A Complete Algorithm for Generating Landmarks

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Julio Castillo
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
juliocc@gmail.com

Abstract

A collection of landmarks is complete if the cost of a minimum-cost hitting set equals h^+ and there is a minimum-cost hitting set that is an optimal relaxed plan. We present an algorithm for generating a complete collection of landmarks and we show that this algorithm can be extended into effective polytime heuristics for optimal and satisficing planning. The new admissible heuristics are compared with current state-of-the-art heuristics for optimal planning on benchmark problems from the IPC.

Introduction

A landmark (for a given state) is a subset of actions such that each valid plan contains at least one action in the landmark. Current state-of-the-art admissible heuristics for optimal planning are landmark heuristics. These heuristics calculate a collection of landmarks for the given state and combine the costs of the landmarks into an admissible estimate. Landmark heuristics differ in how landmarks are computed and how costs are combined. The current most successful approach generates landmarks by considering cut-sets in justification graphs and combine the costs using hitting sets (Helmert and Domshlak 2009; Bonet and Helmert 2010).

Landmark heuristics are instances of delete-relaxation heuristics that try to approximate as close as possible the *optimal delete relaxation heuristic* h^+ (Hoffmann 2005). The value of h^+ is the cost of an optimal plan for the task that results of removing the delete lists from the operators. Its value is a lower bound on the true cost h^* yet it is NP-hard to compute (Bylander 1994) or approximate within a constant factor (Betz and Helmert 2009). Nonetheless, h^+ had shown to be a powerful heuristic and thus it is important to design effective methods to compute it.

In this paper, we present an algorithm for computing a complete collection of landmarks from which the exact value of h^+ can be computed. Of course, the algorithm does not run in polynomial time, but it provides new insight for the *principled generation* of landmarks. Indeed, using this insight, we propose novel polytime algorithms for improving a given collection of landmarks and define novel polytime heuristics. These heuristics are compared with the current state of the art on benchmark problems.

The paper is organized as follows. We first review fundamental concepts about landmarks, hitting sets and complete collections. Then, we present a complete algorithm for generating landmarks that is based on a conflict-directed mechanism, and show how to modify the algorithm to run in polynomial time and to improve on some of its limitations. The paper finishes with preliminary experiments and a discussion.

Landmarks

A STRIPS problem with action costs is a tuple $P = \langle F, O, I, G, c\rangle$ where F is the set of fluents, O is the set of actions or operators, I and G are the initial state and goal description, and $c : O \to \mathbb{N}$ is the cost function. We are interested in delete relaxations, so we assume that the operators have empty delete lists, and thus "plan" and "relaxed plan" denote the same. We also assume that all fluents have finite $h^{\max}$ values,[1] which implies that the problem has finite h^+ value. Finally and without loss of generality, we require that all operators have nonempty preconditions, that there are two fluents $s, t \in F$ such that $I = \{s\}$ and $G = \{t\}$, and that there is a unique operator *fin* that adds t; when these simplifying assumptions are not met, they can be achieved through simple linear-time transformations. The precondition and effects of $a \in O$ are denoted by $\text{pre}(a)$ and $\text{post}(a)$, and the h^+ value for state I is denoted by $h^+(P)$.

An (action) *landmark* for P is a set $\{a_1, \ldots, a_n\}$ of actions such that every plan for P must contain at least one of such actions. Notice that this definition is done for delete-free problems. However, if P is not delete free, every plan for P is also a plan for its delete-free relaxation P^+. Thus, every landmark for P^+ is also a landmark for P.

As recently shown (Helmert and Domshlak 2009; Bonet and Helmert 2010), landmarks can be extracted from *justification graphs*. A justification graph is a labeled directed graph on fluents whose edges are defined by a fixed *precondition-choice function* (pcf) D, that maps actions a into preconditions $D(a) \in \text{pre}(a)$, so that there is an edge (p, q) labeled with a iff $D(a) = p$ and $q \in \text{post}(a)$. The justification graph associated to the pcf D is denoted by $G(D)$.

An s-t-cut of $G(D)$ is a partition $(X, X^c = F \setminus X)$ of

[1] The $h^{\max}$ values are obtained by solving the $h^{\max}$ equation in linear time (Bonet and Geffner 2001).

vertices such that $s \in X$ and $t \in X^c$, and its cut-set is the set of edges that cross the cut. Since a plan for P defines an s-t-path on $G(D)$, the labels of the edges in a cut-set form a landmark for P. Furthermore, the collection $\mathcal{F}_L$ of landmarks that can be obtained from cut-sets of justification graphs is complete, meaning that $\mathcal{F}_L$ contains enough information for computing $h^+(P)$ and an optimal (relaxed) plan for P (Bonet and Helmert 2010).

Unfortunately, the number of pcfs is exponential as well as the number of s-t-cuts (in general) for a given $G(D)$. However, not every landmark in $\mathcal{F}_L$ is required and the question is how to compute a complete subset of $\mathcal{F}_L$ in a principled and informed manner. Before answering this question, let us to recall how landmarks are used to compute heuristics and to formally define complete collections.

Hitting Sets and Complete Collections

A hitting set problem is a pair $\langle \mathcal{A}, c\rangle$ where $\mathcal{A} = \{A_1, \ldots, A_n\}$ is a family of subsets and $c : A \to \mathbb{N}$ is a cost function, $A = \cup_{i=1}^{n} A_i$. A hitting set H for $\langle \mathcal{A}, c\rangle$ is a subset of A that "hits" every A_i; i.e., $H \cap A_i \neq \emptyset$. The cost of H is $c(H) = \sum_{a \in H} c(a)$ and it is of minimum cost if its cost is no greater than the cost of any other hitting set.

A collection $\mathcal{L} = \{L_1, \ldots, L_n\}$ of landmarks can be thought as an instance $\langle \mathcal{L}, c\rangle$ of a hitting set problem. Indeed, if $\min\langle \mathcal{L}, c\rangle$ denotes the cost of a minimum-cost hitting set, then $\min\langle \mathcal{L}, c\rangle \leq h^+(P)$. A collection $\mathcal{L}$ is said to be *complete* for P iff $\min\langle \mathcal{L}, c\rangle = h^+(P)$ and there is a minimum-cost hitting set H for $\langle \mathcal{L}, c\rangle$ that is a plan for P.

Computing a minimum-cost hitting set for a given $\langle \mathcal{A}, c\rangle$ is NP-hard, but it is fixed-parameter tractable with respect to its width. The width of $\langle \mathcal{A}, c\rangle$ is defined as the size of the largest connected component of its interaction graph. The interaction graph for $\langle \mathcal{A}, c\rangle$ is an undirected graph whose vertices are the elements in A and there is an edge (a, b) iff there is a subset $A_i \in \mathcal{A}$ that contains $\{a, b\}$. By using dynamic programming, one can show that a minimum-cost hitting set can be computed in $O(\|\langle \mathcal{A}, c\rangle\| + w4^w)$ time for problems of width w (Bonet and Helmert 2010), where $\|\langle \mathcal{A}, c\rangle\|$ is the length of the encoding of $\langle \mathcal{A}, c\rangle$. This bound is a worst-case theoretical bound. In our experiments, a minimum-cost hitting set is found with search and this bound is not attained.

A Complete Algorithm

Interestingly, the basis for designing a complete algorithm is implicit in Bonet and Helmert's proof for the completeness of the collection $\mathcal{F}_L$, because the proof not only shows completeness, but also suggests an efficient method for improving an incomplete collection. Indeed, the following result builds on such proof.

Theorem 1. *Let $P = \langle F, O, I, G, c\rangle$ be a relaxed planning task. There is a linear-time Algorithm A that on input $H \subseteq O$ outputs YES, if H contains a plan for P, or a landmark L, not hit by H, if H does not contain a plan for P.*

Proof. Let R be the subset of fluents reachable from I by only using operators in H. If R contains t, output YES. Otherwise, construct the pcf D as follows:

- if $\text{pre}(a) \subseteq R$, set $D(a)$ to some $p \in \text{pre}(a)$,
- if $\text{pre}(a) \not\subseteq R$, set $D(a)$ to some $p \in \text{pre}(a) \setminus R$.

Consider now the s-t-cut (R, R^c) of $G(D)$ where $R^c = F \setminus R$. It is an s-t-cut because $s \in R$ and $t \in R^c$. We claim that H does not hit the cut-set; i.e., that there is no operator $a \in H$ that labels an edge in $G(D)$ going from R into R^c.

Indeed, let a be an arbitrary element of H. If $\text{pre}(a) \subseteq R$, then $\text{post}(a) \subseteq R$ and no edge labeled by a cross the cut because such edges go from R into R. If $\text{pre}(a) \not\subseteq R$, then $D(a) \notin R$ and no edge labeled by a cross the cut because such edges originate in R^c. Therefore, H does not hit the cut-set. Finish with the fact that the labels for the edges in the cut-set make up a landmark L for P.

The algorithm performs a reachability analysis for computing R, then constructs the pcf D, and finally computes the landmark L. With the proper data structures, all these tasks can be performed in linear time. □

Observe that Algorithm A is not fully specified as there may be more than one choice when constructing the pcf, and thus different implementations are possible. In any case, Algorithm A can be used to generate a complete collection of landmarks as follows. Let $\mathcal{L}$ be an initial (possibly empty) collection of landmarks, and H a minimum-cost hitting set for $\langle \mathcal{L}, c\rangle$. Then, by calling $A(H)$ one can decide whether H contains an optimal relaxed plan for P, or else obtain a new landmark L that is not hit by H. In the former case, we know that $\mathcal{L}$ is a complete collection while in the latter case, $\mathcal{L}$ can be extended into $\mathcal{L}' := \mathcal{L} \cup \{L\}$. Then, a minimum-cost hitting set H' for $\mathcal{L}'$ can be computed, and the whole process repeated. This algorithm computes a complete collection and it is called Algorithm $C1$.

In the worst case, $C1$ does not run in polynomial time because 1) computing a minimum-cost hitting set for $\mathcal{L}$ is NP-hard, but also because 2) it may incur in an exponential number of iterations. In spite of this, Algorithm $C1$ is interesting because it constructs a complete collection using a *conflict-directed mechanism*: for a candidate collection $\mathcal{L}$, $C1$ computes an optimal plan H (i.e. hitting set) with respect to $\mathcal{L}$ and applies it. If H succeeds, $\mathcal{L}$ is complete. Otherwise, the conflict is analyzed by Algorithm A and a new landmark not achieved by H is obtained. $C1$ offers an starting point for the principled generation of landmarks instead of a method based on the blind generation of pcfs and cut-sets.

Indeed, by simply imposing a bound on the width of $\mathcal{L}$ and a bound on the number of iterations, Algorithm $C1$ becomes Algorithm $C2$ that is tractable but incomplete, and it is shown in Fig. 1. Algorithm $C2$ has two parameters w and N that bound the width of the collection and the number of iterations respectively. The notation $\mathcal{L} \oplus L$ refers to the extension of $\mathcal{L}$ with landmark L. The extension is not a simple union because it involves the removal of dominated landmarks from $\mathcal{L}$; i.e., landmarks $L' \in \mathcal{L}$ with $L \subset L'$. As a result, the extended collection $\mathcal{L} \oplus L$ may be of smaller size and even of smaller width.[2]

[2]The decrease in width or size is the reason why the algorithm may incur in an exponential number of iterations.

```
1: L := {initial collection of landmarks}
2: i := 0
3: while width(L) ≤ w and i < N do
4:     H := {minimum-cost hitting set for ⟨L, c⟩}
5:     L := A(H)
6:     if L = YES then terminate
7:     L := L ⊕ L
8:     i := i + 1
9: end while
```

Figure 1: Algorithm $C2(w, N)$ for computing a collection of landmarks. The parameters w and N are bounds on the width and the number of iterations respectively.

Clearly, if $C2(w, N)$ terminates at line 6, then $c(H) = h^+(P)$. Otherwise, $c(H) \leq h^+(P)$. Therefore, the Algorithm $C2(w, N)$ can be used to compute admissible heuristics for optimal planning in polytime for fixed w and N. Unfortunately, we had seen in our experiments that $width(\mathcal{L})$ grows quite fast reaching the bound w in few iterations. This is because Algorithm A tends to generate landmarks in the same connected component instead of generating landmarks in a more balanced manner. Thus, although Algorithm $C2$ is correct, it is often ineffective when compared to other heuristics.

In the following, we present a technique for dealing with this problem that we call *saturation* and which permits the *continual* generation of landmarks past the time of achieving the width bound. As we will see, this technique can be used to obtain better heuristics without giving up the polynomial time guarantees for both optimal and satisficing planning.

Saturation

Let w be the width bound, $\mathcal{L}$ a collection of landmarks and H a minimum-cost hitting set for $\mathcal{L}$. Further, let L be a landmark not hit by H, as the one generated by $A(H)$. If the width of $\mathcal{L} \oplus L$ is greater than w, then $\mathcal{L}$ cannot be extended. However, we can generate a new landmark L' by calling A with $H \cup H_S$ where H_S is a (perhaps suboptimal) hitting set for $\{L\}$. If A detects that $H \cup H_S$ contains a plan, then we know that $c(H) \leq h^+(P)$ and $c(H)$ is an admissible estimate. Otherwise, A computes a new landmark L' that is not hit by $H \cup H_S$. This landmark is used to extend $\mathcal{L}$ if the width of $\mathcal{L} \oplus L'$ is less than or equal to w, or to extend $\{L\}$ in the other case. This process can be iterated a number of times to obtain an admissible estimate $c(H)$ for optimal planning or a non-admissible estimate $c(H) + c(H_S)$ for *satisficing planning*.

We call the set H_S the *saturation* of H with respect to $\{L\}$. In general, we keep a collection $\mathcal{S}$ of landmarks that need to be "saturated" with respect to a given minimum-cost hitting set H. The algorithm that performs saturation is called Algorithm $C3$ and shown in Fig. 2.

As before, Algorithm $C3$ is not fully specified as there may be more than one choice when computing the hitting set H_S. However, there are two methods that should be mentioned. The first method names the whole approach and it just consists of selecting actions from each landmark in S that is not hit by H. This method can be done incrementally by inserting one (or more) actions from L each time that $\mathcal{S}$ is extended with L. The second method computes a suboptimal hitting set using the linear-time "pricing" method of Chvatal (1979). For lack of space, we just say that the pricing method computes a *greedy* hitting set H_S for $\mathcal{S}$ by including at each iteration an action a that maximizes the number of remaining landmarks hit by a per unit cost of a.

```
1: L := {initial collection of landmarks}
2: S := ∅
3: i := 0
4: while i < N do
5:     H := {minimum-cost hitting set for ⟨L, c⟩}
6:     H_S := {hitting set for ⟨S, c⟩ given H}
7:     L := A(H ∪ H_S)
8:     if L = YES then terminate
9:     if width(L ⊕ L) ≤ w then L := L ⊕ L
10:    else S := S ∪ {L}
11:    i := i + 1
12: end while
```

Figure 2: Algorithm $C3(w, N)$ for computing a collection of landmarks using saturation. The parameters w and N are bounds on the width and the number of iterations.

Preliminary Experiments

We performed experiments on a total of 881 instances for 27 domains from previous International Planning Competitions. We are interested in the quality of the heuristics as measured by the number of expanded nodes until the last f-layer, an objective measure of quality that is not affected by the tie-breaking criteria used for the open list. The evaluation compares the heuristics LM-cut (Helmert and Domshlak 2009), $\max h_p^{\text{LM-cut}}$ and $h_{p,w}^{\text{LM-cut}}$ for $p = 3$ and $w = 5$ (Bonet and Helmert 2010), and 3 novel "saturation" heuristics that were instantiated into 6 different heuristics.

The saturation heuristics differ in how the initial collection of landmark is obtained and how the saturation is computed. For the former, we use an idea similar to $h_{p,w}^{\text{LM-cut}}$ of performing $p \in \{1, 3\}$ passes of the LM-cut method to generate the initial collection of landmarks. For the latter, we tried the pricing method (denoted by $h_{p,w}^{\text{price}}$), and an incremental saturation in which whenever a landmark L needs to be saturated, only one action (denoted by $h_{p,w}^{\text{one}}$) or all actions (denoted by $h_{p,w}^{\text{all}}$) in L are inserted into H_S. In all cases, we use a width bound $w = 5$ and maximum number of iterations $N = 25$.

We do not have enough space to present the results in detail. Instead, we make the following remarks:

- As expected, no heuristic expanded more nodes than the LM-cut heuristic $h^{\text{LM-cut}}$.
- No heuristic improved on $h^{\text{LM-cut}}$ for airport, blocks, openstacks (2006 instances), parcprinter and logistics (2000 instances), suggesting that $h^{\text{LM-cut}}$ computes h^+ on these domains.
- The domains can be divided into three groups according to whether the percentage of the reduction on the number

				$h^{\text{price}}_{p,5}$		$h^{\text{one}}_{p,5}$		$h^{\text{all}}_{p,5}$	
domain	$h^{\text{LM-cut}}$	$\max h^{\text{LM-cut}}_3$	$h^{\text{LM-cut}}_{3,5}$	$p=1$	$p=3$	$p=1$	$p=3$	$p=1$	$p=3$
trucks	23,326	0.1%	0.2%	8.6%	11.5%	5.9%	**17.4%**	2.2%	4.9%
zenotravel	8,406	24.8%	**25.4%**	0.0%	25.3%	0.0%	25.3%	0.0%	25.3%
tpp	23,417	28.1%	28.4%	18.9%	28.6%	15.7%	**28.8%**	15.5%	28.7%
scanalyzer	8,097	32.4%	32.7%	0.2%	**33.1%**	0.3%	**33.1%**	0.2%	33.0%
pipesworld-tankage	6,442	36.1%	**38.5%**	0.0%	38.2%	0.0%	**38.5%**	0.0%	38.3%
elevators	166,459	36.7%	37.8%	5.3%	37.7%	5.5%	**38.6%**	4.9%	37.8%
satellite	4,117	36.0%	36.6%	0.8%	**38.7%**	0.3%	37.4%	1.2%	37.5%
mprime	337	42.1%	**42.4%**	2.3%	**42.4%**	2.3%	**42.4%**	2.3%	**42.4%**
depot	18,394	36.5%	42.0%	0.0%	42.9%	0.6%	**43.0%**	0.0%	42.6%
pegsol	985,520	27.7%	27.9%	44.3%	**50.7%**	39.0%	48.2%	30.8%	44.5%
woodworking	43,675	46.7%	46.8%	25.5%	54.5%	25.5%	**57.0%**	25.5%	**57.0%**
openstacks-2006	481,665	57.8%	61.8%	4.0%	**64.8%**	4.0%	**64.8%**	4.0%	64.4%
freecell	68,265	73.3%	74.3%	0.1%	74.5%	17.8%	**76.6%**	1.1%	74.5%

Table 1: Total number of expanded nodes for *all instances* in a given domain for $h^{\text{LM-cut}}$, and percentages of the reduction on the number of expanded nodes with respect to $h^{\text{LM-cut}}$ for other heuristics. Top domains are those considered of "moderate" reduction (10%-50%) while bottom domains are considered of significant reduction (50%-100%). Best reductions are shown in boldface.

of expanded nodes with respect to $h^{\text{LM-cut}}$ is minor (0%-10%), moderate (10%-50%) or significant (50%-100%). Table 1 shows the domains of moderate and significant reduction ordered by this percentage.

- $h^{\text{LM-cut}}_{3,5}$ is the best heuristic in terms of expanded nodes only for zenotravel, and is tied with other heuristics in pipesworld-tankage and mprime. In the other domains, the saturation heuristics are better than $h^{\text{LM-cut}}_{3,5}$.
- For $p=1$, $h^{\text{LM-cut}}_{1,5}$ equals $h^{\text{LM-cut}}$. However, for $p=1$, the saturation heuristics are better than $h^{\text{LM-cut}}$ and, in some cases, better than $h^{\text{LM-cut}}_{3,5}$ (see Table 1).
- The heuristic $h^{\text{one}}_{3,5}$ obtained the most reductions across all domains (see Table 1).
- The dominant factor in time is the number of passes of the LM-cut loop used to initialize $\mathcal{L}$. In problems with minor or moderate reduction, the new heuristics are slower than $h^{\text{LM-cut}}_{3,5}$ (which is slower than $h^{\text{LM-cut}}$). In problems with significant reduction, the running times improve substantially approaching that of $h^{\text{LM-cut}}$.

These results show that is possible to generate landmarks in a focused manner to improve landmark heuristics.

Discussion

We defined the simple and efficient Algorithm A that decides when a set H of actions contains a relaxed plan for a given task, or computes a landmark not hit by H when it contains no such plan. Algorithm A is used to design Algorithm $C1$ that computes a complete collection of landmark using a conflict-directed approach, and its modification Algorithm $C3$ that runs in polytime and uses saturation. Saturation is a general idea that can be implemented in several ways. We proposed three saturation methods and evaluated them over benchmarks problems from the IPCs. The results show that the new heuristics improve on the current state of the art for optimal planning.

In the future, we would like to develop an effective algorithm for computing exact h^+ values, and to define better admissible heuristics and also non-admissible landmark-base heuristics for satisficing planning.

Acknowledgments Thanks to M. Helmert, P. Haslum and J. Hoffmann for interesting discussions. Part of this work was done during a visit of B. Bonet to Freiburg University.

References

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. *Proc. 32nd Annual German Conf. on Artificial Intelligence*, 9–16.

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. *Proc. 19th European Conf. on Artificial Intelligence*, 329–334.

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

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? *Proc. 19th Int. Conf. on Automated Planning and Scheduling*, 162–169.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.