Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Landmarks Revisited

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

Malte Helmert and **Matthias Westphal**
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Freiburg, Germany
{helmert,westpham}@informatik.uni-freiburg.de

Abstract

Landmarks for propositional planning tasks are variable assignments that must occur at some point in every solution plan. We propose a novel approach for using landmarks in planning by deriving a pseudo-heuristic and combining it with other heuristics in a search framework. The incorporation of landmark information is shown to improve success rates and solution qualities of a heuristic planner. We furthermore show how additional landmarks and orderings can be found using the information present in multi-valued state variable representations of planning tasks. Compared to previously published approaches, our landmark extraction algorithm provides stronger guarantees of correctness for the generated landmark orderings, and our novel use of landmarks during search solves more planning tasks and delivers considerably better solutions.

Introduction

Landmarks for propositional planning were introduced by Porteous, Sebastia and Hoffmann (2001) and later studied in more depth by the same authors (Hoffmann, Porteous, and Sebastia 2003). According to their definition, *landmarks* are propositions that must be true at some point in every solution plan for a given planning task. For example, consider a Blocksworld task where the goal is to have block A stacked on block B. If some other block C is initially stacked on B, then C must be unstacked from B and B must be clear at some point for the goal to be achieved. Hence, clear(B) is a landmark for this problem instance. From the definition, goals are trivially landmarks, so on(A, B) is another landmark. We can also conclude an *ordering* on these two landmarks, denoted by clear$(B) \rightarrow$ on(A, B), indicating that block B must be clear *before* A can be stacked on it.

Hoffmann, Porteous and Sebastia propose an algorithm, called LM$^{\text{RPG}}$ in the following, that extracts landmarks and their orderings from the relaxed planning graph of a planning task. They use landmarks in a local search procedure, called LM$^{\text{local}}$ in the following, which searches iteratively for plans to the "nearest" landmarks, rather than searching for a plan to the goal. Their experiments demonstrate a substantial speed-up compared to an otherwise identical planner that does not use landmarks. However, they also note that the higher greediness of their search results in longer plans, and the iterative search for subgoals can fail on solvable tasks even if the underlying base planner is complete.

In this work, we propose a new way of using landmarks within a heuristic search planner which generally leads to shorter plans and an improved success rate compared to both the original LM$^{\text{local}}$ algorithm and state-of-the-art heuristics that do not exploit landmarks. Furthermore, we present an alternative algorithm for identifying landmarks and landmark orderings, in particular for a multi-valued state variable representation of planning tasks. Unlike LM$^{\text{RPG}}$, our algorithm only generates sound orderings.

The rest of this paper is organised as follows: first, we give some background on planning, in particular with multi-valued state variables, and landmarks. Next, we describe our algorithms for finding and using landmarks. We then evaluate our approach experimentally, followed by a discussion of the results and a conclusion.

Notation and Background

We consider planning in the SAS$^+$ planning formalism (Bäckström and Nebel 1995). A concise SAS$^+$ representation of a planning task can be generated from a typical PDDL representation automatically (Helmert 2008).

Definition 1 ***SAS$^+$ planning task***
*An **SAS$^+$ planning task** is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ where:*

- *$\mathcal{V}$ is a finite set of **state variables**, each with a finite domain $\mathcal{D}_v$. A **fact** is a pair $\langle v, d \rangle$ (also written $v \mapsto d$), where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$. A **partial variable assignment** s is a set of facts, each with a different variable. (We use set notation such as $\langle v, d \rangle \in s$ and function notation such as $s(v) = d$ interchangeably.) A **state** is a partial variable assignment defined on all variables $\mathcal{V}$.*
- *$\mathcal{O}$ is a set of **operators**, where an operator is a pair $\langle pre, eff \rangle$ of partial variable assignments.*
- *s_0 is a state called the **initial state**.*
- *$s_\star$ is a partial variable assignment called the **goal**.*

*An operator $o = \langle pre, eff \rangle \in \mathcal{O}$ is **applicable** in state s iff $pre \subseteq s$. In that case, it can be **applied** to s, which produces the state s' with $s'(v) = eff(v)$ where $eff(v)$ is defined and $s'(v) = s(v)$ otherwise. We write $s[o]$ for s'. For operator sequences $\pi = \langle o_1, \dots, o_n \rangle$, we write $s[\pi]$ for $s[o_1] \dots [o_n]$*

(only defined if each operator is applicable in the respective state). The operator sequence π is a ***plan*** *iff $s_\star \subseteq s_0[\pi]$.*

Each state variable of an SAS^+ planning task has an associated directed graph which captures the ways in which the value of the variable changes through operator application (Jonsson and Bäckström 1998).

Definition 2 ***domain transition graph***
The ***domain transition graph*** *of a state variable $v \in \mathcal{V}$ of an SAS^+ task $\langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ is the digraph $\langle \mathcal{D}_v, A \rangle$ which includes an arc $\langle d, d' \rangle$ iff $d \neq d'$ and there is an operator $\langle pre, eff \rangle \in \mathcal{O}$ with $pre(v) = d$ or $pre(v)$ undefined, and $eff(v) = d'$.*

Domain transition graphs have many uses, for example for computing heuristic goal distance estimates (Helmert 2004). We use them for deriving landmarks.

Definition 3 ***landmark***
Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be an SAS^+ planning task, let $\pi = \langle o_1, \dots, o_n \rangle$ be an operator sequence applicable in s_0, and let $i \in \{0, \dots, n\}$.

A fact F ***is true at time*** *i in π iff $F \in s_0[\langle o_1, \dots, o_i \rangle]$.*

A fact F ***is added at time*** *i in π iff F is true at time i in π, but not at time $i-1$. (Facts in s_0 are considered added at time 0.)*

A fact F ***is first added at time*** *i in π iff F is true at time i in π, but not at any time $j < i$.*

A fact F is a ***landmark*** *of Π iff in each plan for Π, it is true at some time.*

Note that facts in the initial state and facts in the goal are always landmarks by definition (consider $i = 0$ and $i = n$, respectively). In addition to knowing which landmarks a planning task has, it is also useful to know in which order they must be reached. The following definition follows the terminology of Hoffmann et al. (Porteous, Sebastia, and Hoffmann 2001; Hoffmann, Porteous, and Sebastia 2003).

Definition 4 ***orderings between facts***
Let A and B be facts of an SAS^+ planning task Π.

There is a ***natural ordering*** *between A and B, written $A \to B$, iff in each operator sequence where B is true at time i, A is true at some time $j < i$.*

There is a ***necessary ordering*** *between A and B, written $A \to_{\mathrm{n}} B$, iff in each operator sequence where B is added at time i, A is true at time $i-1$.*

There is a ***greedy-necessary ordering*** *between A and B, written $A \to_{\mathrm{gn}} B$, iff in each operator sequence where B is first added at time i, A is true at time $i-1$.*

Natural orderings are the most general; every necessary or greedy-necessary ordering is natural, but not vice versa. Similarly, every necessary ordering is greedy-necessary, but not vice versa. Hoffmann et al. introduce two further types of orderings, *reasonable* and *obedient reasonable*, which are less important to our contribution.

Hoffmann et al. show that deciding if a fact is a landmark and deciding orderings between facts are PSPACE-complete problems. Thus, practical methods for finding landmarks are incomplete (fail to find a given landmark or ordering) or unsound (falsely declare a fact to be a landmark, or determine a false ordering).

Previous Extraction of Landmarks and Orderings

The LM^{RPG} algorithm by Hoffmann et al. proceeds in three stages. First, potential landmarks and orderings are suggested by a fast *candidate generation procedure*. Second, a *filtering procedure* evaluates a sufficient condition for landmarks on each candidate fact, removing those which fail the test. (Note that unsound landmark orderings may remain.) Third, reasonable and obedient reasonable orderings between the landmarks are approximated.

We will briefly explain the first two stages, adapted to SAS^+ planning (LM^{RPG} is based on the STRIPS formalism). For a more detailed exposition, including the third stage, we refer to the original description of the algorithm (Hoffmann, Porteous, and Sebastia 2003).

In general, landmarks can be generated from a set of known landmarks (e. g., the facts in the goal) through backchaining. If a fact B is a landmark that is not already true in the initial state and all achievers of B (operators that have B as an effect) share a common precondition A, then A is a landmark, too. Unfortunately, with this strong condition only few landmarks are usually found (Hoffmann, Porteous, and Sebastia 2003). Instead, one may restrict attention to those operators which achieve B *for the first time*. However, it is PSPACE-hard to determine this set of operators exactly, so the candidate generation procedure of LM^{RPG} uses an approximation based on *relaxed planning graphs*. For SAS^+ tasks, these are defined as follows: for $i \in \mathbb{N}_0$, let

$$S_i := \begin{cases} s_0 & i = 0 \\ S_{i-1} \cup \bigcup_{\langle pre, eff \rangle \in \mathcal{O}_{i-1}} eff & i > 0 \end{cases}$$

$$O_i := \{\langle pre, eff \rangle \in \mathcal{O} \mid pre \subseteq S_i\} \setminus \bigcup_{j<i} O_j.$$

The set S_i is an over-approximation of all facts that can be reached by applying up to i operators under the usual simplifying assumptions for "relaxed plans" (Hoffmann and Nebel 2001). The set O_i contains precisely the operators which are applicable in the relaxation after i steps, but not previously. Landmark candidates are then suggested as follows: starting from a known landmark B which first appears in the relaxed planning graph in S_i ($i > 0$), consider all operators in O_{i-1} that achieve B. If these operators have a common precondition A, then A is a landmark candidate which is ordered *greedy-necessarily* before B.

In addition, LM^{RPG} finds further landmarks with a *one-step lookahead*: it may happen that the first achievers of a landmark B do not share a precondition, but that there is a fact A which is in turn needed for the preconditions of the first achievers. Let the set of operators $\{o_1, \dots, o_n\}$ be the set of first achievers of a landmark B, and let $X := \{L_1, \dots, L_n\}$ be facts s. t. $L_i \in pre_{o_i}$ – i. e., each L_i is part of the precondition for o_i. Then X is a *disjunctive* landmark in the sense that one of the facts in X needs to occur in every plan. While LM^{RPG} does not record such disjunctive landmarks, they are used as intermediaries for finding landmarks: if the union of all first achievers of the facts in X share a precondition A, then A is a landmark that must occur (at least) two steps before B. To avoid having to test an exponential number of such intermediaries, LM^{RPG} only considers sets X where all facts share the same predicate symbol in the original PDDL representation.

Due to the various approximations within the candidate generation procedure, there is no guarantee that the generated landmarks are sound. Therefore, the filtering procedure in the second stage applies a sufficient criterion to eliminate non-landmarks. Each fact A is tested by removing all achievers of A from the original task, and then checking whether the resulting task still has a relaxed solution. If not, then A is indeed critical to the solution of the original task, and is thus guaranteed to be a landmark. Otherwise, the candidate A is rejected. We remark that this pruning criterion guarantees that LM^{RPG} only generates true landmarks; however, landmark orderings are not pruned, and there are no guarantees of soundness for them. Indeed, LM^{RPG} may even generate landmarks with cyclic orderings (Porteous, Sebastia, and Hoffmann 2001).

In later work, Porteous and Cresswell (2002) propose a different approximation for the set of *first achievers* of B that considers more operators and guarantees the correctness of the found landmarks and orderings. Rather than building the relaxed planning graph using all operators and stopping when B first occurs, any operator is left out that would add B. 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* B in the planning task; we denote it by $pb(B)$ (for *possibly before*). Any operator that achieves B and is applicable given $pb(B)$ qualifies as being *possibly applicable before* B in the original task, and conversely, any operator that is indeed applicable before B in the original task will be contained in this approximation.

Zhu and Givan (2003) propose a technique for finding landmarks by propagating "necessary predecessor" information in a planning graph. Their approach is less closely related to ours than the work by Hoffmann et al., so we do not discuss it in detail.

Using Landmarks as Intermediary Goals

For exploiting landmarks during search, Hoffmann et al. propose the LM^{local} procedure which decomposes the planning task into smaller subtasks by making the landmarks intermediary goals. Instead of searching for the goal of the task, LM^{local} iteratively aims to achieve a landmark that is minimal with respect to the orderings.

In detail, LM^{local} first builds a landmark graph (with landmarks as vertices and orderings as arcs). Possible cycles are broken by removing some arcs. The sources S of the resulting directed acyclic graph are handed over to a base planner as a disjunctive goal, and a plan is generated to achieve one of the landmarks in S. This landmark, along with its incident arcs, is then removed from the landmark graph, and the process repeats from the end state of the generated plan. Once the landmark graph becomes empty, the base planner is asked to generate a plan to the original goal. (Note that even though all goal facts are landmarks and were thus achieved previously, they may have been violated again.)

As a base planner for solving the subtasks any planner can be used; Hoffmann et al. experimented with FF. They found that the decomposition into subtasks can lead to a more directed search, solving larger instances than plain FF in many domains. However, they also observed that solutions were often longer than those produced by plain FF, as it may happen that the disjunctive search control frequently switches between different parts of the task which may have destructive interactions. Sometimes this even leads to dead ends, so that LM^{local} fails on solvable tasks.

In an extension to this work, Sebastia, Onaindia, and Marzal (2006) employ a refined preprocessing technique that groups landmarks into consistent sets minimising the destructive interactions between the sets. Taking these sets as intermediary goals, they avoid the increased plan length, however, the preprocessing is computationally expensive and may take longer than solving the original problem.

Finding Landmarks for SAS^+ Planning

Our algorithm for finding landmarks is similar to LM^{RPG}, but differs in some ways. We adapted it to the SAS^+ setting, use the *possibly before* criterion to ensure that only sound orderings are found and applied some further extensions. As a result, we generally find more landmarks and orderings.

Instead of the one-step lookahead that LM^{RPG} performs to find further landmarks, we opt for the more general approach to admit disjunctive landmarks (Porteous and Cresswell 2002). Like LM^{RPG} we create disjunctive sets of facts from the preconditions of first achievers of a landmark B such that a set contains one precondition fact from each first achiever of B. Like LM^{RPG}, we require that all facts must stem from the same predicate symbol, and we also discard any sets of size greater than 4 in order to limit the number of possible sets. Each set A found this way is then recorded as a disjunctive landmark and ordered greedy-necessarily before B. If B is a disjunctive landmark, then the first achievers of B are all operators which achieve one of the facts in B.

An additional cheap and easy way of extracting more landmarks is offered by the SAS^+ representation using domain transition graphs (DTGs; see Def. 2). Given a *simple* (i. e., non-disjunctive) landmark $B = \{v \mapsto d'\}$ that is not part of the initial state s_0, 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 d that occurs on *every* path from $s_0(v)$ to d', then $A = \{v \mapsto d\}$ is a landmark, which can be naturally ordered before B. 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 d' are still connected – if not, A is a landmark. Note that all assignments to v which are not in $pb(B)$ can be removed prior to this test, as they can only occur *after* B and do not have to be tested.

After all landmarks have been generated, we can introduce some further natural orderings: for all landmarks A and B, if $B \notin pb(A)$, then B cannot be achieved without achieving A first, and hence we add the ordering $A \rightarrow B$.

As an optional post-processing step, we may also introduce *reasonable* and *obedient reasonable* orderings in the same way as LM^{RPG} (Hoffmann, Porteous, and Sebastia 2003). Note, however, that these are not always sound.

Using Landmarks as a Pseudo-Heuristic

Our aim is to incorporate the landmark information while searching for the original goal of the planning task. For this purpose, it is desirable to be able to smoothly integrate the landmark information with other useful heuristics.

The most straightforward way of using landmark information for search is to 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 the greedy-necessary predecessor 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 in best-first search.

Simply using pure landmarks counting, as outlined above, in best-first search already leads to good results in some cases. The results can furthermore be substantially improved by combining landmark counting with other heuristics, and by using *preferred operators*. Preferred operators (Helmert 2006) are operators that are believed to be useful for improving the heuristic value from a given state. For example, in the case of the FF heuristic, operators that can appear at the start of a relaxed plan from the state to the goal (*helpful actions*) are preferred. Exploiting preferred operators in heuristic search has been shown to improve results notably (Helmert 2006; Hoffmann and Nebel 2001). In our case, 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 simple acceptable landmark.

In the following, we describe in detail the search algorithm used for our experiments. As a framework, we use the Fast Downward (FD) planner (Helmert 2006). FD translates STRIPS tasks to SAS^+ and uses best-first search to solve them. It already contains the functionality to combine various heuristics and to use preferred operators. When configured to use more than one heuristic and no preferred operators, the FD planner manages several queues for state expansion, one for each heuristic. Any state that is evaluated during search is evaluated by all heuristics, and its successors are saved in each queue with the heuristic value computed by the heuristic of that queue. When retrieving the state to evaluate next, FD alternates between the queues, thus giving equal importance to all heuristics. If FD is configured to use preferred operators with one or more of the heuristics, it constructs an additional queue for each such heuristic. When a state is evaluated and expanded, those successor states that are reached via a preferred operator are put into the preferred operator queues, in addition to being put into the regular queues (for more details, see Helmert, 2006). States in the preferred operator queues thus are evaluated earlier on average. In addition, FD can be configured to give even more impact to preferred operators by using those queues more often than the regular queues.

Experiments

We evaluated our new techniques for generating and using landmarks on nearly all planning tasks from the international planning competitions 1998–2006, leaving out only the trivial Movie domain. In all experiments, the time and memory limits were 30 minutes and 3 GB respectively for each task, running on a 2.66 GHz Intel Xeon CPU.

Since the *generation method* for landmarks is orthogonal to their *usage* during search, and furthermore search algorithms using landmarks can be combined with different *base planners*, we can vary three independent dimensions. In order to keep the number of configurations manageable, we conducted two different experiments, with one of the three dimensions fixed in each of them.

In the first experiment, we evaluate our new method for using landmarks with three different base planners, keeping the landmark generation method fixed. Specifically, we use the original LM^{RPG} algorithm for generating landmarks. The three base planners are all based on greedy best-first search, each using a different heuristic, namely the FF heuristic, Causal Graph heuristic, and a "blind" heuristic assigning 1 to non-goal states and 0 to goal states. For each heuristic, we compare an algorithm using no landmark information (*base*) to the local search algorithm by Hoffmann et al. (*local*) and our new usage of landmarks as a pseudo-heuristic (*heur*). Preferred operators (Helmert 2006) were used in all applicable cases (i. e., whenever a non-blind heuristic was used), and reasonable and obedient reasonable orders were used in all configurations using landmarks.

Tab. 1 shows the percentage of tasks solved by each algorithm in each domain. With all three base planners, the landmarks pseudo-heuristic outperforms the other two alternatives (*base* and *local*). The results also show that the landmarks pseudo-heuristic can be beneficially combined with a base heuristic: when using the FF or Causal Graph heuristic the results are significantly better than with the blind heuristic. By contrast, the local landmarks search algorithm performs essentially identically with all three base planners. When used in conjunction with the FF heuristic, it is even worse than the base planner on average. This is mostly due to the incompleteness of the LM^{local} approach – it is prone to getting stuck in dead ends.

Overall, the best results are achieved when using the FF heuristic in the base planner. Here, the average difference of 1 percentage point between the landmarks pseudo-heuristic and the base planner (see last row of the table) may not seem big at first, but we note that in 10 of the 31 domains, using landmarks leads to more problems being solved than in the base planner, while the converse is only true in 3 domains. Over all domains, there are 25 tasks solved by *heur* but not *base* and 12 tasks solved by *base* but not *heur*. A detailed

Domain	FF heuristic base	FF heuristic local	FF heuristic heur	CG heuristic base	CG heuristic local	CG heuristic heur	blind heuristic base	blind heuristic local	blind heuristic heur
Airport (50)	**72**	32	64	46	32	**48**	34	32	**64**
Assembly (30)	100	97	100	10	**97**	83	0	**97**	7
Blocks (35)	100	100	100	100	100	100	43	100	100
Depot (22)	86	**100**	95	45	100	100	9	100	**95**
Driverlog (20)	100	100	100	100	100	100	25	100	100
Freecell (80)	95	80	**98**	89	80	**98**	16	80	**98**
Grid (5)	100	100	100	80	100	100	20	100	100
Gripper (20)	100	100	100	100	100	100	25	100	100
Logistics-1998 (35)	94	100	100	100	100	100	6	**100**	97
Logistics-2000 (28)	100	100	100	100	100	100	36	100	100
Miconic (150)	100	100	100	100	100	100	27	100	100
Miconic-FullADL (150)	91	91	90	89	**91**	90	41	**91**	42
Miconic-SimpleADL (150)	100	100	100	100	100	100	37	100	100
MPrime (35)	89	80	**97**	100	80	100	37	80	86
Mystery (30)	53	53	**57**	57	53	**60**	37	53	53
Openstacks (30)	100	100	100	70	100	100	23	100	100
OpticalTelegraphs (48)	4	**8**	4	2	**8**	6	2	8	**100**
Pathways (30)	93	**100**	97	23	100	100	13	100	100
Philosophers (48)	96	67	**100**	100	67	100	8	67	**73**
Pipesworld-NoTankage (50)	84	78	**88**	48	78	**84**	22	78	78
Pipesworld-Tankage (50)	78	58	**86**	28	58	**64**	12	58	**66**
PSR-Large (50)	64	**66**	64	64	**66**	62	20	**66**	64
PSR-Middle (50)	100	100	100	100	100	100	50	100	100
PSR-Small (50)	100	100	100	100	100	100	94	100	100
Rovers (40)	100	100	100	80	100	100	10	100	100
Satellite (36)	97	97	97	97	97	97	11	97	97
Schedule (150)	99	61	100	99	61	**100**	7	61	**94**
Storage (30)	63	63	60	**67**	63	63	40	**63**	57
TPP (30)	100	100	100	77	100	100	17	100	100
Trucks (30)	40	3	40	30	3	30	13	3	**23**
Zenotravel (20)	100	100	100	100	100	100	35	100	100
Averaged over domains	87	82	**88**	74	82	**87**	25	82	**84**

Table 1: Percentage of tasks solved using three different base planners (FF heuristic, Causal Graph heuristic, blind heuristic) and three different methods for using landmarks (base planner using no landmarks, Hoffmann et al.'s LM^{local} algorithm, landmark pseudo-heuristic). Bold results indicate better performance than the other two methods for a given base planner and domain. In all cases, Hoffmann et al.'s LM^{RPG} algorithm was used for generating landmarks and orderings. (Total number of tasks in each domain is shown in parentheses after the domain name in all tables.)

Domain	FF heuristic base	FF heuristic heur
Airport (50)	**6**	2
Depot (22)	0	**2**
Freecell (80)	1	**3**
Logistics-1998 (35)	0	**2**
Miconic-FullADL (150)	**2**	0
MPrime (35)	0	**3**
Mystery (30)	0	**1**
Pathways (30)	1	**2**
Philosophers (48)	0	**2**
Pipesworld-NoTankage (50)	0	**2**
Pipesworld-Tankage (50)	1	**5**
Schedule (150)	0	**1**
Storage (30)	**1**	0
Total	12	**25**

Table 2: Comparing the number of tasks solved exclusively by the FF-heuristic base planner and the landmark pseudo-heuristic approach, respectively. An entry of n for a given approach and domain means that the approach solved n tasks in this domain which the other approach did not solve. Domains where both approaches solved the same set of tasks are not shown.

	Hoffmann et al.		Zhu & Givan		New generation method	
Domain	LMs	Orderings	LMs	Orderings	LMs (Disj./DTG)	Orderings
Airport (50)	**42614**	294965	37156	73850	38203 (1014/7287)	**1459285**
Depot (22)	1420	4937	1240	2629	**1440** (159/179)	**6961**
Freecell (80)	**8448**	38809	7855	13700	7716 (0/2834)	**95330**
Gripper (20)	960	1400	960	1380	**1420** (460/460)	**2780**
Logistics-1998 (35)	2374	5261	2177	1965	**2909** (732/1230)	**8167**
Miconic-SimpleADL (150)	6583	8676	**10045**	**11469**	6583 (0/80)	10762
MPrime (35)	**199**	159	132	92	164 (44/51)	**198**
Rovers (40)	**2827**	1946	1565	785	2338 (379/2)	**2095**
Schedule (150)	8572	6508	7555	5491	**11530** (0/2958)	**9466**
Total	121056	433977	**153370**	345515	140630 (4977/19052)	**2104220**

Table 3: Numbers of landmarks and orderings produced by different generation methods. For the new generation method, numbers in parentheses indicate disjunctive landmarks and landmarks found by the domain transition graph criterion, respectively. Bold results indicate largest number of landmarks/orderings found in a given domain across the three approaches. Totals (last row) are across all domains from Tab. 1, not just the subset shown in this table.

Domain	base vs. local-HPS	base vs. heur-HPS	base vs. heur-ZG	base vs. heur-RHW
Airport (50)	**4**/0 (+6%)	0/**14** (−1%)	2/**14** (−2%)	0/**14** (−1%)
Assembly (30)	16/**23** (±0%)	13/**16** (±0%)	30/**43** (−1%)	0/0 (±0%)
Blocks (35)	**57**/37 (+22%)	34/**51** (−7%)	20/**48** (−3%)	20/**65** (−13%)
Depot (22)	40/**45** (−11%)	27/**45** (−5%)	22/**45** (−16%)	22/**54** (−17%)
Driverlog (20)	45/45 (−1%)	35/**60** (−4%)	30/**60** (−4%)	30/**60** (−5%)
Freecell (80)	**57**/6 (+12%)	**61**/23 (+5%)	**81**/7 (+20%)	**72**/10 (+12%)
Grid (5)	**40**/20 (+5%)	40/40 (+3%)	**60**/20 (+4%)	20/**60** (−7%)
Gripper (20)	**100**/0 (+7%)	**100**/0 (+4%)	0/0 (±0%)	0/**100** (−23%)
Logistics-1998 (35)	**71**/14 (+5%)	31/**37** (−1%)	**57**/20 (+2%)	14/**60** (−3%)
Logistics-2000 (28)	**96**/0 (+13%)	**46**/7 (+2%)	**60**/14 (+3%)	**32**/14 (+1%)
Miconic (150)	6/**80** (−8%)	0/**96** (−17%)	23/**56** (−2%)	0/**96** (−18%)
Miconic-FullADL (150)	23/**25** (±0%)	8/**30** (−1%)	**62**/20 (+6%)	**50**/28 (+4%)
Miconic-SimpleADL (150)	**96**/0 (+28%)	**58**/28 (+4%)	6/**80** (−9%)	**56**/29 (+3%)
MPrime (35)	14/**20** (+2%)	5/**22** (−6%)	5/**22** (−6%)	8/**28** (−6%)
Mystery (30)	3/**23** (−5%)	0/**26** (−9%)	0/**26** (−9%)	0/**26** (−12%)
Openstacks (30)	**86**/0 (+3%)	**86**/0 (+3%)	**76**/0 (+2%)	**76**/0 (+2%)
OpticalTelegraphs (48)	0/0 (±0%)	0/0 (±0%)	0/0 (±0%)	0/0 (±0%)
Pathways (30)	33/33 (+1%)	23/**40** (±0%)	26/**36** (±0%)	30/**33** (±0%)
Philosophers (48)	**60**/2 (+25%)	**25**/4 (+3%)	**22**/4 (+2%)	**25**/4 (+3%)
Pipesworld-NoTankage (50)	**36**/24 (+9%)	18/**40** (−3%)	22/**42** (−3%)	30/**42** (±0%)
Pipesworld-Tankage (50)	**24**/22 (+8%)	22/**40** (+4%)	**38**/32 (+20%)	22/**42** (±0%)
PSR-Large (50)	10/**24** (−5%)	8/**14** (−2%)	**14**/12 (−1%)	12/12 (+1%)
PSR-Middle (50)	2/**32** (−5%)	6/**18** (−2%)	12/**18** (±0%)	12/**14** (+3%)
PSR-Small (50)	**14**/0 (+4%)	**4**/0 (+1%)	0/0 (±0%)	**2**/0 (±0%)
Rovers (40)	22/**62** (−2%)	22/**50** (−2%)	22/**52** (−2%)	22/**55** (−2%)
Satellite (36)	25/**55** (−6%)	2/**63** (−10%)	11/**61** (−7%)	22/**50** (−6%)
Schedule (150)	16/**33** (−2%)	20/**66** (−8%)	32/**58** (−3%)	20/**66** (−8%)
Storage (30)	**46**/0 (+44%)	**16**/0 (+22%)	6/6 (+1%)	**26**/6 (+25%)
TPP (30)	**86**/0 (+32%)	13/**50** (−3%)	13/**50** (−3%)	16/**46** (−2%)
Trucks (30)	**3**/0 (+10%)	6/6 (±0%)	**13**/3 (+1%)	**13**/10 (+1%)
Zenotravel (20)	**60**/10 (+14%)	**40**/10 (+4%)	**40**/20 (+3%)	25/**45** (+0%)
Averaged over domains	**38**/20 (+6%)	25/**29** (−1%)	26/**28** (−1%)	22/**34** (−3%)

Table 4: Plan length comparison. Each result column compares the base planner to a configuration using landmarks. An entry like "**71**/14 (+5%)" indicates that the base planner found a shorter plan than the landmark configuration for 71% of the instances and a longer plan for 14% of the instances. (In the remaining cases, both produced plans of equal length.) The number in parentheses indicates that the plans generated by the landmark approach were 5% longer on average. The landmarks configurations use different methods for using landmarks (*local*: local landmarks search approach; *heur*: landmarks pseudo-heuristic) and different landmark generation methods (*HPS*: Hoffmann et al.; *ZG*: Zhu & Givan; *RHW*: new method from this paper). All configurations are based on best-first search with the FF heuristic and only compare on instances solved by both approaches.

comparison is shown in Tab. 2.

In this experiment, we did not vary the method of generating landmarks. We have also run our search algorithm *heur* with landmarks from alternative generation methods, namely the one proposed by Zhu and Givan (2003) and our new method for SAS^+ planning introduced in this paper. We do not report detailed results here, as the average results are very similar to the landmark generation method by Hoffmann et al. In particular, the same average coverage is achieved for all configurations using the FF heuristic as a base, while average results for other base planners vary by up to one percentage point.

However, the three approaches have slightly different strengths and weaknesses. For example, Tab. 1 (presenting results for the landmark generation method of Hoffmann et al.) shows that in the Airport domain and using the FF heuristic in the base planner, the *base* configuration (no landmarks) solves 72% of the tasks, while the *heur* configuration (landmarks pseudo-heuristic) only solves 64%. Hence, we do not seem to find useful landmarks in this domain. Using Zhu and Givan's landmarks, however, *heur* solves 80% of the Airport tasks, a considerable improvement over the baseline (see also Fig. 1). In the Philosophers domain, on the other hand, Tab. 1 shows that *heur* solves all tasks, while using Zhu and Givan's landmarks instead reduces the success rate to 75%.

Interestingly, the performance differences between the landmark generation approaches cannot be explained purely by the *number* of landmarks and/or orderings found. Tab. 3 shows the number of landmarks and orderings for the three approaches in some example domains, and summed up over all domains (including those not shown in the table). None of the generation approaches dominates the others consistently: for each approach, there is a domain where it finds more landmarks than the others. Zhu and Givan's procedure often finds fewer landmarks and orders than the other two; this is also the case in the Airport domain. At the same time, it leads to the best success rate of the three approaches in this domain. Our generation technique finds slightly more landmarks than LM^{RPG} on average. More notably, it finds many more orderings than both other approaches. While this does not affect the average number of tasks solved, it can make a difference in terms of *plan quality*, i. e., the length of the solution plans found.

To highlight this issue, Tab. 4 contains a comparison of plan lengths for various combinations of landmark generation and search procedures. In this second experiment, we always use the FF heuristic in the base planner, since this produced the highest success rate in the first experiment. Each column compares the base planner (without landmarks) to a different planner configuration that makes use of landmarks. As expected, the local search algorithm by Hoffmann et al. typically leads to significantly longer plans than the base planner. (For *local*, we only show results for one of the three landmark generation methods. The general observation holds for all three methods.) In 5 (10) domains, *local* increases the plan length for more than 80% (50%) of the tasks, compared to the base planner. In 5 domains, the increase is more than 20% on average. In Storage, it is a striking 44%.

In contrast, using the landmarks pseudo-heuristic reduces plan lengths compared to the base planner. This is true for all three landmark generation methods; however, the best result is achieved when using our landmarks generation procedure (*heur-RHW* in the table). Compared to the base planner, it decreases plan length by 3% on average, with its best domain being Gripper, where plan length is decreased in every task (by 23% on average), and its worst domain Storage, where plan length is increased in 26% of the tasks (by 25% on average). Gripper is an example where disjunctive landmarks are particularly helpful. Without them, all landmarks are of the form "at *ball1 roomb*" and "at *robot roomb*". This means that after picking up a ball in room a, the fastest way to achieve a new landmark is to move the robot to room b and drop the ball there. Such a landmark search results in plans where each ball is carried individually. With disjunctive landmarks, we have additional landmarks of the form "carry *ball1 right* $\vee$ carry *ball1 left*". This means that when the robot has picked up one ball, it can immediately achieve a new landmark by picking up another ball with its free gripper. As a result, our landmark generation method leads to optimal plans in the Gripper domain (see also Fig. 1).

Finally, some remarks on runtime. Computing landmarks is usually very inexpensive, as relaxed planning graphs can be built in linear time. For most tasks, landmark computation time is below one second. Therefore, overall runtime is dominated by search time for all but the simplest planning tasks. Using the landmarks pseudo-heuristic during search in addition to a base heuristic results in somewhat larger runtime per state expansion (because every state now needs to be evaluated by two heuristics). On small or medium-size problems, this overhead often translates into somewhat longer overall runtime of the landmarks approach.

As problems grow larger, however, the higher goal-directedness of the landmark search often pays off, as fewer states are evaluated compared to the base planner. Thus, for more difficult problems, the runtime of the *heur* approach is often lower than for the base planner. Averaged over all tasks solved by both approaches, the runtime of *heur* is at most 18% higher than that of the base planners. (The increase of 18% occurs when using the FF heuristic as base and the landmarks of Hoffmann et al.; using our landmarks instead results in an average *decrease* in runtime of 1%). Fig. 1 shows detailed results for some particular domains.

Conclusion

We showed how landmark information can be used in a heuristic search framework to increase the number of problem instances solved and improve the quality of the solutions. As opposed to the previously published landmark approach by Hoffmann, Porteous and Sebastia (2003), our algorithm cannot run into dead ends and we generally achieve better solutions. Our approach can easily be combined with other heuristic information, while the earlier approach appears not to benefit significantly from additional heuristics. As an example, we showed that two state-of-the-art heuristics, FF and Causal Graph, can both be significantly improved by integrating landmark information.

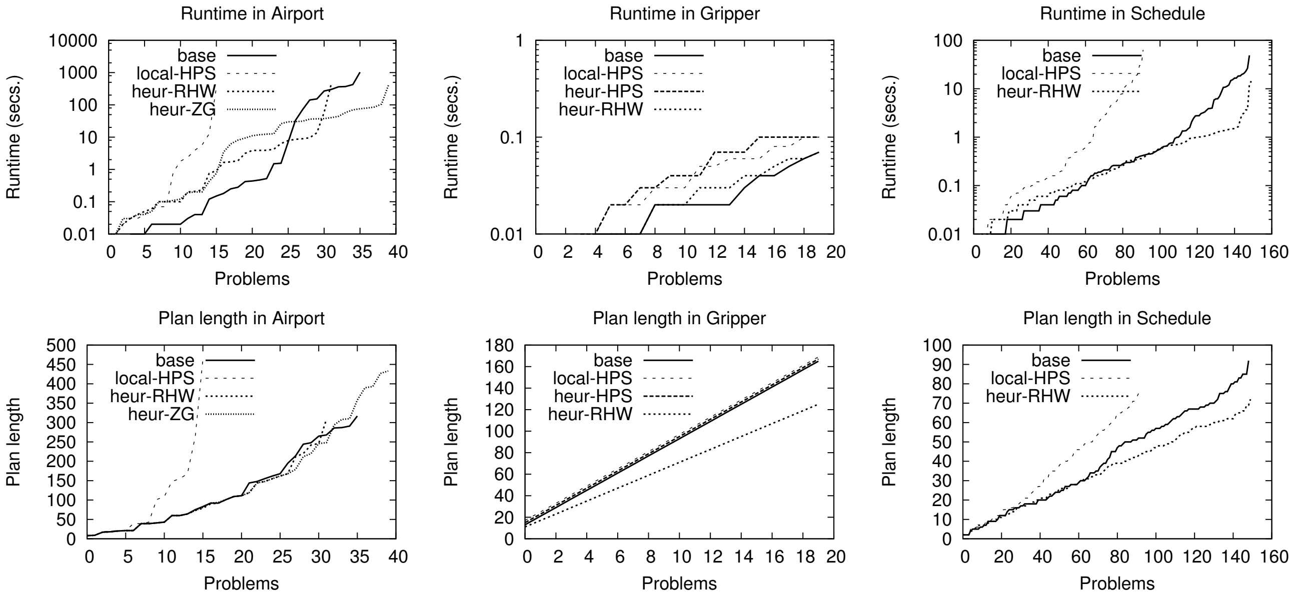

Figure 1: Runtimes and plan lengths of various landmark approaches in three particular planning domains (Airport, Gripper, Schedule). Base planner for all plots is best-first search with the FF heuristic. In the graphs at the top, a point at $(10, 0.02)$ indicates that 10 of the instances in the domain were solved by the respective approach in 0.02 seconds or less. Similarly, in the graphs at the bottom, a point at $(18, 101)$ indicates that for 18 of the instances in the domain, solutions of length at most 101 were found.

Acknowledgments

We thank Charles Gretton for helpful discussions.

NICTA is funded by the Australian Government, as represented by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council, through the ICT Centre of Excellence program.

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See http://www.avacs.org/ for more information.

References

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

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 161–170. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2008. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*. To appear.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2003. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Jonsson, P., and Bäckström, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence* 100(1–2):125–176.

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)*.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., and Borrajo, D., eds., *Pre-proceedings of the Sixth European Conference on Planning (ECP'01)*, 37–48.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2006. Decomposition of planning problems. *AI Communications* 19(1):49–81.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *Proceedings of the ICAPS'03 Doctoral Consortium*, 156–160.