Landmark Extraction via Planning Graph Propagation

Lin Zhu and **Robert Givan***
Electrical and Computer Engineering, Purdue Univeristy, West Lafayette, IN 47907-1285
{lzhu, givan}@purdue.edu

Abstract

The planner GRAPHPLAN is based on an efficient propagation of reachability information which then effectively guides a search for a valid plan. We propose a framework in which a broader class of information, including the original reachability information, can be propagated in the plan graph in polynomial time. As an example, we exhibit an algorithm for propagating "landmark" information, where a landmark is a proposition or action that must occur in any correct plan. This algorithm computes in polynomial-time a set of planning landmarks significantly larger than previously published landmark computation algorithms. We show how to leverage this algorithm to extract an incrementally computable planning heuristic. We present results showing that enforced hill-climbing using this heuristic is significantly more robust than FFPLAN in three very different planning domains (*Sokoban*, *Blocks-world*, and *Logistics*)—somewhat slower than FFPLAN when FFPLAN performs well, but much faster when FFPLAN performs badly. We include a discussion of our ongoing work improving the resulting heuristic search as well as other directions for utilizing the landmarks extracted.

Planning

We generally follow (Koehler & Hoffmann 2000) and (Blum & Furst 1997) for notation regarding STRIPS planning graphs, varying somewhat for our purposes.

STRIPS Planning. Let X be a finite set of propositions. A *state* S is a finite subset of X. An *action* o is a triple $o = (\text{PRE}(o), \text{ADD}(o), \text{DEL}(o))$ where $\text{PRE}(o)$ are the *preconditions*, $\text{ADD}(o)$ is the *add list* and $\text{DEL}(o)$ is the *delete list*, each being a set of propositions. The result $\text{RESULT}(S, (o_1, \ldots, o_n))$ of applying an action sequence $(o_1, \ldots, o_n)$ to a state S is given by $\text{RESULT}(\text{RESULT}(S, (o_1, \ldots, o_{n-1})), (o_n))$, where for n equals 1 the result is undefined unless $\text{PRE}(o_1) \subseteq S$, and $(S \cup \text{ADD}(o_1)) - \text{DEL}(o_1)$ otherwise.

A *planning task* P is a set of actions containing actions START and FINISH, where the action START has no preconditions and the action FINISH has no add or delete effects. We say that the preconditions of the FINISH action are the *goal*. The corresponding *relaxed planning task* P' (which ignores delete effects) is $P' = \{(\text{PRE}(o), \text{ADD}(o), \emptyset) \mid (\text{PRE}(o), \text{ADD}(o), \text{DEL}(o)) \in P\}$. A *sequential plan*, or *plan* for short, for a task P is an ordered action sequence $\vec{o}$ from O^*, beginning with START such that $\text{RESULT}(\emptyset, \vec{o})$ is defined. We call the plan *successful* if it also ends with FINISH. We say that a plan *achieves* the last action of the plan and any proposition in the resulting state. We say that propositions and actions achieved by plan prefixes *occur during* the plan at the "time" given by the length of the prefix.

A *parallel plan* is an ordered sequence $\vec{\alpha}$ of action sets $\alpha_i \subseteq O$ of *independent* parallel actions. The actions in a set α are *independent* if, for any state S and any ordering $\vec{o}$ of the actions in α, the state $\text{RESULT}(S, \vec{o})$ is the same. We say two actions *interfere* if one deletes a precondition or an add-effect of the other, i.e., if $(\text{DEL}(o_1) \cap \text{ADD}(o_2)) \cup (\text{DEL}(o_2) \cap \text{ADD}(o_2))$ is non-empty.

Plangraph Propagation. We assume that the action set O contains a "no-op action" $(\{p\}, \{p\}, \emptyset)$ for each proposition p in X. The *planning graph* is a directed graph (V, E) where the nodes V can be partitioned into disjoint *levels* O_1, X_1, O_2, X_2, ..., O_k for some k, where each X_i is a separate copy of the proposition set X and each O_i is a copy of the action set O. Every edge in E connects an action o in some level O_i with either a precondition in $\text{PRE}(o)$ in the preceding level X_i or an effect in $\text{ADD}(o)$ in the following level X_{i+1}, and every such precondition or effect edge is present[1].

We consider *propagation methods* that attach labels to every node in the planning graph, representing information of interest. Node labels are computed from the labels on adjacent nodes at the previous level and any information needed from the node being labeled (in forward propagation). To specify a propagation method, we specify the syntax and semantics of the labels, the method for initializing the labels on the first level, and the labeling method for computing a node label from its adjacent predecessors.

For example, the relaxed reachability analysis for FFPLAN in (Hoffmann & Nebel 2001) is a propagation

*We are grateful to Alan Fern and Matthew Greig for useful discussions and editorial suggestions.

[1]We assume that the number of graph levels, k, is chosen large enough that the edges in the final levels would repeat if the graph were further extended. In practice, graphs with fewer levels may be used, effectively.

method on the planning graph for the relaxed planning problem. In that case, the node labels are Boolean values semantically representing (necessary but not sufficient) claims of action applicability at action levels and proposition achievability at proposition levels. Only START is labeled true on the first level, and every action (proposition) node is then labeled with the conjunction (disjunction) of the labels at the previous level. As is typical for propagation methods, nodes labeled false can be omitted from the planning graph.

Quadratic planning graph. The *quadratic planning graph* is a directed graph (V_q, E_q), where V_q contains the nodes $V = X_1 \cup O_1 \cup \ldots \cup X_k$ from the standard planning graph, but also contains all pairs of nodes appearing the same level in the standard planning graph, i.e.

$$V_q = V \cup (O_k \times O_k) \cup \bigcup_{1}^{k-1} (O_i \times O_i) \cup (X_i \times X_i).$$

The edges E_q contain those in E as well as edges into (o_1, o_2) from each node in $\text{PRE}(o_1) \cup \text{PRE}(o_2) \cup (\text{PRE}(o_1) \times \text{PRE}(o_2))$ and edges out of (o_1, o_2) to each node in $\text{ADD}(o_1) \cup \text{ADD}(o_2) \cup (\text{ADD}(o_1) \times \text{ADD}(o_2))$.

The first phase of GRAPHPLAN can be viewed as a propagation method on the quadratic planning graph. In this case, boolean values are propagated, representing (necessary but not sufficient) claims that the labeled proposition/action/pair is reachable at the given level (where a pair of actions or propositions is reachable iff the pair elements can be simultaneously reached). In the initial level, only the START action is labeled true. At subsequent action levels, pairs of interfering actions are labeled false and other action and action-pair nodes are labeled with the conjunction of adjacent-previous-node labels. Proposition and proposition-pair nodes are labeled with the disjunction of adjacent-previous-node labels. Again, any node labeled false can typically be omitted from the graph.

We note that other kinds of propagation can easily be designed. Each propagation will compute labels based on the labels of adjacent previous nodes along with some information local to the node to be labeled. That local information could include labels left by previous propagations or the number of the level being labeled, for example.

Landmark Extraction via Planning Graph Propagation

Planning graph propagations can be used to compute various kinds of planning *landmarks.* A landmark for a planning problem is a proposition or an action must be occur during any successful plan[2]. We are interested in action landmarks in addition to proposition landmarks because an action landmark can represent an important conjunction of propositions (the preconditions of the action).

[2]When the planning graph propagation algorithms are used with graphs that are not fully extended to quiescence, the "landmarks" found are only guaranteed to be present in successful plans of the shorter parallel length. This can be an asset, as the algorithm may find "landmarks" that occur on all short plans, directing the planner towards a shorter plan.

Proposition landmarks were introduced in (Porteous, Sebastia, & Hoffmann 2001), where the problem of finding landmarks for a planning task was shown to be PSPACE-hard and incomplete methods for finding landmarks were given. The resulting landmarks were shown to be useful in accelerating the FFPLAN and IPP planners.

Once a candidate proposition landmark is identified, a simple planning graph propagation can (soundly but not completely) verify that the candidate is, in fact, a landmark by simply checking that the FINISH action can never be taken in the relaxed problem (delete effects removed), but with all actions adding the candidate landmark removed. We call this simple propagation *landmark verification*. The method shown in (Porteous, Sebastia, & Hoffmann 2001) finds landmarks by heuristically identifying candidates and then applying this propagation to verify which candidates are landmarks.

We next give a planning-graph propagation that computes the set of *causal* landmarks that will survive the landmark verification process. We define "causal" below, and claim that all desired landmarks are causal. Thus, this algorithm eliminates the need for heuristic candidate generation, and in fact finds many more landmarks in our example domains than those found by the heuristic-candidate-generation approach. The algorithm finds landmarks in a single propagation that is analogous to, but somewhat more expensive than, a single landmark verification propagation (that is, substantially less expensive than a large number of landmark verifications).

We call a propositional landmark *causal* when every successful plan contains an action that requires the landmark as a pre-condition. We call every action landmark causal. Non-causal landmarks are in some sense "accidental" effects of actions required for other purposes. Non-causal landmarks can be misleading to the system[3], and we consider it a feature of our propagation that it rejects such landmarks (whereas landmark verification alone accepts non-causal landmarks).

In our propagation method for computing landmarks, the propagated labels are actions or propositions. Action or proposition N is labeled with label L at level i in the planning graph[4] to represent the claim that any i-step parallel plan achieving N must contain an occurrence of L. In the initial graph level, an action level, every action is labeled with itself and no other labels. Each proposition node in the graph is labeled with the intersection of the labels on its predecessor action nodes (because a landmark label applies

[3]Consider, for instance, a blocks-world domain where there are both HANDEMPTY and HANDFULL propositions for some reason. In such a domain, HANDFULL will typically be a non-causal landmark for most goals, but also typically misleading as a subgoal. In general, when the problem representation contains propositions at different levels of abstraction, more general propositions will be (possibly misleading) landmarks whenever specialized propositions under them are landmarks. Here, HANDFULL is an abstract landmark that does not guide the planner as to what block to pick up.

[4]Here, we number the action and proposition levels separately, so that there will be an action level 1 and a proposition level 1

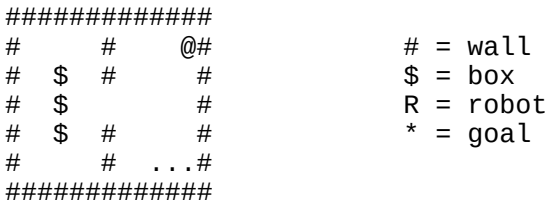

Figure 1: A Sokoban Problem for Landmarks

to a proposition only if it applies to all actions that add that proposition). Each action node in the graph, after the first level, is labeled with the union of the labels on its predecessor proposition nodes (because a landmark label applies to an action if it applies to *any* of its predecessors). When the propagation is complete, any label on a goal proposition in the final level is a causal landmark for plans achieving the goal.

We performed experiments of our landmark extraction algorithm and the method *ff-L* described in (Porteous, Sebastia, & Hoffmann 2001) on a set of domains. In most cases, our algorithm computes a set of nontrivial[5] landmarks significantly larger than *ff-L*. We only show one example here. On the *Sokoban*[6] problem shown in Figure 1, while *ff-L* finds out only trivial landmarks, our algorithm finds out all the landmarks[7].

In the following section, we introduce landmark counting based on a new propagation. And in the final section, we will discuss ideas of utilizing the landmarks computed by methods of this section.

Landmark Counting and Heuristic Search

In this section, we first introduce a method for incorporating landmarks into a heuristic-search planner. We start by naively counting the landmarks discussed in the previous section to construct a heuristic, discuss various ideas to make the landmarks more accurate, and finally develop a new propagation method for *landmark counting* that is used to calculate an improved heuristic. Finally, we present a preliminary empirical comparison of the landmark-counting heuristic to FFPLAN's heuristic and discuss future directions.

Our method is based on the heuristic-search planner FFPLAN(Hoffmann & Nebel 2001). The success of FFPLAN mainly comes from its efficient and accurate heuristic, and its unique search strategy – *enforced hill-climbing*, which is incomplete but often very fast[8]. Unlike pure hill-climbing which iteratively selects single actions with the best one-step-look-ahead heuristic value (which may be worse than the heuristic value of the current state) and often has difficulty with local minima and plateaus, enforced hill-climbing iteratively use breadth-first search to find *action sequences* that lead to states with heuristic values that are strictly better than the current state. FFPLAN's heuristic is based on reachability analysis and provides an upper bound on the optimal sequential plan length of the *relaxed planning task*. Here we attempt to improve the strength of FFPLAN's heuristic by incorporating information about landmarks.

Incorporating Landmarks into a Heuristic. Landmarks provide information about important subgoals that must occur in any successful plan. Intuitively, counting the number of unachieved action landmarks from a state provides an estimate of the number of critical steps ahead, which may provide a "bigger-vision view" than heuristics that count all actions.

Here we improve upon this naive counting heuristic by combining the action and proposition landmark information to obtain a count strictly more complete (i.e. closer to the true plan length). The method is motivated by two observations.

First, there are situations where action landmarks are not sufficient to represent "landmarks" that are naturally present to human. For example, in the blocks world, if block A is being held and the goal is to pick up B, then A must be put somewhere to make the hand empty. The concept of "remove A from the hand" consists of a set of actions and can not be represented by any one of them. However, we observe that actions in this set all contain the proposition HOLDING(A) in their delete lists. We thus introduce propositions as virtual actions and use them to either represent the set of actions achieving them or the set deleting them.

Second, we can often infer that an action must be performed a certain number of times in every successful plan or that a proposition must change its truth value a certain number of times [9]. We want these counts to be reflected in our heuristic and we will do this by developing propagation algorithms for the counts. One idea for propagating counts is to replace the intersection and union operations (or similarly AND and OR operations) from the previous propagation algorithms with maximum or minimum operations on counts. However, we can do more. For an action node, if all of its preconditions claim counts on a proposition A such that 5 is the largest, and if A is in the action's delete list, it is safe to claim that A must change its truth value at least 6 times after the action is taken. We now formalize this idea with the following new propagation method. Space precludes giving all of the details.

The propagated labels are counts on action occurrences or proposition transitions. A node N is labeled with count C on an action A at level i in the plan graph to represent the

[5] A landmark is trivial if it is immediately suggested by the START and FINISH actions.

[6] *Sokoban* is a puzzle involving pushing boxes on two-dimensional boards consisting of "walls" and "corridors" in which a robot must push all the blocks into goal positions through corridors. With the lack of a "pulling" action, ill-chosen moves can lead to deadlocked positions.

[7] All the landmarks are around the door essentially claiming that the human must go through that door and push blocks back through the door.

[8] In case the *enforced hill-climbing* fails, which doesn't happen often, it resorts to an expensive but complete search.

[9] As an example, HANDEMPTY must change its truth value 7 times to solve the *Sussman anomaly*.

claim that any i-step parallel plan achieving N must contain at least C occurrences of A. Similarly, counts on propositions claim a lower bound on the number of corresponding proposition transitions. In the initial graph level, an action level, the START action gets a count of 1 and all other actions and propositions get a count of 0. For each proposition node, every action or proposition gets a count equal to the largest of the counts of that action or proposition on the predecessor action nodes, with the count on the proposition itself adjusted to true. To adjust a count to true, the count is replaced with the smallest odd number that is greater than or equal to it. (Adjusting a count to false is similarly defined.) On each action node after the first level, each action or proposition gets a count equal to the smallest of the counts on the predecessor proposition nodes, with all propositions in the node's precondition set and add list adjusted to true, and all the propositions in the node's delete list adjusted to false. When the propagation is complete, counts are extracted from the FINISH action on the final level[10].

Extracting the Heuristic. To use the counts on both action occurrences and proposition transitions, we give a lower bound of the number of actions to achieve those landmarks. The virtual actions represented by proposition transitions may overlap with each other and with the real actions. For example, (stack A B) can achieve both of the virtual actions "clear A from hand" and "stack something on B". We cast the heuristic computation as generalized bin-packing. We view virtual and real actions as items, where each proposition item has an achiever set representing the set of real actions that can achieve the proposition and each real action item has an achiever set consisting of itself. Two items can be packed into the same bin only if the intersection of their achiever sets is nonempty. The problem is to determine how many bins are necessary to pack all of the items. We observe that if we label bins with real actions, an item can only be packed into the bins with labels in its achiever set.

We develop a set of sound simplification rules for the problem motivated by (Korf 2002):

1. If an item can be packed into only one type of bin (i.e., it has a singleton achiever set), then pick a new bin for that item and pack as many other items into it as possible.
2. If one label A dominates label B, i.e., every achiever set containing B also contains A, then B is discarded from all the achiever sets.

We iteratively apply these sound rules and when no rule applies we greedily select a bin in which as many items as possible are packed. The resulting number of bins is our heuristic value. Alternatively, we can apply these sound rules as a preprocessing and view the rest problem as an integer programming instance for which we can use linear programming to efficiently compute a lower bound.

Empirical Results. We call our implementation of the heuristic-search planner LC, and compare it with the state-of-the-art planner FFPLAN 2.3. FFPLAN has a highly optimized C implementation and LC is a preliminary prototype written in Scheme. We tested the programs on machines running Red Hat 7.2 Linux with dual 1.6GHz Athlon CPUs and 3.5G main memory, giving each problem instance a 24 hours time bound.

[10] As an example, for the Sussman anomaly, our algorithm will compute a count of 5 for HANDEMPTY

	FFPLAN (in C)			LC (in Scheme)		
	States	Time	Len	States	Time	Len
g1	60	0.02	19	140	11	17
g2	1,560	0.17	123	576	146	53
g3	7,959	1.34	243	1,219	831	103
g4	2,962	0.78	217	1,935	2,592	161
g5	-	-	-	2,698	6,503	227
g6	-	-	-	3,559	15,544	301
bw8-0	25	0.02	18	79	3	20
bw8-1	101,430	4.53	32	117	5	26
bw8-2	23	0.02	16	68	3	16
bw9-0	146,624	7.29	30	305	15	34
bw9-1	189	0.01	28	125	7	30
bw9-2	34	0.01	26	126	7	26
bw10-0	63	0.02	34	667	45	34
bw10-1	8,726	0.38	38	1,324	91	34
bw10-2	313	0.02	34	950	68	36
bw11-0	83	0.02	34	431	40	40
bw11-1	4,859,941	4,008.54	46	347	34	30
bw11-2	44	0.02	34	410	37	34
bw12-0	1,743	0.09	44	177	32	36
bw12-1	54	0.01	34	914	111	38
bw13-0	75	0.03	42	1,216	203	42
bw13-1	85,942	6.14	46	1,148	179	44
bw14-0	99	0.03	40	313	71	50
bw14-1	733	0.05	42	477	99	38
bw15-0	-	-	-	2,439	640	40
bw15-1	153	0.03	52	31,249	8,901	54
log6-0	36	0.02	25	323	3	25
log6-1	18	0.02	14	158	2	14
log8-0	48	0.02	31	563	21	32
log8-1	76	0.02	44	926	30	48
log10-0	90	0.04	46	1390	89	50
log10-1	72	0.03	42	988	65	43
log12-0	82	0.04	42	1281	87	46
log12-1	161	0.03	73	2359	162	79

Table 1: *Comparing* FFPLAN *2.3 to* LC *on three domains. The first 6 rows correspond to Sokoban problems, the next 20 rows to Blocks-world problems, and the remaining 8 rows to Logistics. The three columns for each planner show the number of states that were evaluated, the time cost in seconds for search and the plan length respectively. Missing elements indicate that the planner didn't finish in 24 hours.*

We tested both methods on three domains. The upper part of Table 1 (problems g1 to g6) shows the results for *Sokoban* domain. Our tests are based on a series of problems similar to that shown in Figure 2 with the number in the problem name representing the number of blocks.

To achieve the goal, the robot must push the blocks in sequence through the door on the middle wall, and not allow any block to touch any wall except the middle ones. Our landmark extraction can successfully find all the landmarks (which are mainly near the door and along the middle line of the upper room), and intuitively, those landmarks represents the important subgoals. Thus, the landmark counts can

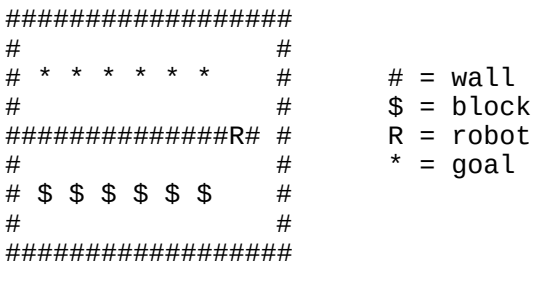

Figure 2: A Sample Sokoban Problem

efficiently guide the search. As shown in Table 1, the number of evaluated states grows almost linearly with the plan lengths, which indicates that the heuristics are very informative. The time cost grows more dramatical because as the number of proposition and actions grows, the time cost of each heuristic calculation also grows. As a comparison, FFPLAN's search space explodes much more quickly.

The middle part of Table 1 (problems bw8-0 to bw15-1) shows *Blocks-world* domain results for all the problems with 8-15 blocks in the *AIPS-2000 Planning Competition.* In this domain, FFPLAN does well in most problems, but performs very badly on some of the problems. As a comparison, LC searches more than 3,000 states only in one problem.

The lower part of Table 1 (problems log6-0 to log12-1)shows *Logistics* domain results for problems from the *AIPS-2000 Planning Competition.* LC performs somewhat slower than FFPLAN in this domain, but remains comparable, especially in comparison to the difference on the *Sokoban* domain.

We do observe that LC's heuristic calculation is much slower than FFPLAN's. The efficiency difference between C and Scheme is one reason for this. In addition, although our landmark counting and FFPLAN's reachability analysis share the structure of a linear graph, our method propagates vectors of numbers while FFPLAN propagates only Boolean values. To make our method more efficient, we are investigating incremental computations of our heuristic.

Incremental Heuristic Computation. Making the heuristic calculation more efficient is a critical issue for heuristic planning.

In the earlier part of this paper, we presented a simplification of the propagations we actually implement. In particular, there is no need to keep a copy of every level of every action or proposition. We can repeatedly compute a new level from the previous level, discarding old levels—the resulting propagation becomes a fixed-point computation. In this view, we can leverage (but do not do so for the results shown here) an incremental shortest path algorithm, DynamicSWSF-FP, developed recently(Ramalingam & Reps 1996)(Liu, Koenig, & Furcy 2002). While our heuristic calculation does not exactly meet their requirements and thus does not get the theoretical guarantees of their algorithm, we have preliminary experimental results showing that our calculation gains the practical benefits of DynamicSWSF-FP. We are currently exploring this direction.

Another future direction is to combine the heuristics of FFPLAN and LC. Since landmark counting intuitively provides a "bigger-vision view" but has bigger plateaus, we consider FFPLAN's heuristic only when landmark counting cannot distinguish.

Future Directions

In this section, we briefly discuss other ideas of utilizing landmarks.

Extracted landmarks can to used as subgoals to guide plan search. In this approach, it's critical to decide correct ordering of the landmarks. In our propagation, since every proposition and action gets a label of its own landmarks, the propagation can thus get a sound partial ordering as side effect. We are investigating ways to combine these orderings with those introduced in (Koehler & Hoffmann 2000) and (Porteous, Sebastia, & Hoffmann 2001), as well as other propagations that can possibly extract other ordering information.

The extracted landmarks and their ordering soundly construct a partial plan that doesn't need to be backtracked, thus we are also interested in combining this with local-search planners like LPG(Gerevini & Serina 2002).

References

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

Gerevini, A., and Serina, I. 2002. Lpg: a planner based on local search for planning graphs. In *Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02)*.

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

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research* 12:338–386.

Korf, R. E. 2002. A new algorithm for optimal bin packing. In *Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, 731–736.

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

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, 37–48.

Ramalingam, G., and Reps, T. W. 1996. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms* 21(2):267–305.