Landmark-Based Heuristics and Search Control for Automated Planning (Extended Abstract)

Silvia Richter
Dematic, Offenbach, Germany
and
NICTA, Australia
silvia.e.richter@gmail.com

The dissertation on which this extended abstract is based was the recipient of the ICAPS Best Dissertation Award 2012 [Richter, 2010.]

Abstract

Automated planning is the process of automatically selecting actions that achieve a desired outcome. This paper summarises several contributions that improve the efficiency of automated planning via heuristic search. We discuss novel heuristics based on *landmarks* and a search algorithm for anytime planning. Furthermore, we analyse various search-enhancement techniques and show how the combination of these techniques lead to a planning system that proved highly successful in the 2008 and 2011 International Planning Competitions.

1 Introduction and Background

Given a declarative description of a task, a planning system finds an action sequence (a plan) that leads from a given initial state to a state that satisfies a specified goal description. We consider planning in the SAS$^+$ planning model [Bäckström and Nebel, 1995], where the state space is given by propositional variables with finite domain, and actions transform one state into another. The quality of a plan is measured via its length or via associated costs of the actions it comprises. We will in the following assume the traditional unit-cost planning (where all actions have an associated cost of 1) unless otherwise noted, and refer to the alternative as *cost-based* planning.

Although the planning problem in general is computationally intractable, many problem instances can be solved efficiently by exploiting their structure. Knowledge about such structure or certain properties of a planning task, so-called *control knowledge*, can often be extracted automatically from the problem description.

A useful tool for making the structure of SAS$^+$ planning tasks explicit is the *domain transition graph* that captures the ways in which the value of a given state variable v may change [Jonsson and Bäckström, 1998; Helmert, 2006]. It is a directed graph with vertex set $\mathcal{D}_v$ (the domain of v) that contains an arc between two nodes d and d' if there exists an action that can change the value of v from d to d' [Helmert, 2004].

A *landmark* is a subgoal that must be achieved in every plan [Hoffmann *et al.*, 2004], i. e., a propositional formula over the facts of the planning task that must be satisfied at some point during the execution of any plan for the task. Note that facts in the initial state and facts in the goal are always landmarks by definition. We restrict ourselves to fact landmarks and disjunctive landmarks (disjunctions of facts) in the following, unless otherwise noted.

Various kinds of *orderings* between landmarks can be defined and exploited during the planning phase [Hoffmann *et al.*, 2004]. For example, we may be able to determine that a given landmark φ must be reached *before* another landmark ψ in any plan (natural ordering) or even that φ must be true *one step before* ψ becomes true *for the first time* (greedy-necessary ordering). A set of landmarks and associated orderings between them can be represented in a *landmark graph*.

2 Landmarks

In this section, we present novel methods for extracting *landmarks*, a particular type of control knowledge, from planning tasks and propose a way of using them as a heuristic estimator for judging progress during planning. We furthermore analyse the performance gain achieved via landmarks in *cost-based* planning and find that landmarks can be particularly helpful in this setting.

Finding Landmarks

Deciding whether a given formula is a landmark and deciding orderings between landmarks are PSPACE-hard problems. Thus, practical methods for finding landmarks are typically incomplete. In general, landmarks can be generated from a set of known landmarks (e. g. the facts in the goal) through backchaining. For example, if a fact B is a landmark that is not already true in the initial state and all achievers of B (actions that have B as an effect) share a common precondition A, then A is also a landmark [Hoffmann *et al.*, 2004].

We propose a tractable algorithm for finding fact landmarks and disjunctive landmarks that is partly based on the backchaining method by Hoffmann *et al.* [2004] mentioned above, adapting it to the SAS$^+$ setting. In addition, our algorithm exploits the SAS$^+$ representation by using domain transition graphs (DTGs) and other techniques to find further landmarks and orderings.

Landmarks can be derived from DTGs as follows: given a fact landmark $\psi = \{v \mapsto l\}$ (where variable v takes on the value l) that is not part of the initial state s_0 of the planning task, consider the DTG of v. The nodes of the DTG correspond to the values that can be assigned to v, and the arcs to the possible transitions between them. If there is a node l' that occurs on *every* path from the *initial state value* $s_0(v)$ to the *landmark value* l, then that node corresponds to a landmark value l' of v: We know that every plan achieving ψ requires that v takes on the value l', hence the fact $\varphi = \{v \mapsto l'\}$ is a landmark that can be naturally ordered before ψ.

Our landmark procedure finds slightly more landmarks and many more orderings, on average, than previous procedures and has shown to lead to better solution quality and comparable coverage when used in an otherwise identical setting [Richter *et al.*, 2008].

Using Landmarks as a Heuristic

Previously, landmarks have been used as subgoals [Hoffmann *et al.*, 2004]. With that approach, the solution to a planning task is found by solving subtasks given by the landmarks and concatenating the plans for these subtasks. However, this approach is incomplete and exhibits unsatisfactory performance in several planning domains. We developed a complete approach that uses landmark information directly in computing the heuristic values of states during plan search.

We approximate the goal distance of a state s by the estimated number of landmarks that still need to be achieved from s onwards. Given a sequence of actions π resulting in s, the heuristic value of s is given by

$$h_{LM}(s, \pi) := (L \setminus \mathit{Reached}(s, \pi)) \cup \mathit{ReqAgain}(s, \pi)$$

where L is the set of all discovered landmarks, $\mathit{Reached}(s, \pi)$ is the set of *reached* landmarks, and $\mathit{ReqAgain}(s, \pi)$ is the set of reached landmarks that are *required again*, with the following definitions based on a given landmark graph $G = (L, O)$: A landmark φ is first reached in a state s if it is true in that state, and all landmarks ordered before φ in G are reached in the predecessor state from which s was generated. Once a landmark has been reached, it remains reached in all successor states. For the initial state, reached landmarks are all those that are true in the initial state and do not have any predecessors in G. A reached landmark φ is *required again* if it is not true in s and (a) it forms part of the goal or (b) it is a direct precondition for some unreached landmark ψ, i. e., there is a greedy-necessary ordering between φ and ψ and ψ is not reached in s. (Note that $h_{LM}(s, \pi)$ is not a proper state heuristic in the usual sense, as its definition depends on the way s was reached during search.)

Using this landmark counting in greedy best-first search already leads to good results in some cases. However, the amount of useful landmarks varies substantially from one planning domain to the other, and the landmark heuristic in this simple form is not competitive with established planning heuristics like the FF heuristic [Hoffmann and Nebel, 2001]. Results can be substantially improved by combining landmark counting with other heuristics in a multi-heuristic search [Helmert, 2006] and by using *helpful actions* / *preferred operators*.

	Algorithm		
Base Heuristic	base	LM-local	LM-heur
FF heuristic	87	82	**88**
CG heuristic	74	66	**87**
blind heuristic	25	52	**84**

Table 1: Coverage results for the landmark heuristic and alternative approaches.

Preferred operators are actions that are deemed likely to improve the heuristic value from a given state (e. g. because they contribute to solving a simplified version of the task and may thus be considered likely to also be vital for the original task). The "preferredness" of actions can be used as a criterion for determining the order of generation for new states, complementing the information given by heuristic state values. Exploiting preferred operators in planning has been shown to improve results notably [Helmert, 2006; Hoffmann and Nebel, 2001].

We take an action to be a preferred operator in a state if applying it achieves a *reachable* landmark in the next step, i. e., a landmark whose predecessors have already been reached. If no reachable landmark can be achieved within one step, the preferred operators are those actions which occur in a *relaxed plan* [Hoffmann and Nebel, 2001] to the nearest reachable fact landmark.

Tab. 1 contains experimental results for a large set of benchmark tasks from past International Planning Competitions (IPCs). Shown is the coverage (average percentage of tasks solved) when using our approach of combining the landmark heuristic with a base heuristic (LM-heur); a search just using the base heuristic (base); and the previous approach of using landmarks as subgoals in a search with the base heuristic (LM-lokal). We compare the three approaches with three different base heuristics. With all three, LM-heur results in best coverage and furthermore improves solution quality [Richter *et al.*, 2008].

Landmarks in Cost-Based Planning

Planning with non-unit action costs has proven to be a considerable challenge to automated planners due to the difficulty of balancing runtime vs. solution quality. We find that a cost-sensitive variant of the popular FF heuristic that had been proposed for cost-based planning [Keyder and Geffner, 2008], while leading to higher-quality solutions than its cost-unaware counterpart, results in solving far fewer tasks, and hence does not pay off in standard IPC settings. Using landmarks in combination with this heuristic, however, proves to be particularly helpful, as the landmarks mitigate the poor coverage of the cost-sensitive FF/add heuristic (see Tab. 2) while typically causing little detriment to plan quality [Richter and Westphal, 2010]. This fact plays a significant role in the success of the LAMA planner (see Section 4).

3 Search Strategies and Enhancements

In this section, we focus on improving the underlying *search algorithms* to increase coverage and solution quality. We summarise a study of two popular search-control techniques,

Domain	F	F_c	FL	FL_c	FF	FF(h_a)
Cyber Security	30	28	30	30	4	23
Elevators	30	15	30	22	30	23
Openstacks	30	30	30	30	30	25
PARC Printer	25	16	24	23	30	16
Peg Solitaire	30	30	30	30	30	29
Scanalyzer	28	30	30	30	30	28
Sokoban	25	25	23	24	27	17
Transport	26	22	29	30	29	23
Woodworking	30	28	28	30	17	29
Total	**254**	224	**254**	249	227	213

Table 2: Coverage for cost-based planners (F_c, FL_c and FF(h_a)) and planners ignoring costs (F, FL and FF) on IPC 2008 benchmarks. F, F_c, FL and FL_c are four variations of the same planner using the FF/add heuristic, where c denotes cost-based planning and L denotes the use of landmarks.

Search	PO Use	Coverage Score	Eval. Score	Quality Score	Time Score
Lazy	No POs	74.03	54.34	68.19	67.63
	Heur > PO	79.11	62.92	72.65	72.29
	PO > Heur	83.71	70.75	75.40	79.08
	PO Pruning	84.02	70.52	76.01	79.49
	Dual Queue	83.43	65.11	77.00	76.14
	B. Dual Queue	**86.74**	**72.58**	**78.83**	**81.95**
Eager	No POs	75.07	52.42	72.08	68.08
	Heur > PO	75.80	54.28	72.24	69.31
	PO > Heur	81.42	58.20	76.86	74.66
	PO Pruning	84.64	**68.12**	79.33	**79.74**
	Dual Queue	**86.89**	60.91	**83.22**	79.17
	B. Dual Queue	83.65	58.60	79.22	75.96

Table 3: Results for the FF heuristic. A coverage score of 74.03 means that the configuration solves on average 74.03% of the tasks in a randomly picked domain.

preferred operators and *deferred evaluation*, and propose a novel algorithm for *anytime planning*.

Analysis of Search Enhancements

As described in Section 2, *preferred operators* have been used in various successful planning systems. Another search technique that has become popular in planning is to use a variation of best-first search called *deferred evaluation* [Helmert, 2006], where the successors of a state are not evaluated upon generation, but upon expansion. States are inserted into the open list with the heuristic estimate of their parent. According to Helmert [2006], this technique can decrease the number of costly state evaluations substantially and thus increase performance, especially when combined with preferred operators.

Until recently, little data existed on the respective usefulness of preferred operators and deferred evaluation. For example, the use of preferred operators (POs) in Fast Downward is different from their use in FF. While the authors of both planners report significantly improved performance compared to *not* using POs, the question remained which usage is better, or how some of the many other conceivable approaches for using POs would perform. The question was also open whether deferred evaluation is advantageous in the absence of POs or when using POs in a different fashion than the one examined by Helmert.

We address these questions by examining the performance of various approaches to using POs as well as their interaction with deferred evaluation. The methods for using POs we compare are the following:

Pruning. Like the FF planner, use POs to prune the search space and only evaluate successor states reached via POs. (Note that this restriction makes the search incomplete.)

Dual queue. Like the Fast Downward planner, keep the states that are reached via a preferred operator in a second open list. Select the next state to be expanded alternately from the regular open list containing all successors and from the open list containing just preferred successors, pruning duplicates upon removal. This method may gain less leverage from preferred operators than pruning, but the search stays complete.

Boosted dual queue. Like the IPC 2004 version of Fast Downward, use a modified dual-queue approach that gives higher precedence to the queue containing preferred successors.

Pref.* > *heur. Like the YAHSP planner [Vidal, 2004], choose a preferred successors whenever one exists, otherwise expand non-preferred successors. Within each of the two groups, choose according to heuristic values. This method is similar to pruning, but not as restrictive.

Heur.* > *pref. Use preferred operators for tie-breaking and expand, among states of equal heuristic value, preferred successors first. This method places the least emphasis on preferred operators.

We conducted experiments with various heuristics (h_{FF}, h_{add}, h_{CG}, h_{cea}) on tasks from IPCs 1–5, using the PO methods detailed above with and without deferred evaluation (where we denote the two search types "Lazy" and "Eager", respectively). The results for the FF heuristic, measured via scores from the range 0–100, are shown in Tab. 3. For details on the scoring mechanism we refer to the literature [Richter and Helmert, 2009]. Overall, the results are remarkably consistent across the different heuristics and can be summarised as follows:

Preferred operators are extremely helpful. With the best PO setting, the number of unsolved tasks can roughly be halved and POs are far more important for performance even than the choice of heuristic.

Best PO use depends on the search type. For Eager, a certain amount of preferred operators is useful, but strong uses of preferred operators can be counter-productive. When using deferred evaluation, stronger use of preferred operators always seems to improve results. This can be explained by the fact that Lazy is less informed by heuristic values and thus stands to gain more from preferred operators than Eager.

Deferred evaluation trades time for quality. The best performance obtainable from each search variant (using the dual-queue setting for Eager, and boosted dual-queue for Lazy) is very similar. Lazy consistently evaluates fewer states than Eager and is usually faster. Eager, on the other hand, finds plans of better quality.

To more closely analyse how the benefit from preferred operators varies with different characteristics of the search space, we conducted a set of experiments on a manually-designed search space. We aimed to create a scenario where preferred operators can be useful but also harmful, depend-

ing on how well the preferredness of an operator predicts that it actually leads to a better state (measured by varying the "recognition rate" in our experiment), and how well informed the heuristic is in comparison.

The results show that performance can be notably improved via preferred operators even if the heuristic has high quality. It is evident that PO pruning suffers more strongly from ill-informed preferred operators than the dual-queue approach. The relative performance of the PO approaches, however, is notably independent of the quality of the heuristic. In our scenario, the dual-queue approach typically becomes better than not using POs if the recognition rate is more than 20–30%; PO pruning has a higher threshold of 40–50%.

Restarting Algorithm for Anytime Planning

Anytime search algorithms solve optimisation problems by quickly finding a (usually suboptimal) first solution and then finding improved solutions when given additional time. To deliver an initial solution quickly, they are typically greedy with respect to the heuristic cost-to-go estimate h. Furthermore, anytime algorithms typically *continue* their search after finding the first solution. For example, anytime algorithms based on Weighted A* (WA*) run a continuous WA* search, possibly adjusting their data structures and parameters after each new solution found. They use the cost of the best known solution for pruning and/or decrease the weight over time (reordering but keeping the open list). Examples for such algorithms are the Anytime WA* algorithm [Hansen and Zhou, 2007] and the ARA* algorithm [Likhachev *et al.*, 2008].

We note that the combination of initial greediness (low-h bias) and continued search can cause poor performance: if the heuristic estimates are inaccurate, the search can be led into an area of the search space that contains no goal or only poor goals. In particular once a goal is found, the continued search often spends much time exploring the search space around the previous goal, since the states there have low h-value. In such cases, *restarting* the search can be beneficial to overcome the bad decisions that were made near the start state.

Our proposed algorithm RWA* runs iterated WA* with decreasing weight. Whenever a better solution is found, it empties the open list and restarts from the initial state. It re-uses search effort by not calculating the heuristic value of a state more than once and by making use of the best path to a state found so far.

While restarts are a well-known technique in *randomised* algorithms (e. g., in SAT and CSPs) to undo bad random parameter choices, our use of restarts in a *deterministic*, A*-type algorithm seems counter-intuitive at first glance. Our results, however, shown in Fig. 1, confirm that it is worthwhile to discard information that may be biased due to greediness.

We implemented each of the search algorithms within the Fast Downward planning framework while keeping all other settings of the planner fixed. The results demonstrate that RWA* dominates other anytime methods in planning, in particular when search enhancements are used. Experiments with a controlled search space furthermore suggest that restarts are particularly useful if the greedy search is highly suboptimal, e. g. if the heuristic values are systematically biased [Richter *et al.*, 2010].

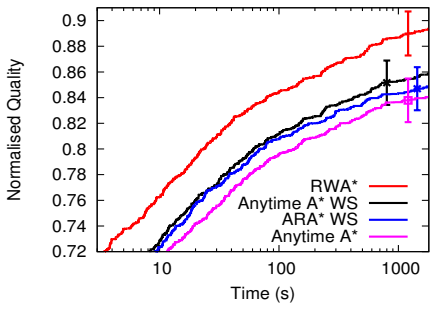

Figure 1: Anytime performance in PDDL planning.

Domain	Base	C3	FF(h_a)	FF(h_a^s)	LAMA
Cyber Security	4	9	20	20	28
Elevators	21	16	9	10	20
Openstacks	21	10	8	8	27
PARC Printer	27	18	16	23	21
Peg Solitaire	20	20	21	23	29
Scanalyzer	24	23	24	24	26
Sokoban	21	18	15	18	24
Transport	18	6	15	14	27
Woodworking	14	24	22	22	25
Total	169	143	150	162	**227**
(Total IPC-08)	(176)	(151)	(157)	(169)	(236)

Table 4: IPC scores for the top planners of IPC 2008.

4 The LAMA Planner and Extensions

The above ideas were combined and integrated into the LAMA planning system. LAMA uses landmarks in addition to a cost-sensitive variant of the FF heuristic, and employs the search enhancements *preferred operators* and *deferred heuristic evaluation* in their most effective combination. It finds a first solution quickly using greedy best-first search and then improves on this solution using restarting anytime search as long as time remains.This planner showed remarkable success, winning the sequential satisficing tracks at the 2008 and 2011 International Planning Competitions by a significant margin.

Tab. 4 shows results for the top planners of IPC 2008 in the competition setting. The performance criterion is the "IPC score" that combines coverage and solution quality in one measure. The advantage LAMA shows over its competitors is partly due to its anytime search. If LAMA is stopped after the first solution, it achieves a score of 182, which still notably dominates the other planners [Richter and Westphal, 2010].

Conjunctive Landmarks

Lastly, we provide an outlook on possible future work by investigating more complex types of landmarks. We show that using higher-order (conjunctive) landmarks can significantly improve the heuristic estimates obtained from a landmark heuristic. However, the additional effort required for finding and using such landmarks does not necessarily pay off [Keyder *et al.*, 2010].

References

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[Hansen and Zhou, 2007] Eric A. Hansen and Rong Zhou. Anytime heuristic search. *JAIR*, 28:267–297, 2007.

[Helmert, 2004] Malte Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS 2004* [2004], pages 161–170.

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

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Hoffmann *et al.*, 2004] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.

[ica, 2004] *Proc. ICAPS 2004*, 2004.

[Jonsson and Bäckström, 1998] Peter Jonsson and Christer Bäckström. State-variable planning under structural restrictions: Algorithms and complexity. *AIJ*, 100(1–2):125–176, 1998.

[Keyder and Geffner, 2008] Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proc. ECAI 2008*, pages 588–592, 2008.

[Keyder *et al.*, 2010] Emil Keyder, Silvia Richter, and Malte Helmert. Sound and complete landmarks for and/or graphs. In *Proc. ECAI 2010*, pages 335–340, 2010.

[Likhachev *et al.*, 2008] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *AIJ*, 172(14):1613–1643, 2008.

[Richter and Helmert, 2009] Silvia Richter and Malte Helmert. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, pages 273–280, 2009.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:127–177, 2010.

[Richter *et al.*, 2008] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proc. AAAI 2008*, pages 975–982, 2008.

[Richter *et al.*, 2010] Silvia Richter, Jordan T. Thayer, and Wheeler Ruml. The joy of forgetting: Faster anytime search via restarting. In *Proc. ICAPS 2010*, pages 137–144, 2010.

[Vidal, 2004] Vincent Vidal. A lookahead strategy for heuristic search planning. In *Proc. ICAPS 2004* [2004], pages 150–159.