cost}(a)$. Finding the set L' that yields the maximum lower bound amounts to solving a weighted independent set problem, but there are reasonably good and fast approximation algorithms (e.g. Halldórsson 2000). The second bound is simply the continuous relaxation of the integer programming formulation of the hitting set problem:

$$\min \sum_a \text{cost}(a) x_a \text{ s.t. } \sum_{a \in l} x_a \geq 1 \ \forall l \in L,\ x_a \in \{0,1\} \forall a$$

Relaxing the integrality constraint results in an LP, which we solve using standard techniques (simplex).[2]

Caching Because the branch-and-bound algorithm is invoked recursively on subsets of L, it may prove increased lower bounds on the cost of hitting these subsets. Improved bounds are cached, and used in place of the lower bound calculation whenever a subset is encountered again. This is implemented by a transposition table, taking the "state" cached as the subset of landmarks that remain to be hit. In the course of computing h^+, we will be solving a series of hitting set problems, over a strictly increasing collection of landmarks: $\emptyset$, $\{l_1\}$, $\{l_1, l_2\}$, etc. Cached lower bounds may be used not only if a subset of L is encountered again within the same search, but also if it is encountered while solving a subsequent hitting set problem.

Decomposition Suppose the collection of landmarks can be partitioned into subsets $L_1, \ldots L_k$, such that no two landmarks in different partitions have any action in common. Then, an optimal hitting set for the entire collection can be constructed by finding, independently, an optimal hitting set for each partition and taking their union. It is rarely the case that the entire landmark collection can be partitioned in this way, but as we select actions to include and remove landmarks that have already been hit, the set that remains becomes sparser. Thus, opportunities for decomposition can arise during the search. We found that most of the time when a split occurs, all partitions but one consist of only one or two landmarks. These tiny subproblems are solved optimally by simple special-purpose routines, and their optimal cost used to update the bounds on remaining subproblems.

Dominance Eliminating dominated elements is a standard technique to speed up hitting set algorithms (e.g. Beasley 1987). Let $L(a)$ be the subset of landmarks in the current collection that include action a. If, for actions a and a', $L(a') \subseteq L(a)$ and $\text{cost}(a) \leq \text{cost}(a')$, we say that *$a$ dominates a'*. Dominated actions are not considered for inclusion in the hitting set, since the dominating action hits at least the same sets, and adds no more to the cost. If two actions hit the same landmarks and have the same cost, both dominate each other, but obviously only one can be excluded.

Dominance is determined by the current landmark collection, so it must be updated when a new landmark is added. However, we do not update dominance as landmarks are removed during search.[3]

Results and Conclusions

We compare our method of computing landmarks with that presented by Bonet & Castillo (2011), when each is used by our Iterative Landmark Algorithm to generate cost-optimal relaxed plans. Bonet & Castillo propose a method based on finding cuts in a graph, similar to how the LM-Cut heuristic is computed. This is not guaranteed to find minimal landmarks, but involves essentially only one relaxed reachability

[1]This is essentially the same algorithm as the implementation of relaxed plan graph construction described by Hoffmann & Nebel (2001, Section 4.3). Liu, Koenig & Furcy (2002) also discuss a similar algorithm, referring to it as "Generalised Dijkstra".

[2]Fisher & Kedia (1990) describe a lower bound which is weaker, but cheaper to compute, using a greedy approximation and a local improvement loop on the dual of the above LP. We found that although the bound it gives is often close to the optimal solution to the LP, solving the LP to optimality still pays off.

[3]We also do not remove landmarks which become subsumed as actions are removed on backtracking. Iterating these inferences to a fixpoint yields a powerful propagator (De Kleer 2011), whose implementation we leave as future work.

computation, whereas our method may use up to $O(|A|)$ of them. In our experiments, all aspects of the implementation apart from landmark generation are identical.

We compare them across problem sets from IPC 1998–2011, counting the number of problems solved within a 30 minute CPU time limit. The IPC 2000 Blocksworld and Logistics problem sets include the larger instances intended for planners with hand-coded control knowledge. For IPC 2008 & 2011 domains, we use the problem sets from the sequential satisficing track.

Table 1 summarises the results: for each domain, it shows the total number of problems (col 1), the number solved by each method (col 2), average number of landmarks generated (over all problems, col 3), average time in seconds (over problems solved by both, col 4) – see Figure 3 (left) for runtimes on individual problems – and average width (cf. below) of hitting set problems (over planning problems solved by both, col 5). As a point of reference, we include the number of problems for which Helmert and Domshlak (2009) report they could compute h^+, using domain-specific procedures or heuristic search with existing admissible heuristics.

Clearly, investing more time into computing minimal landmarks pays off. Our method solves a large superset of the problems solved by Bonet & Castillo's method, and is often significantly faster. There are two main reasons for this: first, our method usually requires fewer landmarks to be generated before a relaxed plan is found; second, and more significant, using minimal landmarks leads to simpler hitting set problems. The optimal hitting set solver needs to be invoked on far fewer problems (a quarter, on average), and as shown in Figure 3 (right), those are solved with fewer node expansions (less than a third, on average). The worst case complexity of the hitting set problem is bounded by the *width* of the set collection (Bonet and Helmert 2010). We also found a strong correlation between width and the practical problem difficulty (70% of problems with above-median width also have an above-median number of node expansions, and vice versa), though there is no simple quantitative relationship. Minimal landmarks yield problems of lower width, and since the optimal hitting set computation dominates runtime in most cases, this leads to a significant speedup. Of course, this comes at a cost: our method spends an average of 31.1% of its time generating landmarks, compared to only 3.3% using Bonet & Castillo's method. In the Schedule domain, all hitting set problems encountered are very easy (accounting for less than 15 seconds in total on the hardest instance), so the difference in runtime is dominated by the overhead for generating minimal landmarks.

Our algorithm solves as many or more problems than heuristic search or domain-specific methods in nearly all domains. Notable exceptions are Satellite (for which the results reported by Helmert and Domshlak were obtained by a domain-specific method) and TPP. Using this larger data set, we can estimate the average error of the LM-Cut heuristic, relative to h^+, on initial states to 3.5% over unit-cost problems and 15.3% over non-unit cost domains. Helmert and Domshlak estimate the average relative error to be only 2.5%, but note that "It is likely that in cases where we cannot determine h^+, the heuristic errors are larger [...]", a hypothesis that we can now confirm.

Domain	#	# solved			avg. #lms		avg. time (s)		avg. width	
		ML	BC	HD	ML	BC	ML	BC	ML	BC
Airport	50	**50**	**50**	37	**246**	247	21.91	**14.90**	**18**	20
Blocksworld 3-ops (small)	35	**35**	35		**36**	56	**0.19**	6.24	**6**	25
Blocksworld 3-ops (large)	66	**53**	18		**305**	595	**102.33**	193.67	**18**	118
Blocksworld 4-ops (small)	35	**35**	**35**	**35**	**17**	**17**	**0.01**	**0.01**	**1**	**1**
Blocksworld 4-ops (large)	66	**66**	**66**		**66**	**66**	0.57	**0.54**	**1**	**1**
Depots	22	**18**	12	10	**164**	412	**0.33**	9.00	**23**	92
Driverlog	20	13	8	**14**	**170**	556	**0.15**	63.35	**29**	93
Freecell	60	**17**	0	6	**275**	531				
Gripper	20	**20**	**20**	**20**	**47**	**47**	**0.01**	**0.01**	**1**	**1**
Logistics'00 (small)	27	**27**	**27**	26	**46**	53	**0.06**	0.21	**9**	13
Logistics'00 (large)	52	**23**	10		**331**	485	**1.23**	158.45	**60**	141
Logistics'98	35	**15**	6	10	**373**	710	**0.02**	0.09	**5**	22
Miconic	150	**150**	99	**150**	**57**	704	**0.08**	160.66	**11**	250
MPrime	35	**28**	17	24	**79**	546	**0.84**	93.69	**17**	167
Mystery	28	**25**	19	18	**68**	441	**0.46**	45.72	**17**	120
Pathways	30	**28**	8	5	**196**	315	**0.06**	207.04	**18**	86
Pipesworld NoTankage	50	**20**	9	18	**243**	997	**0.25**	7.94	**29**	120
Pipesworld Tankage	50	**15**	6	11	**219**	756	**0.07**	19.30	**19**	121
PSR (small)	50	**50**	**50**	**50**	**3**	**3**	**0.05**	**0.05**	**1**	**1**
Rovers'06	40	18	**19**	14	**365**	581	**26.31**	88.35	**54**	101
Satellite'04	36	8	5	**9**	**261**	573	**0.18**	3.72	**26**	80
Schedule	500	**500**	**500**		101	**92**	30.91	**8.25**	**7**	9
Storage	30	**23**	20		**115**	214	**1.21**	41.71	**25**	64
TPP	30	15	15	**18**	**255**	654	**5.50**	28.91	**32**	98
Trucks (ADL)	30	**30**	**30**	10	**45**	77	**0.38**	3.01	**7**	40
ZenoTravel	20	**13**	10	**13**	**135**	334	**0.91**	91.81	**29**	70
Cybersec	30	**30**	27		**138**	298	**21.16**	157.68	**57**	122
Elevators	30	**27**	11		**437**	737	**21.43**	176.87	**180**	412
Openstacks (ADL)	30	**30**	**30**		**124**	162	**1.31**	7.87	**1**	79
ParcPrinter	30	**30**	**30**		**79**	100	**0.20**	0.36	**9**	13
PegSol	30	**30**	**30**		**40**	61	**0.05**	0.16	**24**	39
Scanalyzer	30	**15**	4		**118**	446	**0.03**	240.48	**8**	158
Sokoban	30	**30**	**30**		**59**	147	**0.20**	7.58	**21**	78
Transport	30	**6**	**6**		**522**	911	**0.45**	1.83	**43**	85
Woodworking	30	**19**	9		**113**	302	**0.18**	26.02	**12**	50
Barman	20	**18**	5		**264**	1372	**7.46**	147.71	**133**	480
FloorTile	20	**12**	9		**248**	559	**10.08**	248.97	**104**	220
NoMystery	20	**5**	4		324	**175**	**1.58**	13.40	**44**	165
Visitall	20	**2**	0		**1624**	1636				

Table 1: Comparison between our Minimal Landmark (ML) generation method, Bonet & Castillo's method, and results reported by Helmert and Domshlak (HD). Domains with non-unit action costs are grouped below the line.

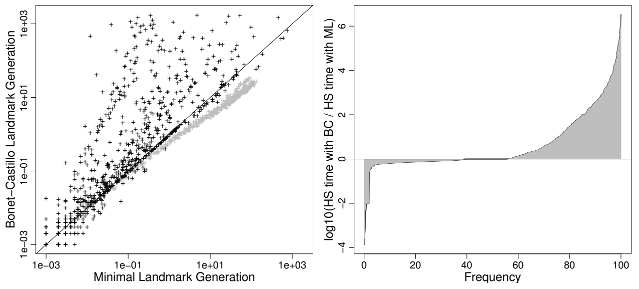

Figure 3: Left: Time to compute h^+ using Minimal Landmark Generation and Bonet-Castillo methods, on problems solved by both. The schedule domain (500 instances) is in gray. Right: Distribution of the relative time spent on hitting set computation with the two methods.

We also tried the algorithm on the minimal seed-set problem. It solves all instances, in marginally less time (2/3 on average) than that reported by Gefen and Brafman (2011) for their domain-specific algorithm.

Acknowledgements P. Haslum and S. Thiébaux are supported by ARC Discovery Project DP0985532 "Exploiting Structure in AI Planning". The authors would also like to acknowledge the support of NICTA which is funded by the Australian Government as represented by DBCDE and ARC through the ICT Centre of Excellence program.

References

Beasley, J. 1987. An algorithm for the set covering problem. *European Journal of Operational Research* 31:85–93.

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In *Proc. of the ICAPS'09 Workshop on Heuristics for Domain-Independent Planning*.

Bonet, B., and Castillo, J. 2011. A complete algorithm for generating landmarks. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 315–318.

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

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. 19th European Conference on Artificial Intelligence (ECAI'10)*, 329–334.

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

Chvatal, V. 1979. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research* 4(3):233–235.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Additive-disjunctive heuristics for optimal planning. In *Proc. 18th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 44–51.

De Kleer, J. 2011. Hitting set algorithms for model-based diagnosis. In *Proc. 22nd International Workshop on Principles of Diagnosis (DX'11)*, 100–105.

Fisher, M., and Kedia, P. 1990. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science* 36(6):674–688.

Gefen, A., and Brafman, R. 2011. The minimal seed set problem. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 319–322.

Gefen, A., and Brafman, R. 2012. Pruning methods for optimal delete-free planning. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers. ISBN: 1-55860-856-7.

Halldórsson, M. 2000. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications* 4(1):1–16.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. 20th National Conference on AI (AAAI'05)*, 1163–1168.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169.

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

Liu, Y.; Koenig, S.; and Furcy, D. 2002. Speeding up the calculation of heuristics for heuristic search-based planning. In *Proc. 18th National Conference on Artificial Intelligence (AAAI'02)*, 484–491.

Pommerening, F., and Helmert, M. 2012. Optimal planning for delete-free tasks with incremental lm-cut. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Porco, A.; Machado, A.; and Bonet, B. 2011. Automatic polytime reductions of NP problems into a fragment of STRIPS. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 178–185.