The LAMA Planner
Using Landmark Counting in Heuristic Search

Silvia Richter
Griffith University, Queensland, Australia
and
NICTA, Queensland, Australia
silvia.richter@nicta.com.au

Matthias Westphal
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Freiburg, Germany
westpham@informatik.uni-freiburg.de

Abstract

LAMA is a propositional planning system based on heuristic search. Its core feature is the use of a pseudo-heuristic derived from *landmarks*, propositions that must be true in every solution of a planning task. It builds on the Fast Downward Planning System, using non-binary (but finite domain) state variables, and *multi-heuristic* search. A weighted A* search is used with iteratively decreasing weights, so that the planner continues to search for plans of better quality until the search is terminated.

Introduction

LAMA is a planning system based on heuristic state space search, in the spirit of HSP, FF and Fast Downward (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert 2006). It builds on the Fast Downward System (Helmert 2006), inheriting the general structure of Fast Downward, the translation of PDDL tasks with binary state variables to representations with multi-valued state variables, and a search architecture that is able to exploit several heuristics simultaneously. The core feature of LAMA is the use of *landmarks* as a pseudo-heuristic and for generating preferred operators. The *landmark counting* heuristic was introduced in a AAAI 2008 article (Richter, Helmert, and Westphal 2008).

Structure of the Planner

LAMA consists of three separate programs:

1. the *translator* (written in Python),
2. the *knowledge compilation* module (written in C++), and
3. the *search engine* (also written in C++).

To solve a planning task, the three programs are called in sequence; they communicate via text files.

Translator

The purpose of the *translator* is to transform the planner input, specified in the propositional fragment of PDDL (including ADL features and derived predicates, but not the preferences and constraints introduced for IPC-5), into a multi-valued state representation similar to the SAS$^+$ formalism (Bäckström and Nebel 1995).

The main components of the translator are an efficient grounding algorithm for instantiating schematic operators and axioms and an invariant synthesis algorithm for determining groups of mutually exclusive facts. Such fact groups are consequently replaced by a single multi-valued state variable encoding *which* fact (if any) from the group is satisfied in a given world state.

The groups of mutually exclusive facts found during translation serve an important purpose for determining orders between landmarks. This is why LAMA does not read the multi-valued state representations offered at ICP-6 (object fluents) directly, but instead performs its own translation from the traditional (binary-variable) representation to SAS$^+$.

For more details on the translator component, see the article on Fast Downward by Helmert (2006). We have modified this component only in some minor ways, including the extraction of all mutually exclusive facts (as mentioned above), the handling of action costs for IPC-6, and some small enhancements.

Knowledge Compilation

Using the multi-valued task representation generated by the translator, the *knowledge compilation* module is responsible for building a number of data structures which play a central role in the subsequent landmark generation and search.

For example, *domain transition graphs* are produced which encode for each state variable the ways in which it may change its value through operator applications. Furthermore, the knowledge compilation module constructs *successor generators* and *axiom evaluators*, data structures for efficiently determining the set of applicable actions in a given state of the planning task and for evaluating the values of derived state variables. Again, we refer to Helmert (2006) for more detail on the knowledge compilation component.

Search Engine

Using the data structures generated by the knowledge compilation module, the *search engine* attempts to find a plan using heuristic search with some enhancements, such as the use of *preferred operators* (similar to helpful actions in FF) and *deferred heuristic evaluation*, which mitigates the impact of large branching factors in planning tasks with fairly

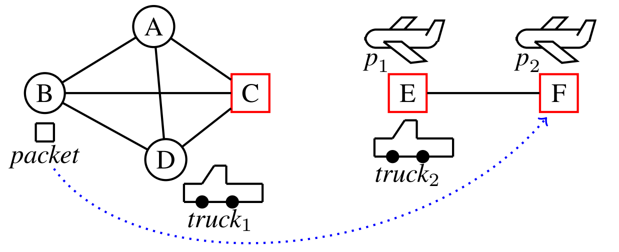

Figure 1: A logistics task: transport packet x from B to F.

accurate heuristic estimates. Deferred heuristic evaluation means that states are not evaluated upon generation, but upon expansion. States are thus not selected for expansion according to their own heuristic value, but according to that of their parent. If many more states are generated than expanded, this leads to a substantial reduction in the number of heuristic estimates computed, if at a loss of heuristic accuracy.

The rules of the 6th International Planning Competition (IPC-6), for which LAMA was designed, suggest a type of search that takes plan quality into account. LAMA first runs a greedy best-first search, aimed at finding a solution as quickly as possible. Once a plan is found, it searches for progressively better solutions by running a series of weighted A* searches with decreasing weight. The cost of the best known solution is used for pruning the search, while decreasing the weight makes the search progressively less greedy, trading speed for solution quality.

The search engine is configured to use several heuristic estimators (namely, the FF heuristic and the landmarks pseudo-heuristic) within an approach called *multi-heuristic search* (Helmert 2006). This technique attempts to exploit strengths of the utilised heuristics in different parts of the search space in an orthogonal way. To this end, it uses separate open lists for each of the different heuristics as well as separate open lists for the preferred operators of each heuristic. Newly generated states are evaluated with respect to all heuristics, and added to all open lists (with the value estimate corresponding to the heuristic of that open list). When choosing which state to expand next, the search engine alternates between the different heuristics, and expands states from preferred-operator queues with higher priority than states from other queues.

Landmarks

Landmarks are variable assignments that must occur at some point in every solution plan. They were first introduced by Porteous, Sebastia and Hoffmann (2001) and later studied in more depth by the same authors (Hoffmann, Porteous, and Sebastia 2004). Consider the logistics example task in Fig. 1, where the goal is to transport the packet from location B to location F. In order to achieve the goal, the packet must be loaded onto *truck*$_1$ and unloaded at the airport C, in order to then be loaded into one of the planes p_1 or p_2. Hence, the facts "*packet* is on *truck*$_1$", and "*packet* is at C"

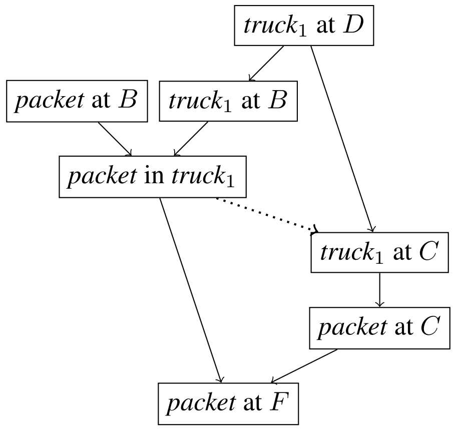

Figure 2: Partial landmark graph for the example task in Fig. 1, showing simple landmarks.

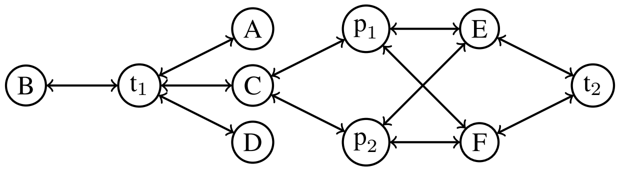

Figure 3: Domain transition graph for the packet from Fig. 1.

are landmarks for this task. It is also possible to infer orders between landmarks, e. g. in this case, that "*packet* is on *truck*$_1$" must be true *before* "*packet* is at C". The landmarks and orders can be stored in a directed graph called the landmark graph. For our example, a partial landmark graph is depicted in Fig. 2.

Our algorithm for finding landmarks and orderings between them is similar to the one by Porteous and Cresswell (2002), which is in turn based on the one by Hoffmann et al. Like Porteous and Cresswell we admit *disjunctive* landmarks (sets of propositions of which one needs to be true at some point), but we adapted the algorithm to the SAS$^+$ setting and use domain transition graphs to derive further landmarks.

We find landmarks by backchaining from already known landmarks, starting with the goals (which are landmarks by definition, as they have to be true in every solution plan). For any given landmark L that is not true in the initial state, we consider the *shared preconditions* of its *possible first achievers*. Possible first achievers are those operators that a) have L as an effect, and b) can be possibly applied at the end of a partial plan (starting in the initial state) which has never made L true. Their shared preconditions are those propositions (if any) that are a precondition for each of the operators. Every such shared precondition must be true in order to reach L and is thus a landmark, which can be ordered before L.

Since it is PSPACE-hard to determine the set of *actual* first achievers of a landmark L, we use an over-approximation containing every operators that can *possibly* be a first achiever. By intersecting over the preconditions of more operators we do not loose correctness, though we may of course miss out on some landmarks. The approximation of first achievers of L is done with the help of a relaxed planning graph (RPG) (Hoffmann and Nebel 2001). During construction of the RPG we leave out any operator that would add L. When the relaxed planning graph levels out, its last set of facts is an over-approximation of the set of facts that can be achieved *before* L in the planning task; we denote it by $pb(L)$ (for *possibly before*). Any operator that is *applicable* given $pb(L)$ and achieves L is a possible first achiever of L.

We also create disjunctive sets of facts from the first achievers' shared preconditions, such that a set contains one precondition fact from each first achiever. Hence, these sets form disjunctive landmarks. All facts in a disjunctive landmark must stem from the same predicate symbol, and we discard any sets of size greater than 4 in order to limit the number of possible sets.

We generate further landmarks by exploiting the domain transition graphs (DTGs) generated by the knowledge compilation module. For each variable, a corresponding DTG has a node for each value that can be assigned to the variable, and arcs for possible transitions between them (where a transition can be achieved through operator application). For example, assume that the location of the packet in our example is encoded with a state variable v. The DTG of v is depicted in Fig. 3. From its initial value B, the location of the packet can change to "in *truck*$_1$" (denoted as t_1 for short), from there to any of the locations A, B, C and D and so on.

Given a *simple* (i. e., non-disjunctive) landmark $L = \{v \mapsto l\}$ that is not part of the initial state, we consider the DTG of the landmark's variable v. If there is a node that occurs on *every* path from the *initial state value* $s_0(v)$ of the variable to the *landmark value* l, then that node corresponds to a landmark value l' of the variable: We know that every plan achieving L requires that v take on the value l', hence the fact $L' = \{v \mapsto l'\}$ can be introduced as a new landmark and ordered before L. To find these kinds of landmarks, we iteratively remove one node from the DTG and test with a simple graph algorithm whether $s_0(v)$ and l are still connected – if not, the removed node corresponds to a landmark. Nodes corresponding to assignments of v which are not in $pb(L)$ are removed from the DTG prior to this test, as they can only occur *after* B and do not have to be tested. However, we remember these nodes and if such a node is later found to be a landmark (e. g. by the backchaining procedure), we can introduce an ordering between B and the node.

Consider again the landmark graph of our example in Fig. 2. Most of the landmarks and orders in it can be found by the backchaining procedure even when restricting it to simple, i. e., non-disjunctive, landmarks, because the propositions are direct preconditions of their successor nodes in the graph. There are two exceptions: "*packet* in *truck*$_1$" and "*packet* at C". These two landmarks are however found with the DTG method. The DTG in Fig. 3 shows immediately, that the package location must be both t_1 and C on any path from the initial state (where it has value B) to the goal F.

If we introduced another truck in the left city, the fact "*packet* in *truck*$_1$" would no longer be a landmark. However, using disjunctive landmarks we would get a landmark for the packet being inside one of the two trucks.

Inconsistencies found in the translating phase are exploited to determine further orders between the landmarks, using the definition of *reasonable orders* and the generation conditions proposed by Hoffmann et al. (2004). For example, the order depicted by a dotted line in Fig. 2 is such a reasonable order. For more details on how landmarks and their orders are derived, see the AAAI 2008 article (Richter, Helmert, and Westphal 2008).

The Landmark Counting Heuristic

The LAMA planning system uses landmarks as a pseudo-heuristic. We estimate the goal distance of a state s by the number of landmarks l that still need to be achieved from s onwards. We estimate this number as $\hat{l} := n - m + k$, where n is the total number of landmarks, m is the number of landmarks that are *accepted*, and k is the number of accepted landmarks that are *required again*. A landmark B is accepted in a state s if it is true in that state and all landmarks ordered before B are accepted in the predecessor state from which s was generated. An accepted landmark remains accepted in all successor states. An accepted landmark is *required again* if it is not true in s and it is a direct precondition of some landmark which is not accepted. Note that $\hat{l}$ is not a proper state heuristic in the usual sense, as its definition depends on the way s was reached during search. Nevertheless, it can be used like a heuristic.

We also generate preferred operators along with the landmark heuristic. An operator is preferred in a state if applying it achieves an *acceptable* landmark in the next step, i. e., a landmark whose predecessors have already been accepted. If no acceptable landmark can be achieved within one step, the preferred operators are those which occur in a relaxed plan to the nearest acceptable landmark.

Integrating Action Costs

The landmark heuristic as outlined above estimates the goal distance of states, i. e., the number of operator applications needed to reach the goal state from a given state. Due to the inclusion of action costs in IPC-6, however, we are interested in generating least cost plans rather than short plans. Hence, the heuristics used during search should estimate the *cost* of reaching the goal from a state rather than its goal distance.

The FF heuristic that is also used in our framework can easily be adapted to action costs, as we can use action costs in the underlying *additive heuristic* (Bonet and Geffner 2001). When generating a relaxed plan from the additive heuristic estimates, we simply use the cost rather than the length of that relaxed plan as our estimate for the cost-to-go of a given state. See Keyder and Geffner (2008) for a detailed description of this cost-sensitive version of the FF heuristic which they call FF(h_a).

For the landmarks pseudo-heuristic this method is not directly applicable, since no actual plan is formed by the heuristic. Instead, we *weigh* landmarks with an estimate on their minimum cost. Rather than counting the number of landmarks that still need to be achieved from a state, the heuristic value is then the sum of all minimum costs of those landmarks. The cost estimate for each landmark is the minimum cost that is required to make the landmark true for the first time, i. e., the minimum of all action costs of its first achievers.

Zero-cost actions can lead to problems in a standard cost-sensitive search like weighted A*. Since zero-cost actions can always be added to a search path “for free”, i. e., without negative side effects, the search may explore very long search paths without getting closer to a goal. In the worst case, this can prevent it from finding a solution within the given time limit. To avoid this problem, LAMA adds a constant of 1 to all action costs. Of course this means that a plan consisting of 5 originally zero-cost actions is deemed worse by LAMA than a plan consisting of 2 action which originally cost 1, whereas the opposite is true. A smaller value for the constant add-on would lessen the problem, though not solve it completely.

Acknowledgements

We would like to acknowledge the support of NICTA. NICTA is funded through the Australian Government’s *Backing Australia’s Ability* initiative, in part through the Australian Research Council.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

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

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proc. ECAI 2008*.

Porteous, J., and Cresswell, S. 2002. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG ’02)*, 45–54.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Proc. ECP 2001*, 37–48.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI 2008*, 975–982.