Ordered Landmarks

Jörg Hoffmann HOFFMANN@INFORMATIK.UNI-FREIBURG.DE
Institut für Informatik,
Universität Freiburg,
Freiburg, Germany
Julie Porteous JULIE.PORTEOUS@CIS.STRATH.AC.UK
Department of Computer and Information Sciences,
The University of Strathclyde,
Glasgow, UK
Laura Sebastia LSTARIN@DSIC.UPV.ES
Dpto. Sist. Informáticos y Computación,
Universidad Politécnica de Valencia,
Valencia, Spain

Abstract

Many known planning tasks have inherent constraints concerning the best order in which to achieve the goals. A number of research efforts have been made to detect such constraints and to use them for guiding search, in the hope of speeding up the planning process.

We go beyond the previous approaches by considering ordering constraints not only over the (top-level) goals, but also over the sub-goals that will necessarily arise during planning. Landmarks are facts that must be true at some point in every valid solution plan. We extend Koehler and Hoffmann's definition of *reasonable* ordering constraints between top level goals to the more general case of landmarks. We show how landmarks can be found, how their reasonable ordering constraints can be approximated, and how this information can be used to decompose a given planning task into several smaller sub-tasks. Our methodology is completely domain- and planner-independent. The implementation demonstrates that the approach can yield significant runtime performance improvements when used as a control loop around state-of-the-art suboptimal planning systems, as exemplified by FF and LPG.

Technical Report Nr. 197, February 2004

1. Introduction

Given the inherent complexity of the general planning problem it is clearly important to develop good heuristic strategies for both managing and navigating the search space involved in solving a particular planning instance. One way in which search can be informed is by providing hints concerning the order in which planning goals should be addressed. This can make a significant difference to search efficiency by helping to focus the planner on a progressive path towards a solution. Work in this area includes that of Koehler and Hoffmann (2000). They introduce the notion of *reasonable orderings* which states that a pair of goals A and B can be ordered so that B is achieved before A if it isn't possible to reach a state in which A and B are both true, from a state in which just A is true, without

having to temporarily destroy A. In such a situation it is reasonable to achieve B before A to avoid unnecessary effort.

The main idea behind the work discussed in this paper is to extend those previous ideas on orderings by not only ordering the (top-level) goals, but also the sub-goals that will arise during planning, i.e., by also taking into account what we call the *landmarks*. The key feature of a landmark is that it *must* be true on any solution path to the given planning task. Consider the *Blocksworld* task shown in Figure 1, which will be our illustrative example throughout the paper.

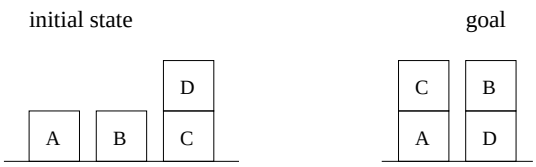

Figure 1: Example *Blocksworld* task.

For the reader who is weary of seeing toy examples like the one in Figure 1 in the literature, we remark that our techniques are *not* primarily motivated by this example. Our techniques are useful in much more complex situations. We use the depicted toy example only for easy demonstration of some of the important points. In the example, *clear(C)* is a landmark because it will need to be achieved in any solution plan. Immediately stacking B on D from the initial state will achieve one of the top level goals of the task but it will result in wasted effort if *clear(C)* is not achieved first. The ordering *clear(C)* $\leq$ *on(B D)* is, however, *not* reasonable in terms of Koehler and Hoffmann's definition. First, *clear(C)* is not a top level goal so it is not considered by Koehler and Hoffmann's techniques. Second, there are states where B is on D and from which *clear(C)* can be achieved without unstacking B from D again (compare the definition of reasonable orderings given above). But reaching such a state requires unstacking D from C, and thus achieving *clear(C)*, in the first place. This, together with the fact that *clear(C)* must be made true at some point, makes it sensible – if not reasonable, in Koehler and Hoffmann's formal sense – to impose the order *clear(C)* $\leq$ *on(B D)*.

We propose a natural extension of Koehler and Hoffmann's definitions to the more general case of landmarks (trivially, all top level goals are landmarks, too). We also revise parts of the original definition to better capture the intuitive meaning of a goal ordering. The extended and revised definitions capture, in particular, situations of the kind demonstrated with *clear(C)* $\leq$ *on(B D)* in the toy example above. We also introduce a new kind of ordering that often occurs between landmarks: A can be ordered before B if all valid solution plans make A true before they make B true. We call such ordering relations *necessary*. Typically, a fact *is* a landmark because it is necessarily ordered before some other landmark. For example, *clear(C)* is necessarily ordered before *holding(C)* in the above *Blocksworld* example.

We introduce sufficient criteria for the existence of reasonable orderings between landmarks. The criteria are based on necessary orderings, and inconsistencies between facts. We describe pre-processing techniques that extract landmarks, and that approximate necessary orderings between them. Using an inconsistency approximation technique from the

literature, we can then approximate reasonable orderings. After these pre-processes have terminated, what we get is a partial order between facts (landmarks). To be able to use this partial order for structuring the search for a plan, we remove all cycles (by removing edges/ordering relations that participate in a cycle) and end up with a tree structure, which we call the *landmark generation tree*.[1] This tree can be used to decompose the planning task into small chunks. We propose a method that does not depend on any particular planning framework: the landmarks provide a search control loop that can be used around any base planner that is capable of dealing with STRIPS input. The search control does not preserve optimality so there is not much point in using it around optimal planners such as Graphplan (Blum & Furst, 1997) and its relatives (because optimal planners are generally outperformed by suboptimal planners anyway). It does make sense, however, to use the control in order to further improve the runtime performance of suboptimal approaches to planning. To demonstrate this, we used the technique for control of two versions of FF (Hoffmann, 2000; Hoffmann & Nebel, 2001), and for control of LPG (Gerevini, Saetti, & Serina, 2003). We evaluated these planners across a range of 8 domains. We consistently obtain (sometimes dramatic) runtime improvements for the FF versions. We obtain runtime improvements for LPG in around half of the domains. The runtime improvement is, for all the planners, usually bought at the cost of slightly longer plans (though there are a few cases where the plans become shorter when using landmarks control).

The paper is organised as follows. Section 2 gives the basic notations. Section 3 defines what landmarks are, and in which ordering relations between them we are interested. The computational complexity of exact computation of the relevant pieces of information is investigated – all the decisions turn out to be PSPACE-complete – and we introduce sufficient criteria for the existence of reasonable orderings. Section 4 explains our implemented approximation techniques, and Section 5 explains how one can use landmarks to structure the search of an arbitrary base planner. Section 6 provides our empirical results in a range of domains. Section 7 closes the paper with a discussion of related work, of our contributions, and of future work. Most proofs are moved into Appendix A, and replaced in the text by proof sketches, to improve readability. Appendix B discusses some details regarding our experimental implementation of landmarks control around LPG.

2. Notations

We consider sequential planning in the propositional STRIPS (Fikes & Nilsson, 1971) framework. In the following, all sets are assumed to be finite. A *state* s is a set of logical facts (atoms). An *action* a is a triple $a = (pre(a), add(a), del(a))$ where $pre(a)$ are the action's *preconditions*, $add(a)$ is its *add list*, and $del(a)$ is its *delete list*, each a set of facts. The result of applying (the action sequence consisting of) a single action a to a state s is:

$$Result(s, \langle a \rangle) = \begin{cases} (s \cup add(a)) \setminus del(a) & pre(a) \subseteq s \\ \text{undefined} & \text{otherwise} \end{cases}$$

1. Of course, removing edges incident on cycles throws away ordering information that might be useful during search. Coming up with methods that can exploit the information contained in cycles is a topic for future work.

The result of applying a sequence of more than one action to a state is recursively defined as $Result(s, \langle a_1, \ldots, a_n \rangle) = Result(Result(s, \langle a_1, \ldots, a_{n-1} \rangle), \langle a_n \rangle)$. Applying an empty action sequence changes nothing, i.e., $Result(s, \langle\rangle) = s$. A planning *task* (A, I, G) is a triple where A is a set of actions, and I (the initial state) and G (the goals) are sets of facts (we use the word "task" rather than "problem" in order to avoid confusion with the complexity-theoretic notion of decision problems). A *plan*, or *solution*, for a task (A, I, G) is an action sequence $P \in A^*$ such that $G \subseteq Result(I, P)$.

3. Ordered Landmarks: What They Are

In this section we introduce our framework. We define what landmarks are, and in which ordering relations between them we are interested. We investigate the complexity of deciding about our theoretical concepts. All the decision problems turn out to be PSPACE-complete. We introduce sufficient criteria. These form the basis for our implemented approximation methods, which will be described in Section 4.

Section 3.1 defines our basic concepts, Section 3.2 focuses on reasonable orderings, and Section 3.3 focuses on *obedient* reasonable orderings (orderings that are reasonable if one has already committed to obey a given a-priori set of ordering constraints).

3.1 Basic Concepts

Landmarks are facts that must be true at some point during the execution of any solution plan.

Definition 1 *Given a planning task (A, I, G). A fact L is a* landmark *if for all $P = \langle a_1, \ldots, a_n \rangle \in A^*, G \subseteq Result(I, P) : L \in Result(I, \langle a_1, \ldots, a_i \rangle)$ for some $0 \leq i \leq n$.*

Note that in an unsolvable task *all* facts are landmarks (by universal quantification over the empty set of solution plans in the above definition). Our framework only makes sense if the task at hand is solvable.

Initial and goal facts are trivially landmarks: set i to 0 respectively n in Definition 1. In general, it is PSPACE-complete to decide whether a fact is a landmark or not.

Theorem 1 *Let* LANDMARK *denote the following problem: given a planning task (A, I, G), and a fact L; is L a landmark?*

Deciding LANDMARK *is PSPACE-complete.*

Proof Sketch: PSPACE-hardness follows by a straightforward reduction of the complement of PLANSAT– the decision problem of whether there exists a solution plan to a given arbitrary STRIPS task (Bylander, 1994) – to the problem of deciding LANDMARK. PSPACE-membership follows vice versa. ■

As said before, full proofs are in Appendix A. One of the most elementary ordering relations $L \leq L'$ between a pair L and L' of landmarks is this one: in any action sequence that makes L' true in some state, L must be true in the immediate preceding state. The reason for this is typically that L is a necessary prerequisite for achieving L' (we will

exploit this for our approximation techniques in Section 4). We call such ordering relations *necessary*.

Definition 2 *Given a planning task (A, I, G), and two facts L and L'. There is a* necessary ordering *between L and L', written $L \leq_n L'$, if $L' \notin I$, and for all $P = \langle a_1, \ldots, a_n \rangle \in A^*$: if $L' \in Result(I, \langle a_1, \ldots, a_i \rangle)$ then $L \in Result(I, \langle a_1, \ldots, a_{i-1} \rangle)$.*

The definition allows for arbitrary facts, but the case that we will be interested in is the case where L and L' are landmarks. Note that if $L' \in Result(I, \langle a_1, \ldots, a_i \rangle)$ then $i \geq 1$ as $L' \notin I$. The intention behind a necessary ordering $L \leq_n L'$ is that one must have L true before one can have L' true so it does not make sense to allow such orderings for initial facts L'. Deciding about necessary orderings in general is, again, PSPACE-complete.

Theorem 2 *Let* NECESSARY-ORD *denote the following problem: given a planning task (A, I, G), and two facts L and L'; does $L \leq_n L'$ hold?*

Deciding NECESSARY-ORD *is PSPACE-complete.*

Proof Sketch: PSPACE-hardness follows by reducing the complement of PLANSAT to NECESSARY-ORD. PSPACE-membership follows with a non-deterministic algorithm that guesses action sequences and checks if there is a counter example to the ordering. ■

Another interesting ordering relation are *greedy necessary orderings*, a slightly weaker version of the necessary orderings above. We postulate not that L is true prior to L' in all action sequences, but only in those action sequences where L' *is achieved for the first time.*

Definition 3 *Given a planning task (A, I, G), and two facts L and L'. There is a* greedy necessary ordering *between L and L', written $L \leq_{gn} L'$, if $L' \notin I$, and for all $P = \langle a_1, \ldots, a_n \rangle \in A^*$: if $L' \in Result(I, \langle a_1, \ldots, a_i \rangle)$ and $L' \notin Result(I, \langle a_1, \ldots, a_j \rangle)$ for $0 \leq j < i$, then $L \in Result(I, \langle a_1, \ldots, a_{i-1} \rangle)$.*

These ordering relations capture the fact that, really, what we are interested in is what happens when we directly achieve a fact from the initial state, rather than in some remote part of the state space. The consideration of these more remote parts of the problem, which are inherent in the definition of the non-greedy necessary orderings, can indeed make us lose useful information. A situation of this kind is contained, e.g., in our illustrative *Blocksworld* example (more below). Note that like above with the necessary orderings, the action sequence achieving L' must contain at least one step as $L' \notin I$. Deciding about greedy necessary orderings in general is PSPACE-complete.

Theorem 3 *Let* GREEDY-NECESSARY-ORD *denote the following problem: given a planning task (A, I, G), and two facts L and L'; does $L \leq_{gn} L'$ hold?*

Deciding GREEDY-NECESSARY-ORD *is PSPACE-complete.*

Proof Sketch: By a minor modification of the proof to Theorem 2. ■

We remark that $\leq_n$ is stronger than $\leq_{gn}$, that is, with $L \leq_n L'$ for two facts L and L', $L \leq_{gn} L'$ follows. In fact, $\leq_n$ is strictly stronger than $\leq_{gn}$. Consider the *Blocksworld* example in Figure 1. There is a greedy necessary ordering between *clear(D)* and *clear(C)*, *clear(D)* $\leq_{gn}$ *clear(C)*, but not a necessary ordering, *clear(D)* $\not\leq_n$ *clear(C)*. If we make *clear(C)* true the first time in an action sequence from the initial state, then the action achieving *clear(C)* will always be *unstack D C*, which requires *clear(D)* to be true. On the other hand, there can of course be action sequences which achieve *clear(C)* by different actions (*unstack A C*, for example). But, as we pointed out in the introduction, reaching a state where *clear(C)* can be achieved by such an action involves unstacking D from C, and thus achieving *clear(C)*, in the first place. In that sense, the definition of $\leq_n$ loses useful information here: we will see later that the ordering $\leq_{gn}$ can be used to derive an important (reasonable) ordering constraint.

3.2 Reasonable Orderings

Reasonable orderings were first introduced by Koehler and Hoffmann (2000), for top level goals. We extend their definition, in a slightly revised way, to landmarks. We then introduce a family of criteria sufficient for the existence of these orderings.

Let us first reiterate what the idea of reasonable orderings was originally. The idea introduced by Koehler and Hoffmann is this. If the planner is in a state s where one goal L' has just been achieved, but another goal L is still false, and L' must be destroyed in order to achieve L, then it might have been better to achieve L first: to get to a goal state from s, the planner will have to delete and re-achieve L'. If the same situation arises in *all* states s where L' has just been achieved but L is false, then it seems reasonable to generally introduce an ordering constraint $L \leq L'$, indicating that L should be achieved prior or simultaneously with L'.

The classical example for two facts with a reasonable ordering constraint are *on* relations in *Blocksworld*, where *on(B, C)* is reasonably ordered before *on(A, B)* whenever the goal is to have both facts true in the goal state: obviously, if one achieves *on(A, B)* first then one has to unstack A again in order to achieve *on(B, C)*.

3.2.1 Extension to Landmarks

Think about an unmodified application of Koehler and Hoffmann's definition to the case of landmarks. Consider a state s where we have a landmark L', but not another landmark L, and achieving L involves deleting L'. Does it matter? It might be that we do not need to achieve L anyway. It might also be that we do not need L' anymore once we have achieved L. In both cases, there is no need to delete and re-achieve L', and it does *not* appear reasonable to introduce the constraint $L \leq L'$. The question is, under which circumstances is it reasonable? The answer is given by the above two counter-examples. The situation matters if we

1. need to achieve L from s, and
2. must re-achieve L' again afterwards.

Both conditions are trivially fulfilled when L and L' are top level goals.

In the following, the way in which we deal with the above two conditions for the adequacy of reasonable orderings between landmarks, will become clear. Let us start with the formalities. We first formally fix the set of states where L' has been achieved before L.

Definition 4 *Given a planning task (A, I, G), and two facts L and L'. By $S_{(L', \neg L)}$, we denote the set of states s such that there exists $P = \langle a_1, \ldots, a_n \rangle \in A^*$, $s = Result(I, P)$, $L' \in add(a_n)$, and $L \notin Result(I, \langle a_1, \ldots, a_i \rangle)$ for $0 \leq i \leq n$.*

This definition differs from Koehler and Hoffmann's original definition in that it fixes a bug: in the original definition, it is only postulated that L is not true, $L \notin Result(I, \langle a_1, \ldots, a_n \rangle)$, instead of postulating that L has not been true on any point of the sequence leading to s. Definition 4 captures better the intuition that we want to consider all states where L' has been achieved *before* L – according to the original definition, it might be that L has been achieved already but been deleted again. Our revision of the original definition also provides the first (see above) condition for the adequacy of reasonable orderings between landmarks: if L is a landmark, and has not been true yet on the path to s, then any solution plan from s must achieve it per definition. The following definition formalises the second adequacy condition.

Definition 5 *Given a planning task (A, I, G), and two facts L and L'. L'* is in the aftermath of *L if, for all states $s \in S_{(L', \neg L)}$, and all solution plans $P = \langle a_1, \ldots, a_n \rangle \in A^*$ from s, $G \subseteq Result(s, P)$, there are $1 \leq i \leq j \leq n$ such that $L \in Result(s, \langle a_1, \ldots, a_i \rangle)$ and $L' \in Result(s, \langle a_1, \ldots, a_j \rangle)$.*

Note that, when L and L' are landmarks in this definition, all plans P from s will necessarily have L true at some point (as $s \in S_{(L', \neg L)}$). So the only requirement the definition makes is the point of truth of L'.

A top level goal L' is trivially in the aftermath of all other landmarks L (set j to n in Definition 5). Deciding about the aftermath relation in general is PSPACE-complete.

Theorem 4 *Let* AFTERMATH *denote the following problem: given a planning task (A, I, G), and two facts L and L'; is L' in the aftermath of L?*

Deciding AFTERMATH *is PSPACE-complete.*

Proof Sketch: PSPACE-hardness follows by reducing the complement of PLANSAT to AFTERMATH. PSPACE-membership follows by a non-deterministic algorithm that guesses counter examples. ∎

It is now straightforward to define what a reasonable ordering between a pair of landmarks L and L' is.

Definition 6 *Given a planning task (A, I, G), and two facts L and L'. There is a* reasonable ordering *between L and L', written $L \leq_r L'$, if and only if L' is in the aftermath of L, and*

$$\forall\, s \in S_{(L', \neg L)} : \forall\, P \in A^* :\ L \in Result(s, P) \Rightarrow \exists\, a \in P : L' \in del(a)$$

Once again, the corresponding decision problem is PSPACE-complete – even when not considering the inherent hardness of deciding about the aftermath relation.

Theorem 5 *Let* REASONABLE-ORD *denote the following problem: given a planning task* (A, I, G)*, and two facts* L *and* L' *such that* L' *is in the aftermath of* L*; does* $L \leq_r L'$ *hold?*

Deciding REASONABLE-ORD *is PSPACE-complete.*

Proof Sketch: PSPACE-hardness follows by reducing the complement of PLANSAT to REASONABLE-ORD, with the same construction as used by Koehler and Hoffmann (2000) for the original definition of reasonable orderings. PSPACE-membership follows by a non-deterministic algorithm that guesses counter examples. ■

For illustration of our definitions, reconsider the *Blocksworld* example in Figure 1, and the ordering *clear(C)* $\leq$ *on(B D)* that we mentioned, for motivation, in the introduction. The ordering is not reasonable in Koehler and Hoffmann's sense since (apart from the fact that *clear(C)* is not a top-level goal) there are reachable states where B is on D and C is not clear, but C can be made clear without unstacking B. However, reaching such a state requires unstacking D from C, and thereby achieving *clear(C)* in the first place, so these are not states where *on(B D)* has been achieved *before clear(C)*. These states are therefore not included in our definition of $S_{(on(BD),\neg clear(C))}$. Further, *on(B D)* is a top-level goal so it is in the aftermath of *clear(C)*. Hence we correctly obtain *clear(C)* $\leq_r$ *on(B D)* in the sense of Definition 6.

3.2.2 Sufficient Criteria

We have now introduced quite a number of PSPACE-complete decision problems. To make clear where all of this is going to, we now provide a family of criteria that are sufficient for reasonable orderings between landmarks. We remain impractical in the sense that we still assume we are given a set of landmarks, and a set of greedy necessary ordering constraints between them. Our sufficient criteria are based on this information, combining them with inconsistency between facts. So computing the "input" to the criteria is hard. The methods we use to approximate the required informations are the topic of Section 4.

We first identify a scenario in which greedy necessary orderings imply that some landmark L' is in the aftermath of some other landmark L.

Lemma 1 *Given a planning task* (A, I, G)*, and two landmarks* L *and* L'*. If either*

1. $L' \in G$*, or*
2. *there are landmarks* $L = L_1, \ldots, L_{n+1}$ *such that* $L_i \leq_{gn} L_{i+1}$ *for* $1 \leq i \leq n$*, and* $L' \leq_{gn} L_{n+1}$*,*

then L' *is in the aftermath of* L*.*

Proof Sketch: If $L' \in G$ then L' is trivially in the aftermath of L. Otherwise, under the given circumstances, L' and L_n must be true together at some point in any action sequence

achieving the goal from a state in $S_{(L',\neg L)}$. As L is transitively ordered before L_n, it has to be true prior to (or simultaneously with, if $n = 1$) L'. ■

Now we combine the above with *interference* information. We state several ways in which achieving a fact L leads to deletion of a fact L'. We need the notion of *inconsistency*: two facts are inconsistent if there is no reachable state that contains both of them.[2] Interference is defined as follows.

Definition 7 *Given a planning task (A, I, G), and two facts L and L'. L* interferes *with L' if one of the following conditions holds:*

1. *L and L' are inconsistent;*
2. *there is a fact $x \in \bigcap_{a \in A, L \in add(a)} add(a)$ such that x is inconsistent with L';*
3. *$L' \in \bigcap_{a \in A, L \in add(a)} del(a)$;*
4. *or there is a landmark x inconsistent with L' such that $x \leq_{gn} L$.*

Interference together with the aftermath relation implies reasonable orderings between landmarks.

Theorem 6 *Given a planning task (A, I, G), and two landmarks L and L'. If L interferes with L', and either*

1. *$L' \in G$, or*
2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq_{gn} L_{i+1}$ for $1 \leq i \leq n$, and $L' \leq_{gn} L_{n+1}$,*

then there is a reasonable ordering between L and L', $L \leq_r L'$.

Proof Sketch: By Lemma 1, L' is in the aftermath of L. Let us look at the four possible reasons for interference. If L is inconsistent with L' then obviously achieving L involves deleting L'. If all actions that achieve L add a fact that is inconsistent with L', the same argument applies. The case where all actions that achieve L delete L' is obvious. As for the last case, say we are in a state $s \in S_{(L',\neg L)}$. Then x is not in s (because L' is). Due to $x \leq_{gn} L$, x must be achieved prior to L, and thus L' will be deleted. ■

Theorem 6 gives us four sufficient criteria for reasonable orderings between landmarks: if we have two landmarks L and L' that fulfill the scenario stated in the prerequisite to Lemma 1, and L interferes with L', then $L \leq_r L'$ holds. In Section 4 we will describe how we use this in practice. Now, we push our notions one step further.

2. Deciding about inconsistency is obviously PSPACE-hard: just imagine a task where we insert one of the facts into the initial state, and the other fact such that it can only be made true once the original goal has been achieved. We approximate inconsistency with a sufficient technique developed by Maria Fox and Derek Long (1998) (see Section 4).

3.3 Obedient Reasonable Orderings

Say we already have a set O of ordering constraints $L \leq L'$. Practically we will obtain O by approximating the sufficient criteria given above. Given that we want to restrict search in that we postulate all these ordering constraints to be obeyed, there is the question, are more reasonable ordering constraints implied by the orderings in O? We answer the question in this section by introducing a weaker form of reasonable orderings, which are obedient in the sense that they only arise if one commits to a given set O of ordering constraints.

To illustrate what is to come, consider the following situation. Say we got landmarks L and L', and a state $s \in S_{(L',\neg L)}$. Say further that we must delete L' in order to achieve L. Also, there is a third landmark L_{n+1} such that $L' \leq_{gn} L_{n+1}$ and $L \leq_r L_{n+1}$. Now, if the ordering $L \leq L_{n+1}$ was greedy necessary, $L \leq_{gn} L_{n+1}$, then we would have the reasonable ordering $L \leq_r L'$ by Theorem 6: L and L' would need to be true together immediately prior to the achievement of L_{n+1}, so L' would be in the aftermath of L. However, the ordering constraint $L \leq L_{n+1}$ is "only" reasonable so there is no guarantee that any solution plan from s will adhere to it – a plan can choose to achieve L after L_{n+1} in which case there would not necessarily be a deletion and re-achievement of L'. But if we want to enforce the ordering constraint $L \leq_r L_{n+1}$, disallowing plans that do not obey it, then achieving L' before L leads to deletion and re-achievement of L' and is thus not reasonable. The remainder of this section formalises this idea. The key notion is that of an *obedient action sequence*.

Definition 8 *Given a planning task (A, I, G), a set O of ordering constraints, and a state s. An action sequence $\langle a_1, \ldots, a_n \rangle$ is* obedient in s *if, for all constraints $L \leq L'$ in O, it holds that*

$$min\{i \mid L \in Result(s, \langle a_1, \ldots, a_i \rangle)\} \leq min\{i \mid L' \in Result(s, \langle a_1, \ldots, a_i \rangle)\}$$

where the minimum over an empty set is ∞.

In words, Definition 8 says that, when $L \leq L'$ is a constraint and s is a state, an action sequence obedient in s is one whose execution from the state either does not achieve L' at all, or achieves L at least as early as L'. The rest of our definitions carry over in a straightforward way. Basically, now we are only interested in sets of obedient action sequences rather than in sets of all of them.

3.3.1 Adapted Definitions

Altogether, our framework for reasonable orderings includes the five definitions of necessary orderings, greedy necessary orderings, the states $S_{(L',\neg L)}$, the aftermath relation, and the reasonable orderings themselves. In all these definitions, there is talk about the set of all action sequences, that is, the set of all action sequences that fulfill a certain property (like achieving a fact L'). So all these definitions could be adapted to the obedient framework by additionally requiring that the action sequences in question are obedient (with respect to a given set of ordering constraints). Practically, we use the idea only in the context of the states $S_{(L',\neg L)}$, the aftermath relation, and the resulting reasonable orderings. So we keep the first two definitions unchanged. It is an open question how our approximation techniques

for these two definitions could be adapted to the case of obedient action sequences. The other definitions translate as follows.

Definition 9 *Given a planning task (A, I, G), a set O of ordering constraints, and two facts L and L'.*

1. *By $S^O_{(L', \neg L)}$, we denote the set of states s such that there exists an action sequence $P = \langle a_1, \ldots, a_n \rangle \in A^*$ that is obedient in I, with $s = Result(I, P)$, $L' \in add(a_n)$, and $L \notin Result(I, \langle a_1, \ldots, a_i \rangle)$ for $0 \leq i \leq n$.*

2. *L'* is in the obedient aftermath of *L if, for all states $s \in S^O_{(L', \neg L)}$, and all solution plans $P = \langle a_1, \ldots, a_n \rangle \in A^*$ that are obedient in s, $G \subseteq Result(s, P)$, there are $0 \leq i \leq j \leq n$ such that $L \in Result(s, \langle a_1, \ldots, a_i \rangle)$ and $L' \in Result(s, \langle a_1, \ldots, a_j \rangle)$.*

3. *There is an* obedient reasonable ordering *between L and L', written $L \leq^O_r L'$, if and only if L' is in the obedient aftermath of L, and*

$$\forall\, s \in S^O_{(L', \neg L)} : \forall\, P \in A^* :\ L \in Result(s, P) \Rightarrow \exists\, a \in P : L' \in del(a)$$

We remark that the modified definitions do not change the computational complexity of the corresponding decision problems.[3] As a quick illustration of the new definitions, reconsider the situation described above: there, L' is not in the aftermath of L, but in the obedient aftermath of L because all action sequences that obey the constraint $L \leq_r L_{n+1}$ make L' true at a point simultaneously or behind L (namely immediately prior to L_{n+1}, assuming that there is no action that adds both L and L_{n+1}). As L' must be deleted in order to achieve L, we obtain the ordering $L \leq^O_r L'$. That is, if we commit to the constraint $L \leq_r L_{n+1}$ then it is reasonable to also order L before L'.

3.3.2 Adapted Criteria

We now adapt the sufficient criteria from Section 3.2.2. The only difference lies in the sufficient criterion for the, now obedient, aftermath relation. Instead of the greedy necessary orderings that we used before, we now refer to the ordering constraints in O.

Lemma 2 *Given a planning task (A, I, G), a set O of ordering constraints, and two landmarks L and L'. If either*

1. *$L' \in G$, or*

2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq L_{i+1}$ is in O for $1 \leq i \leq n$, $L' \leq_{gn} L_{n+1}$, and for at least one $1 \leq i \leq n$ there is no action $a \in A$ with $L_i \in add(a)$ and $L_{i+1} \in add(a)$,*

then L' is in the obedient aftermath of L.

3. For the obedient aftermath relation, minor modifications of the proof to Theorem 4 suffice: PSPACE-hardness follows by using the empty set of ordering constraints; PSPACE-membership follows by extending the non-deterministic decision algorithm with flags that check if the ordering constraints are adhered to. Similarly for obedient reasonable orderings and the proof to Theorem 5.

Proof Sketch: If $L' \in G$, then L' is, trivially, in the aftermath of L. Otherwise, assume a state $s \in S_{(L', \neg L)}$. Under the given circumstances any action sequence achieving the goal from s achieves $L = L_1$ at a point prior to or simultaneously with L' because: L' has to be true directly before L_{n+1}; and $L_1, \ldots, L_n$ have to be achieved in a sequence preceding L_{n+1}, where not all of them can be achieved in a single step. ■

Interference together with the obedient aftermath relation implies obedient reasonable orderings between landmarks.

Theorem 7 *Given a planning task (A, I, G), a set O of ordering constraints, and two landmarks L and L'. If L interferes with L', and either*

1. *$L' \in G$, or*
2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq L_{i+1}$ is in O for $1 \leq i \leq n$, $L' \leq_{gn} L_{n+1}$, and for at least one $1 \leq i \leq n$ there is no action $a \in A$ with $L_i \in add(a)$ and $L_{i+1} \in add(a)$,*

then there is an obedient reasonable ordering between L and L', $L \leq_r^O L'$.

Proof Sketch: By Lemma 2 and the same arguments as in the proof to Theorem 6. ■

4. How to Find Ordered Landmarks

We now describe our implemented methods to find landmarks in a given planning task, and to approximate their inherent ordering constraints. We use the following terminology. A set of landmarks together with a set of orderings forms a directed graph structure in the obvious way (there is an edge from L to L' iff L is ordered before L'). We refer to this graph as the *landmark-generation tree* (LGT).[4] Certain sets of facts in the LGT form *AND-nodes*. Facts in an AND-node must be true *together* at some point during plan execution. Such facts are either the (top-level) goals, or sets of landmarks that all have a necessary ordering before the same other landmark. Note that AND-nodes in this sense are reflected in our sufficient criteria for the aftermath relation, Lemmata 1 and 2: basically, to make sure that a landmark L' is in the aftermath of a landmark L, these require that L is transitively ordered before a landmark L_n that is in an AND-node together with L'. (The case where L' is a top-level goal can be understood this way in terms of an implicitly given, possibly different, goal L_n that L is ordered before – all landmarks in the LGT are ordered before some goal.)

Sections 4.1 and 4.2 describe our approximations of the definitions provided in Section 3: Section 4.1 explains how we find landmarks in a given planning task, Section 4.2 explains how we approximate necessary and (obedient) reasonable orderings between these landmarks. Section 4.3 then describes a preliminary form of "lookahead" landmarks/orderings that we have also implemented and used.

4. During our pre-process building this structure, the "tree" can contain cycles; but, as was mentioned in the introduction, we remove such cycles before the actual planning process – the search for a plan – begins.

4.1 Finding Landmarks

By Theorem 1, it is PSPACE-complete to decide whether an arbitrary fact is a landmark or not. Our approximation technique proceeds in two steps: first, a backward chaining process extracts landmark candidates; then, evaluating a simple sufficient condition eliminates those candidates that are not provably landmarks (according to the sufficient condition, that is).

4.1.1 Extracting Landmark Candidates

Candidate landmarks are extracted using the *relaxed planning graph* (RPG) (Blum & Furst, 1997; Hoffmann & Nebel, 2001): relax the planning task by ignoring all delete lists, then build Graphplan's planning graph, chaining forward from the initial state of the task to a graph level where all goals are reached. Because the delete lists are empty, the graph does not contain any mutex relations. Once the RPG has been built, a straightforward backward chaining process extracts our landmark candidates and some first ordering relations between them – namely, our approximation of the greedy necessary orderings (more on these below in Section 4.2.1).

The extraction process works as follows. First, all top level goals are added to the LGT and are posted as goals in the first level where they appeared in the RPG. Then, each goal is solved in the RPG starting from the last level. For each goal g in a level, all actions achieving g are grouped into a set and the intersection I of their preconditions is computed. For all facts p in I we: post p as a (sub-)goal in the first RPG level were it is achieved; insert p as a node into the LGT; insert an edge between p and g into the LGT. When all goals in a level are achieved, we move on to the next lower level. The process stops when the first (initial) level is reached.

L_0	A_1	L_1	A_2	L_2	A_3	L_3
on-table A	pick-up A	holding A	stack B A	on B A	stack C A	on C A
on-table B	pick-up B	holding B	stack B D	on B D	stack C B	on C B
on-table C	unstack D C	holding D	stack B C	on B C	stack C D	on C D
on D C		clear C	put-down B	...	...	...
clear A			...			
clear B			pick-up C	holding C		
clear D			...	...		
arm-empty						

Figure 2: Summarised RPG for the illustrative *Blocksworld* example from Figure 1.

Let us illustrate the extraction process with the *Blocksworld* example from Figure 1. The RPG corresponding to this task is shown in Figure 2. As we explained above, the extraction process starts by adding two nodes representing the goals *on(C A)* and *on(B D)* to the LGT (formally, node set {*on(C A)*,*on(B D)*} and empty edge set). It also posts *on(C A)* as goal in level 3 and *on(B D)* in level 2. There is only one action achieving *on(C A)* in level 3: *stack C A*. So, *holding(C)* and *clear(A)* are new candidates. *holding(C)* is posted as a goal in level 2, *clear(A)* is initially true and does therefore not need to be posted as a goal. The new LGT is: ({*on(C A)*,*on(B D)*,*holding(C)*,*clear(A)*}, {(*holding C*),*on(C A)*), (*clear A*),*on(C A)*)}). As there are no more goals in level 3, we move downwards to solve the goals in level 2. We now have two goals: *on(B D)* and *holding(C)*. In both cases,

there is only one action adding each fact (*stack B D* and *pick-up C*), so their preconditions *holding(B)*, *clear(D)*, *clear(C)*, *on-table(C)*, and *arm-empty()*, as well as the respective edges, are included into the LGT. The goals at level 1 are *holding(B)* and *clear(C)*, which are added by the single actions *pick-up B* and *unstack D C*. The process ends up with the LGT as shown in Figure 3. Facts in AND-nodes are grouped together in boxes.[5]

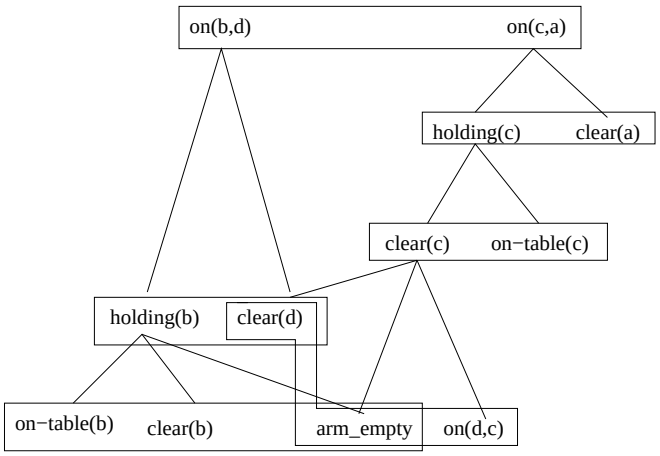

Figure 3: LGT for the illustrative *Blocksworld* task.

4.1.2 Verifying Landmark Candidates

If the LGT contains facts that are not really landmarks, then this can cause incompleteness in our search framework (as we will describe in Section 5). We therefore want to make sure that the LGT contains only landmarks. We do so using the following simple observation.

Proposition 1 *Given a planning task (A, I, G), and a fact L. Define a modified action set A_L as follows.*

$$A_L := \{(pre(a), add(a), \emptyset) \mid (pre(a), add(a), del(a)) \in A, L \notin add(a)\}$$

If (A_L, I, G) is unsolvable, then L is a landmark in (A, I, G).

Deciding about solvability of planning tasks with empty delete lists can be done in polynomial time by building the RPG: the task is unsolvable iff the goals can't be reached (Blum & Furst, 1997; Hoffmann & Nebel, 2001). So our landmark verification process looks at all landmark candidates in turn; a candidate and its incident edges are removed iff the candidate can not, by Proposition 1, be proved to be a landmark.

As an example, say we want to move from city A to city D on the road map shown in Figure 4, using a standard *move* operator.

Landmarks extraction will come up with the following LGT: $\{at(A), at(E), at(D)\}, \{(at(A), at(E)), (at(E), at(D))\}$ – the RPG is only built until the goals are reached

5. Note that the same fact can participate in different AND-nodes. This does not imply that we can merge these AND-nodes: the respective facts must be true together at possibly *different* points during plan execution.

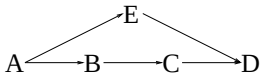

Figure 4: An example road map.

the first time, which happens in this example before *move C D* comes in. However, the action sequence $\langle$*move(A,B), move(B,C), move(C,D)*$\rangle$ achieves the goals without making *at(E)* true. Therefore, *at(E)* is not really a landmark. Our verification process detects this: in the RPG, when ignoring all actions that achieve *at(E)*, the goal is still reachable by the actions that *move* to *D* via *B* and *C*. So *at(E)* and its edges are removed, yielding the final (trivial) LGT with node set {*at(A)*, *at(D)*} and empty edge set.

4.2 Ordering Landmarks

By Theorems 3 and 5, the ordering relations we are interested in (greedy necessary orderings and reasonable orderings) are hard to compute. Our approximation technique proceeds in the following series of steps.

1. In the LGT computed as described above, consider the edges as the approximation of greedy necessary orderings.
2. Use these orderings, and an approximation of inconsistency, to identify new orderings by Theorem 6 (these orderings are reasonable if the orderings approximated in step 1 are really greedy necessary). Include these orderings into the LGT.
3. Based on the LGT and Theorem 7, identify new obedient reasonable orderings, and include them into the LGT.
4. Remove, by greedily removing edges, any cycles that the LGT might contain.

In this section, we shall go through each of these steps in turn.

4.2.1 Approximating Necessary Orderings

Recall Definitions 2 and 3: there is a necessary ordering between L and L' if all action sequences make L true one step before they make L' true. The ordering is *greedy* necessary if that property holds when L' is made true the first time (on the action sequence).

As said, our approximated greedy necessary orderings, written $\leq^a_{gn}$, are simply the edges in the LGT constructed to find landmarks. To see why this makes sense, consider the following. Remember that the LGT is constructed by intersecting preconditions in a relaxed planning graph. Now, if L is in the intersection of *all* achievers of L', then $L \leq_n L'$, and hence $L \leq_{gn} L'$, follow trivially. So we can easily obtain a safe (sufficient) approximation of necessary orderings. However, we are interested in approximating the weaker *greedy* necessary orderings. We did not find a safe approximation of these orderings. Our implemented technique is, however, intuitively justified: intersecting the preconditions of only those of L''s achievers that appear *below* L' in the relaxed planning graph, we consider L''s earliest achievers only. As an illustration, consider the facts *clear(D)* and *clear(C)* in our running example. As said earlier (in Section 3.1), there is a greedy necessary ordering

clear(D) $\leq_{gn}$ *clear(C)* between these facts, but *not* a necessary ordering. And indeed, while there are actions that achieve *clear(C)* but do not require *clear(D)* in their precondition (e.g., *unstack A C*), none of these actions is contained in an RPG level below *clear(C)* – the earliest action achieving *clear(C)* is *unstack D C*. This is also the only action that can achieve *clear(C)* for the first time on any action sequence.

4.2.2 Approximating Reasonable Orderings

As we have detailed in Section 3.2, we say that there is a reasonable ordering $L \leq_r L'$ between two landmarks if:

1. L' is in the aftermath of L – L' must be true at some point after or simultaneously with L; and
2. from a state where L' was reached strictly before L, L can not be reached without deleting L' again.

Under these circumstances, ordering $L \leq L'$ is reasonable ($L \leq_r L'$) in the sense that the planner has to destroy and re-establish L' if it is achieved strictly before L. Lemma 1 provides a sufficient criterion for point 1, Definition 7 – fact interference – provides a sufficient criterion for point 2. Theorem 6 combines these criteria into a sufficient criterion for the existence of a reasonable ordering between a pair of landmarks. The idea underlying our pre-process is to introduce an ordering $L \leq_r L'$ for all pairs of landmarks that satisfy the theorem's prerequisites.

The inputs to Theorem 6 are information about greedy necessary orderings between pairs of facts, and information about inconsistency between pairs of facts. Both inputs are hard to compute. Our approximation to the former piece of information are the $\leq^a_{gn}$ orderings extracted as described above. Our approximation to the latter piece of information is a technique from the literature (Fox & Long, 1998), the TIM API. This provides a function *TIMinconsistent(A,B)* that, for facts A and B, returns TRUE only if A and B are inconsistent. The function is incomplete, i.e., it can return FALSE even if A and B are inconsistent.

Given the required inputs to Theorem 6, implementing the desired pre-process is straightforward. We need to identify pairs of facts that satisfy the sufficient criterion for the aftermath relation, and that interfere. The only question to be resolved is how to efficiently access the pairs of facts that satisfy the aftermath criterion. The criterion demands, for landmarks L and L', that L is transitively ordered (with greedy necessary orderings) before some third landmark L_n that participates in an AND-node with L' (the transitive order is denoted $L = L_1 \leq_{gn}, \ldots, \leq_{gn} L_n$). Our implementation generates all such pairs L and L' in the LGT simply by considering all AND-nodes one-by-one. Each such pair of facts is then tested for interference by evaluating Definition 7 (using the TIM API to approximate fact inconsistency), and an approximated reasonable ordering, written $L \leq^a_r L'$, is introduced if the test succeeds. (The $\leq^a_r$ orderings are not guaranteed to be reasonable as the $\leq^a_{gn}$ orderings are not guaranteed to be greedy necessary.)

Evaluating Definition 7 is conceptually trivial when the inconsistency information is given. Let us have a closer look at the process that generates all pairs L and L' in the LGT

that satisfy the aftermath criterion. As a first illustration, consider L = *clear(C)* and L' = *on(B D)* in Figure 3. *clear(C)* is, via *holding(C)*, transitively ordered before *on(C A)*, which participates in an AND-node (the root node corresponding to the top-level goals) with *on(B D)*. Therefore, *on(B D)* is in the aftermath of *clear(C)* by Lemma 1 (assuming the greedy necessary orderings in the LGT are correct). Observe that, by the same argument, *on(B D)* is in the aftermath of *any* fact that is a descendant of *on(C A)* in the LGT. Our process is thus the following construction of **for**-loops. For all AND-nodes in the LGT, for all landmarks L' in the current AND-node, for all landmarks $L_n \neq L'$ in the AND-node,[6] and for all L that are descendants of L_n in the LGT, generate the pair L and L' (and test it for interference). Stop when all AND-nodes and all landmarks in them have been processed.

In the illustrative *Blocksworld* example, the order *clear(C)* $\leq_r$ *on(B D)*, our motivating example from the introduction, is found as follows. Have a look at Figure 3. Say the above described process has just started with L' = *on(B D)* in the root AND-node. The only candidate for L_n at this AND-node is *on(C A)*. The descendants of *on(C A)* are *holding(C)*, *clear(A)*, *clear(C)*, *on-table(C)*, *clear(D)*, *arm-empty*, and *on(D C)*. Thus *on(B D)* is in the aftermath of all these facts. Of the facts, (only) *clear(D)* and *clear(C)* interfere with *on(B D)*, so the approximated reasonable orderings are *clear(D)* $\leq_r^a$ *on(B D)* and *clear(C)* $\leq_r^a$ *on(B D)*. *clear(C)* interferes with *on(B D)* because of condition 4 in Definition 7: *clear(D)* is inconsistent with *on(B D)*, and is ordered *clear(D)* $\leq_{gn}^a$ *clear(C)*.

4.2.3 Approximating Obedient Reasonable Orderings

The idea behind obedient reasonable orderings, as explained in Section 3.3, is that, if we commit ourselves to obey a set of ordering constraints O, other reasonable orderings might follow. The set of ordering constraints we commit to are the $\leq_{gn}^a$ and $\leq_r^a$ orderings extracted by the processes described previously. The definition of the aftermath relation is weakened to the definition of the *obedient* aftermath (Definition 9), where we are only interested in action sequences that obey the constraints in O. In the sufficient criterion (Lemma 2) that forms the basis for our implemented approximation technique, what changes is the definition of the AND-nodes in the LGT (more details below). We then extract obedient reasonable orderings exactly as before: for all pairs of facts L and L' in the LGT that satisfy the sufficient criteria for the obedient aftermath relation and for fact interference – the prerequisites of Theorem 7 – an approximated obedient reasonable order $L \leq_r^{aO} L'$ is introduced.

The sufficient criterion for the obedient aftermath still demands, for landmarks L and L', that L is transitively ordered before some third landmark L_n that participates in an AND-node with L'. By "transitively ordered", we now mean that there is a chain $L = L_1 \leq , \ldots, \leq L_n$ where the ordering relations are taken out of the constraints O we start from. By "AND-node", we now mean that either L' and L_n are both top-level goals, or there is some other landmark L_{n+1} such that $L' \leq_{gn} L_{n+1}$, and $L_n \leq L_{n+1}$ is in O (subject to the additional constraint that at least two consecutive facts on the chain $L_1, \ldots, L_{n+1}$ can not be achieved simultaneously). Our implementation generates all pairs L and L' of facts in the LGT that satisfy these conditions, in a manner very similar to what we described above for the approximated reasonable orderings. There is a **for** loop over all pairs L' and

6. The descendants of L' itself are already ordered before L' so we need not consider them.

L_n of facts that participate in an AND-node as required here, then all pairs L and L' are generated where L is a descendant of L_n (subject to the above additional constraint) in the LGT that includes all $\leq^a_{gn}$ and $\leq^a_r$ orderings. If L interferes with L', then the ordering $L \leq^{aO}_r L'$ is introduced.

To illustrate this, let us finish off our running *Blocksworld* example. Have a look at Figure 3. As described earlier, approximating reasonable orderings introduces (amongst others) an additional edge from *clear(C)* to *on(B D)*. Say we try to extract obedient reasonable orderings from this enriched LGT. With the additional edge, L' = *holding(B)* and L_n = *clear(C)* participate in an AND-node in the sense explained above, because they are both ordered before *on(B D)* (more precisely, *holding(B)* $\leq^a_{gn}$ *on(B D)*, and *clear(C)* $\leq$ *on(B D)* is in O). Now, in particular L_n = *clear(C)* is a descendant of itself in the LGT, by the empty landmark chain (set $L = L_1 = L_n$ in the above). Moreover, *clear(C)* and *on(B D)* can not be achieved simultaneously (so the additional constraint is met). Thus Lemma 2 fires and we conclude that L' = *holding(B)* is in the obedient aftermath of L = *clear(C)*. Now we test if L interferes with L', and, indeed, we find that condition 4 of Definition 7 fires because *arm-empty* $\leq^a_{gn}$ *clear(C)*, and *arm-empty* is inconsistent with *holding(B)*. Hence we identify the order *clear(C)* $\leq^{aO}_r$ *holding(B)* – which prevents the planner from even picking B up prior to unstacking D from C.

4.2.4 Cycle Handling

In general, the final LGT including all $\leq^a_{gn}$, $\leq^a_r$, and $\leq^{aO}_r$ orderings may contain cycles. For example, the cycle $L \leq_{gn} L'$ and $L' \leq_r L$ might arise if a landmark must be achieved more than once in a solution plan (in the *Blocksworld* domain, cycles of this kind sometimes arise for the fact *arm-empty*). In our current implementation, any cycles in the orderings are removed since they aren't exploited by the search process (as said before, exploiting the information given by cycles is a topic for future work). The cycles are removed by removing edges – ordering constraints – that participate in cycles. Obviously, one would like to remove as few edges as possible. But figuring out the smallest edge set sufficient to break all cycles is NP-hard (FEEDBACK ARC SET (Garey & Johnson, 1979)). We experimented with a variety of methods that greedily remove edges until there are no more cycles. In these experiments, done over a range of benchmark domains, we generally found that there was little to be gained by the different methods. We settled on the following simple removal scheme: (i) remove all edges incident on cycles that correspond to *obedient reasonable* ($\leq^{aO}_r$) orderings; (ii) remove all edges incident on cycles that correspond to *reasonable* ($\leq^a_r$) orderings (after this, the cycles are all removed because $\leq^a_{gn}$ orderings alone can not form cycles due to the way they are computed). In other words, we prioritise to keep necessary over reasonable over obedient reasonable ordering constraints in the LGT. This makes intuitive sense due to the stronger theoretical justification of the different types of orderings.

4.3 Lookahead Orderings

We have implemented another form of orderings, to overcome certain shortcomings of the technology described so far. As said, our approximation of (greedy) necessary orderings is based on intersecting action preconditions. We get an ordering $L \leq^a_{gn} L'$ if all achievers (in

the earliest RPG level) of L' share the precondition L. Now, there are situations where the immediate achievers of L' do not have a shared precondition, but the achievers that are (at least) *two* steps away have one. We observed this in the *Logistics* domain. Let us use this domain for illustration. Say we are facing a *Logistics* task where there are two planes and a package to be moved from *la-post-office* to *boston-post-office*. While extracting landmarks, we will find that L' = *at(pack1 boston-airport)* is a landmark (a landmark candidate, at this point in the algorithm). L' is achieved by the two actions a_1 = *unload pack1 plane1 boston-airport* and a_2 = *unload pack1 plane2 boston-airport*. The preconditions of a_1 are *at(plane1 boston-airport)* and *in(pack1 plane1)*, those of a_2 are *at(plane2 boston-airport)* and *in(pack1 plane2)*. The intersection of these preconditions is empty, so the landmark extraction process described above would stop right here. But no matter if we use a_1 or a_2 to achieve L', the package has to be *in* a plane beforehand; and all the earliest actions in the RPG that achieve such an *in* relation share the precondition L = *at(pack1 la-airport)* (the actions are *load pack1 plane1 la-airport* and *load pack1 plane2 la-airport*). Thus *at(pack1 la-airport)* is a landmark, which can't be found by our above approximation techniques.

We designed a preliminary technique implementing a limited lookahead, during the landmarks extraction process, in order to overcome the difficulty exemplified above. The general situation where we can introduce a *lookahead necessary* ordering $L \leq_{ln} L'$ between facts L and L' is this. Say L' is a landmark, and can be achieved by the actions $a_1, \ldots, a_n$. Say $\{L_1, \ldots, L_m\}$ is a set of facts such that the precondition of each action a_i, $1 \leq i \leq n$, contains at least one fact L_j, $1 \leq j \leq m$. Then the disjunction $L_1 \vee \ldots \vee L_m$ is a landmark in the sense that one of these facts must be true at some point on every solution plan. Now, if all actions that achieve any of the facts L_j share some precondition fact L, then it follows that L is a landmark – a landmark that has to be true (at least) two steps before L' becomes true. When intersecting preconditions not over all actions but only over the earliest achievers in the RPG, we can apply this general principle to the *Logistics* example above: L' is *at(pack1 boston-airport)*, L_1 and L_2 are *in(pack1 plane1)* and *in(pack1 plane2)*, and L is *at(pack1 la-airport)*.

The idea is to test, given a landmark candidate L', if there is an intermediate fact set $\{L_1 \vee \ldots \vee L_m\}$ as above such that the intersection of the preconditions of all achievers of facts in the set is non-empty. Candidates for such intermediate fact sets can be generated by selecting one precondition from each achiever of L'. The obvious difficulty is that the number of such candidates is exponential in the number of different achievers of L'. We implemented a correct but incomplete solution by restricting the test to candidate sets where all facts are based on the same predicate – as the *in* predicate in *in(pack1 plane1)* and *in(pack1 plane2)* above. If there is such a fact set $\{L_1 \vee \ldots \vee L_m\}$, then we check if there is a shared precondition L of the actions achieving the facts L_j at the respective earliest points in the RPG. If the test succeeds, L becomes a new landmark candidate. If, later, both L and L' turn out to really be landmarks, then we introduce the ordering $L \leq^a_{ln} L'$ into the LGT. When approximating reasonable (or obedient reasonable) orderings, the $\leq^a_{ln}$ orders are (only) used at those points where the aftermath relation is checked. More precisely, when it is checked if some landmark L is transitively ordered before some landmark L_n via a sequence $L = L_1 \leq^a_{gn}, \ldots, \leq^a_{gn} L_n$ of greedy necessary orderings, we also allow lookahead necessary orderings. To explain this approach: it is easy to see that Theorems 6 and 7

remain valid for this use of $\leq_{ln}$ orderings. But the theorems do not remain valid when using $\leq_{ln}$ orderings in *any* place where $\leq_{gn}$ orderings can be used.

We found our implementation of this preliminary technique to work well in the *Logistics* domain, see also Section 6.6. From a more general point of view, the technique opens up two interesting lines of future research, regarding *disjunctive landmarks* and *k-lookaheads*. We say more on these topics in the discussion, Section 7.

5. How to Use Ordered Landmarks

Having settled on algorithms for computing the LGT, there is still the question of how to use this information to speed up planning. Porteous and Sebastia (2000) proposed a method that prunes states where some landmark has been achieved too early. The method is applicable (only) in forward state space search. If applying an action achieves a landmark L that is not a leaf of the current LGT, then do not use that action. If an action achieves a landmark L that *is* a leaf, then remove L (and all ordering relations it participates in) from the LGT. In short, do not allow achieving a landmark unless all of its predecessors have been achieved already. Apart from its restriction to a single search approach, the performance improvements obtainable this way appear rather limited. In many domains there are no improvements at all.[7]

Here, we explore an alternative idea that uses landmarks in a more constructive manner, by ways of a decomposition method. Rather than telling the planner what is probably *not* a good thing to do next, the landmarks now tell the planner what probably *is* a good thing to do next. This yields performance improvements in domains that were previously unaffected, and is not restricted to any particular kind of planning approach.

Section 5.1 introduces our decomposition method. Section 5.2 makes some remarks about the theoretical properties of the method, and Section 5.3 discusses a few variations of the method that we have tried (and found to not work as well in practice).

5.1 Disjunctive Search Control

We use the LGT to decompose the planning task into a series of smaller sub-tasks. The decomposition takes place in the form of a search control algorithm that is wrapped around some – any – planning algorithm, called *base planner* in what follows. Similar to before (Porteous & Sebastia, 2000), the core technique is to consider the leaf nodes in an incrementally updated LGT. First, read in the task and compute the LGT. Then, in any search iteration, consider the leaf nodes of the current LGT and *hand these leafs over to the base planner as a goal*. When the base planner has returned a plan, update the LGT by removing the achieved leaf nodes, and iterate. The main difficulty with this idea is that the leaf nodes of the LGT can often not be achieved as a conjunction. Our solution is to pose the leaf nodes as a *disjunctive* goal instead. See the algorithm in Figure 5.

The depicted algorithm keeps track of the current state s, the current plan prefix P, and the current disjunctive goal $Disj$, which is always made up out of the current leaf nodes of the LGT. The initial facts are immediately removed because they are true anyway. When

7. By this kind of search control one does not get any benefit out of the necessary orderings, as these will be adhered to by the forward search anyway. But in many domains (like *Logistics*-type problems) there are no or hardly no ordering constraints other than the necessary ones.

$s := I,\ P := \langle\,\rangle$
remove from LGT all initial facts and their edges
repeat
    $Disj :=$ leaf nodes of LGT
    call base planner with actions A, initial state s and goal condition $\bigvee Disj$
    if base planner did not find a solution P' **then** fail **endif**
    $P := P \circ P',\ s :=$ result of executing P' in s
    remove from LGT all $L \in Disj$ with $L \in add(o)$ for some o in P'
until LGT is empty
call base planner with actions A, initial state s and goal $\bigwedge G$
if base planner did not find a solution P' **then** fail **endif**
$P := P \circ P'$, output P

Figure 5: Disjunctive search control algorithm for a planning task (A, I, G), repeatedly calling an arbitrary planner on a small sub-task.

the LGT is empty – all landmarks have been processed – then the algorithm stops, and calls the underlying base planner from the current state with the original (top level) goals. The search control fails if at some point the base planner did not find a solution. A disjunctive goal can be simulated by using an artificial new fact G as the goal, and adding one action for each disjunct L, where the action's precondition is $\{L\}$ and the add list is $\{G\}$ (this was first described by Gazen and Knoblock (Gazen & Knoblock, 1997)). So our search control can be used around any base planner capable of dealing with STRIPS input. Note that, as all top-level goals are landmarks, an empty LGT means that all goals have been achieved at least once. So unless they have been destroyed again, the initial state for the last call of the base planner will already contain the goals (some more on this below).

5.2 Theoretical Properties

The search control is obviously correctness preserving – eventually, the planner is run on the original goal. Likewise obviously, the method is not optimality preserving in general. Though we did not explore this topic in depth, we do not believe that there are interesting special cases in which optimality is preserved. In the benchmarks we tried, the runtime improvements were indeed often bought at the price of somewhat longer plans, see Section 6.

With respect to completeness, matters are more interesting. As it turns out, the approach is completeness preserving in the large majority of the current benchmarks. The reasons for this are that there, no fatally wrong decisions can be made in solving a subtask, that most facts which have been true once can be made true again, and that necessary ordering relations are respected by any solution plan. We need two notations.

1. A *dead end* is a reachable state from which the goals can not be reached anymore (Koehler & Hoffmann, 2000), a task is dead-end free if there are no dead ends in the state space.

2. A fact L is *recoverable* if, when s is a reachable state with $L \in s$, and s' with $L \notin s'$ is reachable from s, then a state s'' is reachable from s' with $L \in s''$.

E.g., many of the current benchmarks are invertible in the sense that every action a has a counterpart $\overline{a}$ that undoes a's effects (Hoffmann, 2002). Such tasks are dead-end free, and all facts in such tasks are recoverable. Completeness is preserved under the following circumstances.

Theorem 8 *Given a solvable planning task (A, I, G), and an LGT where each fact L in the tree is a landmark such that $L \notin I$. If the task is dead-end free, and for all facts L' in the tree it holds that either L' is recoverable, or all orders $L \leq L'$ in the tree are necessary, then running any complete planner within the search control defined by Figure 5 will yield a solution.*

Proof: Assume that the search control fails at some point. We prove that then, in contradiction to the assumption, s is a dead end. This is obvious if all landmarks have been processed already: the goal to be reached is the original goal. In the other case, the disjunction of all leaf nodes is unsolvable. Say L' is one such leaf. Say P' is a plan solving the goals from s. P' does not add L', as it would otherwise solve the disjunction. The concatenation of the current prefix P with P' is a solution plan, and L' is a landmark not contained in the initial state, so P adds L'. If L' is recoverable, we can achieve it from s and have a contradiction to the unsolvability of the disjunction. Therefore, L' is not recoverable, implying by prerequisite that all orders $L \leq L'$ in the tree are necessary. Say $P = \langle a_1, \ldots, a_n \rangle$, and L' is first added by a_i. At this point, L' was not in the disjunction, as it would otherwise have been removed from the tree. So there is some L with $L \leq L'$ that is first added by some action between a_i and a_n. But then, P does not respect the ordering constraint $L \leq L'$, and neither does the solution plan $P \circ P'$ do so, which is a contradiction to $L \leq_n L'$. It follows that s is a dead end. ■

Verifying landmarks with Proposition 1 ensures that all facts in the LGT really are landmarks. The initial facts are removed before search begins. The tasks contained in domains like *Blocksworld*, *Logistics*, *Hanoi* and many others are invertible (Hoffmann, 2002). Examples of dead-end free domains with only necessary orders are *Gripper* and *Simple-Tsp*. Examples of dead-end free domains where non-necessary orders apply only to recoverable facts are *Miconic*-STRIPS and *Grid*. All those domains (or rather, all tasks in those domains) fulfill the requirements for Theorem 8.

5.3 Search Control Variations

The disjunctive search control in Figure 5 calls the base planner with the original (top level) goals once the whole LGT has been processed. So the only way the search control can speed up the planning process is if the initial state s for that last call of the base planner is sufficiently much closer to a goal state than the original initial state was. Recall that all top level goals are also landmarks, that they thus have been achieved at least once in the control loop, and that they thus will be true in s *unless they have been deleted again in some later control loop iteration.* An obvious idea to avoid the latter phenomenon is, once

a top level goal G has been achieved, to force the base planner to keep G true throughout the rest of the control process. This can be done by keeping a *conjunctive goal* $Conj$ in addition to the disjunctive goal $Disj$. $Conj$ then always contains the top level goals that have been achieved so far – that have been leaf landmarks and were removed – and the goal for the base planner is always $Conj \wedge \bigvee Disj$. The problem with this idea is that one or a set of already achieved original goals might be inconsistent with the leaf landmarks in a later iteration. Forcing the achieved goals to be true together with the disjunction yields in this case an unsolvable sub-task, making the control algorithm fail. We observed this in various benchmarks. We tried a number of ideas based on detecting inconsistencies (with the TIM API) between $Conj$ and $Disj$, and removing facts from $Conj$ that participate in an inconsistency (there is no point in removing the inconsistent leaf landmarks as these *must* be achieved sometime). However, removing inconsistent facts from $Conj$ did not help avoiding unsolvable sub-tasks, due to the incompleteness of TIM's inconsistency approximation (in particular, only *pairwise* inconsistencies are detected). Apart from all this, even in cases where forcing truth of achieved top level goals did not make the search control fail, the technique did not yield better runtime or solution length behaviour in our experiments. This is probably due to the fact that, unless such a goal is inconsistent with some landmark ahead, it is kept true anyway.

We remark that the source of the difficulties observed above is the incompleteness of the inconsistency approximation. If the approximation of binary inconsistencies was complete, then a top level goal G already achieved as a leaf landmark could never be inconsistent with a leaf landmark L in a later iteration. This is because our ordering approximation techniques would detect a reasonable ordering constraint $L \leq_r G$ between L and G: G is a top-level goal and thus in the aftermath of L, and by the detected inconsistency L interferes with G so $L \leq_r G$ can be concluded with Theorem 6. Thus L would be ordered before G in the LGT, and G could not become a leaf landmark first.

There is another aspect of the search control defined by Figure 5 that, at first sight, appears to be an obvious shortcoming: while the leaf landmarks can often not be achieved together in one piece, this does not justify the assumption that *all single leafs must be achieved separately*. But posing the leafs as a disjunctive goal suggests the latter to the base planner. In some domains (like *Logistics*), this increases plan length considerably (see also Section 6.6). A more precise way of dealing with inconsistencies among leaf landmarks is to hand the leafs over to the base planner in the form of a DNF goal, where the members of the disjunction are a partition of the leaf landmarks into *maximal consistent subsets*. Similar to what we have seen above, the main difficulty with this idea is that inconsistency can not be determined exactly (without solving the planning task in the first place). One can approximate consistency by pairwise consistency according to the TIM API, and obtain a partition into maximal consistent subsets in a greedy manner.[8] But of course undetected inconsistencies can make the search control fail. We observed this in various benchmarks. Moreover, in our experiments, independently of the base planner used, the effect of the modified technique on performance was not convincing even in those cases were it preserved solvability. There are a few domains (e.g. *Logistics*) were, compared to the fully disjunctive

8. Finding a partition (DNF) with minimal number of subsets (disjuncts) is NP-hard: this solves PARTITION INTO CLIQUES (Garey & Johnson, 1979) when the graph edges are the pairs of facts that are consistent.

control, the technique improved plan length at the cost of longer runtimes. In other domains (e.g. *Blocksworld-arm*) the modification made no significant difference, in some domains (e.g. *Freecell*) the modification produced clearly worse runtime *and* plan length behaviour.

We remark that, when using both modifications outlined above, our search control becomes a generalisation of the "Goal Agenda Manager" proposed by Koehler and Hoffmann (Koehler, 1998; Koehler & Hoffmann, 2000). In the goal agenda, the top level goals are partitioned into a series of subsets that respects their (approximated) reasonable ordering constraints. The goal handed over to the base planner in any control loop iteration i is then the union (conjunction) of all partition subsets up to point i. Restricted to top level goals, the modified disjunctive search control becomes exactly this. This is because the leaf goals in iteration i correspond exactly to the respective partition subset. Consider the workings of the search control under the above modifications. Top level goals are consistent (in solvable tasks) so will be posed as a single conjunctive goal when considering maximal consistent subsets. The previously achieved leaf landmarks will be exactly the goals in previous partition subsets, so when keeping them in the conjunctive goal *Conj* we end up with exactly what the Goal Agenda Manager does. The generalisation lies in that, in the presence of non-top level goal landmarks, the goal agenda process is enriched with more facts and ordering relations, and a combination of conjunctive and disjunctive (sub-)goals. However, as outlined above, such a control process suffers from difficulties arising from the incompleteness of the inconsistency approximation, and the simpler fully disjunctive search control framework from Figure 5 typically works better. So we have concentrated on this simple framework in our more extensive experiments.

6. Results

We chose to run our disjunctive search control around the three planners FFv1.0, FFv2.3, and LPG. The reason for this choice was that we wanted to show the effects of our techniques on the state-of-the-art in sub-optimal planning. (As our techniques do not preserve optimality there is not much point in using them with optimal planners, which are generally outperformed by suboptimal planners anyway.) FFv1.0 (Hoffmann, 2000) is an early STRIPS version of FF that does not use any goal ordering techniques. FFv2.3 (Hoffmann & Nebel, 2001) is the version of FF that participated in the AIPS-2000 and AIPS-2002 planning competitions. The planner enhances FFv1.0 with the Goal Agenda Manager technique (as well as the ability to handle ADL). We consider FFv2.3 a particularly interesting planner to try our landmarks techniques on as, by doing this, we give an example of how our search control affects the performance of a planner that already uses goal ordering techniques. The LPG version we used (Gerevini et al., 2003) is the one that participated in the AIPS-2002 planning competition.

To come up with a test suite, we ran preliminary tests in 19 different STRIPS benchmark domains, including all (STRIPS) examples used in the AIPS-1998, AIPS-2000, and AIPS-2002 competitions. We selected 8 out of the 19 domains for more extensive tests. The selected domains are: *Blocksworld-arm*, *Blocksworld-no-arm*, *Depots*, *Freecell*, *Grid*, *Logistics*, *Rovers*, and *Tyreworld*. The reasons why we discarded the other domains were the following. The *Movie* instances were trivial to solve for any planner configuration we tried. In *Driverlog*, *Gripper*, *Mprime*, *Mystery*, *Satellite*, and *Zenotravel*, no (or only very few)

non-trivial landmarks/ordering relations were detected so the decomposition imposed by our disjunctive search control essentially came down to serialising the goal set (i.e., achieving all goals one after the other).[9] In *Hanoi* there was a non-trivial LGT but whether to use it or not made no significant difference to any planner's runtime performance (intuitively, because the ordering information is overpowered by the exponentiality in the length of the *Hanoi* solutions). In the STRIPS version of *Schedule*, where operators have a lot of parameters, none of our planners could cope with the pre-processing (namely, the grounding of the operator parameters with all available objects). In *Ferry* and *Miconic-STRIPS*, finally, the performance observations we made were basically the same as those that we made in *Logistics*. As all these three domains are also semantically very similar, we focussed on only a single representative of this class of domains.

For each of the 8 domains we selected for experimentation, we used a problem generator to produce a large suite of test examples. The examples scale in the respective domain's size parameters (e.g., number of blocks in the *Blocksworld* variants), and of each size there are several (in most domains, 20) random instances. In what follows, there is one subsection for each of the 8 domains. In each subsection, we: give a brief description of the domain; provide a table with solution percentage values as well as averaged runtimes and plan lengths (number of actions in the plan); give some intuitions on what kind of ordering information our methods provide in the domain; and discuss the results. The experiments were run on an Athlon 1800 MHz machine with 1Gb RAM, running Linux (Mandrake 9.0). Unsuccessful runs were cut off after a time limit of 300 seconds. The implementation of our landmarks techniques is based on FFv1.0, and the disjunctive search control is integrated in that planner's code. For the other planners, FFv2.3 and LPG, we implemented a simple interface. For each iteration of the disjunctive search control, the respective sub-task is specified via two files in the STRIPS subset of PDDL (McDermott et al., 1998). The implementations of FFv2.3 and LPG are modified to output a results file containing the spent running time, and a sequential solution plan (or a flag saying that no plan has been found). For FFv1.0, the runtime we measure is simply total execution time. For FFv2.3, the runtime we measure also is total execution time, except the time taken in the interface, i.e. the time taken to create the PDDL files – after all, this time is just an unnecessary overhead due to our preliminary implementation. For LPG, matters are a bit more complicated. LPG computes inconsistent facts and actions as a pre-process to planning (inconsistent actions either interfere, or have inconsistent preconditions). Repeatedly calling LPG inside our landmarks control results in repeatedly doing the inconsistency pre-process, producing a large runtime overhead. In a direct implementation of landmarks control within LPG, one would of course do the pre-process only once. Our idea for an experimental implementation thus is to simply ignore the runtime overhead incurred by the superfluous pre-processes. However, in general the inconsistencies computed by the individual pre-processes can be different, depending on the start state LPG is run on. So there is no guarantee that our experimental implementation will produce the same results as a direct implementation, *unless* it is the case that LPG's

9. In the transportation domains *Driverlog*, *Gripper*, *Mprime*, *Mystery*, and *Zenotravel*, due to the presence of several vehicles (gripper hands in the case of *Gripper*) there are no shared preconditions of the actions that can achieve the goal position of an object. The only ordering relations we get refer to the initial position and the goal position of objects. We elaborate this further in the discussion of *Logistics* in Section 6.6.

pre-process provably computes the same information throughout our landmarks control loop. But we found that the latter is indeed the case in 7 of our 8 domains. The reason for this is that state reachability is (largely, in some cases) invertible in these domains; started in states that are reachable from each other, LPG's pre-process finds the same inconsistencies. The detailed arguments are in Appendix B. The only domain where LPG's pre-process might find different information for the different start states is *Freecell*. There we count the total runtime of our experimental implementation, including *all* inconsistency pre-processes. In the other 7 domains, we count the runtime for only a single (the first) one of these pre-processes.[10] It is a piece of open work to integrate landmarks control more tightly with LPG, see also the outlook in Section 7. With the preliminary implementation we evaluate here, the LPG results should be interpreted with care; still they show that one can obtain runtime improvements for LPG when using landmarks to structure the search.

We also ran our disjunctive landmarks control around IPP (Koehler, Nebel, Hoffmann, & Dimopoulos, 1997), and a standard naive breadth-first search. In both cases, we obtained dramatic runtime improvements in all the domains we consider here. We focus on the domains in alphabetical order.

6.1 Blocksworld-arm

Blocksworld-arm is the variant of the Blocksworld that we used in our illustrative example. A robot arm can be used to arrange blocks on a table. There are four operators to stack a block onto another block, to unstack a block from another block, to put a block (that the arm is holding) down onto the table, and to pick a block up from the table. Using the software provided by Slaney and Thiebaux (2001), we generated examples with 20, 22, 24, 26, 28, and 30 blocks, 20 instances per size. Our data is displayed in Table 1.

From left to right, the entries in Table 1 provide the data for the instances of increasing size. For planner "X", "X+L" denotes the same planner, but with our landmarks search control. The top part of the table provides the percentage of instances solved by each planner in each class of instances. The middle part provides averaged runtimes, the bottom part provides averaged plan lengths. For each planner "X", the averages here are computed over (only) those instances that were solved by *both "X" and "X+L"*. In unsolved cases, there is no plan length to include into the average computation; we have tried to use the runtime cutoff value in the runtime average computation but this generally obscured our results more than it helped to understand them. We average over the instances solved by individual planners "X" and "X + L", rather than over instances solved by all planners, because this way we obtain a clearer picture of what the impact of our techniques on each individual planner is. Note that, this way, comparisons between different planners have to be made very carefully. In what follows, we will indeed concentrate on the effects of our landmarks techniques and not say much on inter-planner comparisons.

There are all kinds of landmarks and ordering relations in *Blocksworld-arm*, as we have already seen during the discussions regarding our illustrative example. Our empirical results suggest that our approximation of this information is useful for improving planners. Looking at the solution percentage values, it is obvious that the landmarks help to improve all of

10. There is a further subtlety regarding another pre-process that LPG performs before starting the search; see the discussion in Appendix B.

%solved						
FFv1.0	25	30	20	0	5	0
FFv1.0+L	85	70	65	60	70	70
FFv2.3	80	75	65	50	55	50
FFv2.3+L	85	95	80	60	60	65
LPG	100	100	90	50	35	30
LPG+L	100	100	95	85	80	65
time						
FFv1.0	7.33	9.28	16.00	-	46.99	-
FFv1.0+L	0.29	0.16	1.05	0.34	0.26	1.01
FFv2.3	16.78	6.04	2.66	10.56	0.51	2.05
FFv2.3+L	2.17	6.41	3.54	21.88	6.38	7.08
LPG	38.60	61.58	97.44	145.20	206.48	211.43
LPG+L	9.67	24.03	63.45	31.61	99.51	112.79
length						
FFv1.0	77.20	80.50	74.67	-	92.00	-
FFv1.0+L	60.00	68.50	85.33	80.83	74.00	99.57
FFv2.3	64.29	64.86	75.45	78.50	88.25	88.75
FFv2.3+L	276.43	319.00	346.55	406.50	485.25	439.75
LPG	250.70	308.90	378.33	383.00	419.71	391.33
LPG+L	219.90	255.10	313.56	317.00	338.29	456.67

Table 1: Experimental Results in *Blocksworld-arm*.

the tested planners. The improvement is drastic for FFv1.0 and LPG, and less drastic but still significant for FFv2.3. The runtimes get clearly improved for FFv1.0 and LPG; FFv2.3+L is a little slower than FFv2.3 in those cases solved by both planners. FFv1.0 does not solve a single instance with 26 or 30 blocks, and in the respective table entries for FFv1.0+L, we averaged over all instances solved by FFv1.0+L. For FFv1.0 and LPG, the impact on plan length is a bit inconclusive; most of the time the plans become somewhat shorter when using landmarks. A different picture emerges for FFv2.3 where the plans become a lot longer. (We suspected that the latter phenomenon is due to an interaction between our landmarks control and the Goal Agenda that FFv2.3 uses; investigating this, we did however not find such an interaction. Presumably, the odd plan length behaviour of FFv2.3+L is just an effect of minor implementational differences between FFv1.0 and FFv2.3, such as the ordering of facts and actions in the internal representation.)

6.2 Blocksworld-no-arm

Blocksworld-no-arm is a variant of the Blocksworld where the blocks are moved around directly, i.e. without explicit reference to a robot arm. There are three operators to move a block from another block onto a third block, to move a block from the table onto another block, and to move a block from another block onto the table. As in *Blocksworld-arm*, we used Slaney and Thiebaux's software to generate random examples with 20, 22, 24, 26, 28, and 30 blocks, 20 instances per size. See the data in Table 2.

There are all kinds of landmarks and ordering relations between them in *Blocksworld-no-arm*. For example, in order to move some block x onto a block y, x and y must always be clear (so these fact are shared preconditions). There are destructive interactions between the different sub-goals (e.g. a block can not be clear and have some other block on top of it), which lead to reasonable orderings. Using our techniques, solution percentage gets dramatically improved for FFv2.3; for LPG there is some improvement, too, for FFv1.0

%solved						
FFv1.0	95	80	55	85	50	15
FFv1.0+L	80	70	65	70	35	50
FFv2.3	90	65	55	60	45	45
FFv2.3+L	100	95	100	100	95	100
LPG	100	100	100	100	80	5
LPG+L	100	100	100	100	90	20
time						
FFv1.0	7.57	25.18	29.14	28.74	88.33	106.75
FFv1.0+L	8.93	19.89	27.36	36.34	69.84	84.32
FFv2.3	16.67	27.87	46.13	74.21	44.50	65.59
FFv2.3+L	0.86	1.78	1.78	2.34	3.22	12.11
LPG	21.71	43.19	80.45	135.58	197.32	-
LPG+L	16.82	40.79	67.46	118.89	207.69	-
length						
FFv1.0	28.00	32.69	36.00	37.17	42.60	46.67
FFv1.0+L	65.20	81.38	80.12	90.92	110.00	112.67
FFv2.3	47.56	49.67	58.82	69.33	64.89	76.44
FFv2.3+L	46.67	54.67	62.18	68.17	78.44	78.78
LPG	74.00	86.85	97.45	105.05	114.00	-
LPG+L	97.60	130.00	143.95	154.80	188.50	-

Table 2: Experimental Results in *Blocksworld-no-arm*.

the results are a bit inconclusive but of the largest examples, FFv1.0+L solves a lot more than FFv1.0. The average runtimes in solved examples become clearly better for FFv2.3. For FFv1.0 and LPG, they generally become somewhat better too, but not significantly. LPG and LPG+L solved different instances in the largest example group, which is why the respective table entries are empty. As for average plan length, this becomes slightly worse for FFv2.3 and significantly worse for FFv1.0 and LPG.

While the ordering relations present in *Blocksworld-no-arm* are similar to those in *Blocksworld-arm*, in difference to the latter domain our observed performance improvements are not as significant. One important reason for this is probably that, for all of FFv1.0, FFv2.3, and LPG, *Blocksworld-no-arm* is not as problematic as *Blocksworld-arm* anyway: in both domains our examples contain the same numbers of blocks, but the solution percentages without using landmarks are usually much higher in *Blocksworld-no-arm*.[11]

6.3 Depots

The *Depots* domain is a mixture between *Blocksworld-arm* and *Logistics*, introduced by Long and Fox in the AIPS-2002 planning competition (Long & Fox, 2003). Stacks of crates at different locations must be re-arranged. There are five operators to drive trucks between places, to load/unload a crate onto/from a truck using a hoist, and to drop/lift a crate onto/from a surface (another crate or a pallet) using a hoist. For our random instances, we used the problem generator provided by Long and Fox. The instances all feature 10 locations, 3 trucks, 10 hoists (one at each location), and 20 pallets (at least one at each location). The instances scale in terms of the number of crates. Here, we show the data for instances with 20 to 25 crates. As we found quite a lot of variance in the runtime results for

11. Indeed, Hoffmann (2002) proves that *Blocksworld-no-arm* is an easier domain than *Blocksworld-arm* for planners (such as FF and LPG) using heuristics based on the relaxation that ignores all delete lists.

%solved						
FFv1.0	62.5	72.5	52.5	42.5	22.5	20
FFv1.0+L	85	80	75	67.5	72.5	62.5
FFv2.3	67.5	67.5	55	45	45	27.5
FFv2.3+L	47.5	45	37.5	35	30	30
LPG	100	100	100	100	100	82.5
LPG+L	100	100	100	95	97.5	97.5
time						
FFv1.0	98.18	157.29	130.69	206.64	175.82	146.81
FFv1.0+L	9.48	10.32	26.47	20.12	22.33	41.01
FFv2.3	139.19	138.47	92.49	163.92	185.84	218.27
FFv2.3+L	38.75	15.10	36.27	65.76	8.74	10.63
LPG	32.39	47.59	63.89	78.72	87.23	111.45
LPG+L	22.60	26.71	46.32	51.27	53.38	74.60
length						
FFv1.0	108.86	118.00	121.00	129.64	131.00	127.43
FFv1.0+L	128.48	146.17	152.47	154.27	170.44	161.14
FFv2.3	118.42	118.00	118.00	132.38	132.20	130.00
FFv2.3+L	133.50	141.21	150.09	159.62	154.00	159.50
LPG	137.70	155.43	157.22	165.87	178.85	184.88
LPG+L	154.38	156.03	176.22	194.76	197.41	200.09

Table 3: Experimental Results in *Depots*.

individual *Depots* instances, we generated 40 random instances of each size. See the data in Table 3.

The hoists and crates in *Depots* roughly correspond to the robot arm and blocks in *Blocksworld-arm*, and similarly to that domain we get various landmarks and orderings regarding *on*-relations and *clear*-relations. In addition, we get landmarks to do with the transportation part of the problem, regarding the hoists that must be used in order to load/unload crates onto/from trucks.[12] The effect of our technique on solution percentage is very different for the individual planners in this experiment. While FFv1.0 is drastically improved, FFv2.3 becomes significantly worse; for LPG, the data is a bit inconclusive, and we ran additional experiments (see below). Runtime and plan length are easier to interpret: the runtime in solved instances becomes significantly better for all the planners, especially the FF versions; this improvement is obtained at the cost of somewhat longer plans.

The solution percentage data for LPG suggests that the instances in Table 3 are not yet at LPG's scaling limit. We generated larger instances, with 26 to 30 crates, 3 random examples of each size. The solution percentage values we got for these examples are (LPG/LPG+L): 100/100, 66.6/100, 0/100, 0/66.6, 0/66.6. This suggests that our landmarks techniques pay off more for LPG as the instances become harder. The runtime and plan length behaviour in cases solved by both planners remained similar to the smaller instances: LPG + L was faster than LPG on average, but found somewhat longer plans.

6.4 Freecell

The *Freecell* domain is a STRIPS encoding of the well-known solitaire card game that comes free with Microsoft Windows. The domain was introduced by Fahiem Bacchus in the

12. Due to the presence of several trucks, there are no non-trivial orderings regarding the actual movements of trucks and crates; see the discussion of transportation domains in the section about *Logistics*.

%solved						
FFv1.0	95	85	80	50	35	15
FFv1.0+L	95	65	60	50	10	5
FFv2.3	100	90	85	60	20	15
FFv2.3+L	100	70	70	30	20	5
LPG	0	0	0	0	0	0
LPG+L	10	0	0	0	0	0
time						
FFv1.0	3.95	46.82	54.28	92.60	-	261.65
FFv1.0+L	1.62	2.15	3.05	5.45	-	9.34
FFv2.3	6.02	24.93	62.53	66.56	-	140.37
FFv2.3+L	1.80	2.41	3.31	4.45	-	8.63
LPG	-	-	-	-	-	-
LPG+L	240.23	-	-	-	-	-
length						
FFv1.0	68.00	88.00	103.00	107.25	-	125.00
FFv1.0+L	63.00	69.50	81.50	99.25	-	119.00
FFv2.3	75.50	85.00	93.00	106.50	-	123.00
FFv2.3+L	60.50	67.50	82.00	98.00	-	121.00
LPG	-	-	-	-	-	-
LPG+L	72.50	-	-	-	-	-

Table 4: Experimental Results in *Freecell*.

AIPS-2000 planning competition, and was also included by Long and Fox into the AIPS-2002 planning competition. A brief description of the game is as follows. A set of cards is initially randomly arranged across a number of stacks, and all cards have to be put away on goal stacks in a certain order. There are rules that guide in which ways cards can be moved, and there are a number of "free cells" which can be used for intermediate storage of cards. The domain comprises 10 operators that implement the possible card moves. As said above, in *Freecell* there is no guarantee that our experimental implementation behaves like a direct implementation of landmarks in LPG, so below we provide the total runtime taken. We remark that a direct LPG+L implementation might do better – when counting the time for only a single inconsistencies pre-process, we solved a few more examples within the time limit. To generate our test suite, we used the random generator provided by Long and Fox. Across our instances, we have 4 different suits of cards, 5 free cells, and 10 possible stacks; we scaled the instances by increasing the number of cards in each suit, 20 random instances per size. Table 4 provides our data for the examples with 11 to 16 cards per suit.

In a fashion reminiscent of the Blocksworld-like problems considered above, cards in *Freecell* can be stacked on top each other (under certain additional preconditions regarding their value and their colour), and must be clear in order to be moved. From these structures, we get landmarks regarding *on*-relations and *clear*-relations similar as before. In addition, we get landmarks regarding card values in cases where only one order of values is possible. Most particularly, the goal condition is expressed in terms of *home*-relations for (only) the *topmost* cards in the goal stacks. As the order in which cards can be put onto the goal stack is fixed, for each single card in the goal stacks we get its *home*-relation as a landmark, necessarily ordered before the respective next higher card.

For a reason we don't know, LPG behaved very badly in our *Freecell* experiments. It hardly solved any instance, and adding the landmarks control helped in some, but not many, cases. Averaging over these cases, we got the runtime and plan length averages shown in

the LPG+L entries. For the FF versions, the impact of our landmarks technique is more interesting. The impact on both versions is similar: less tasks are solved, but the runtime in the solved tasks improves considerably, and the plans become somewhat shorter.[13] Why is this? Well, what happens when playing *Freecell* is that the main difficulty is typically to avoid dead end situations where, due to some wrong card moves in the past, one can not move any card any longer, or make any more progress towards a solution. Now, FF uses a forward hill-climbing style search. Pretty much as a matter of chance, this either gets lucky and solves the task quickly, or it gets stuck in a dead end state. Using the landmarks search control, FF's hill-climbing search basically becomes even more greedy, being a lot faster in the "good" cases, but producing a few more "bad" cases. Now, if FF (without our control) encounters a dead end during hill-climbing, the planner starts from scratch, "falling back" onto a complete best-first search. If our landmarks control ends up in a dead end, no such complete fall back search method is invoked (though one could of course do that). The instances that FF's complete search method solves cause most of the difference in the solution percentage values as depicted in Table 4.

6.5 Grid

Grid is a domain that was introduced by Drew McDermott in the AIPS-1998 planning competition. A robot moves on a 2-dimensional grid of locations. The locations can be locked, in which case the robot can not move onto them. Locked locations can be opened with a key of matching shape. The robot can carry one key at a time, and the task is to transport a subset of the keys to their goal locations. For our test examples, we used the random generator provided on the FF homepage.[14] We scaled the instances in terms of parameters n and m. The grid has extension $n \times n$, there are 4 different possible shapes of keys and locks, of each shape there are m keys and m locked locations. Half of the keys must be transported to goal locations. Randomly distributing the initial locations of the keys and the locked locations, the generator sometimes produces unsolvable instances. We filtered these out by hand (whether a *Grid* instance is solvable or not can be concluded from the output of simple pre-processing routines such as are implemented in FF). Table 5 provides our data for the size values (n/m) 8/4, 8/5, 9/4, 9/5, 10/5, 10/6, 20 random instances per size.

The landmarks and orderings our techniques find in *Grid* are the following. The goal is given in the form of a set of *at*-relations of keys. These relations can be achieved using putdown actions, which all share the fact that the robot must be *holding* the respective key. Now, in the RPG the earliest point at which *holding* a key is achieved is by picking the key up at its initial location. So we get the respective landmarks for the position of the robot. (In some special cases, e.g. when the goal location of a key is locked, we also get landmarks regarding the opening of locked locations.) The resulting problem decomposition makes the planner choose an order in which to proceed towards the goal keys, and transport them to their goal. This decomposition turns out to be very useful for the planners in our experiment. The results for the FF versions are excellent. Solution percentage and

13. In the examples with 15 cards per suit, the instances solved by FFv1.0/FFv1.0+L (FFv2.3/FFv2.3+L) were different which is why the respective runtime and plan length table entries are empty.
14. At `http://www.informatik.uni-freiburg.de/~hoffmann/ff-domains.html`.

%solved						
FFv1.0	100	85	90	65	30	0
FFv1.0+L	100	100	100	95	85	90
FFv2.3	80	50	75	60	25	0
FFv2.3+L	90	95	80	60	50	50
LPG	40	15	35	5	0	0
LPG+L	80	70	70	5	0	0
time						
FFv1.0	40.92	126.82	75.85	132.44	136.24	-
FFv1.0+L	4.39	9.56	13.97	10.43	31.88	84.32
FFv2.3	35.17	64.05	74.35	91.58	113.35	-
FFv2.3+L	10.53	7.09	7.71	5.59	6.84	79.82
LPG	123.37	204.63	141.80	-	-	-
LPG+L	151.03	277.42	169.52	-	-	-
length						
FFv1.0	127.15	171.71	155.17	159.77	185.33	-
FFv1.0+L	131.75	166.24	153.33	153.00	176.67	251.61
FFv2.3	127.13	145.60	149.42	149.00	152.00	-
FFv2.3+L	122.93	135.50	148.50	125.33	143.75	250.10
LPG	141.29	131.50	134.50	-	-	-
LPG+L	149.14	131.00	141.50	-	-	-

Table 5: Experimental Results in *Grid*.

average runtime are improved significantly. This performance gain is not obtained at the cost of longer plans, rather differently the plans using landmarks control become slightly (but consistently) shorter. For LPG, the solution percentage is also improved a lot, but the time taken to solve the examples solved by both planners is somewhat worse for LPG+L. Also, the plans become a little longer.

6.6 Logistics

The well-known *Logistics* domain is the classical transportation benchmark in planning. Trucks and planes are the means to transport packages between locations. The locations are grouped together in cities, i.e. a single city can contain several locations. Trucks can only drive within cities, planes fly between different cities. There is one truck in each city. There are 6 operators to drive trucks, to fly planes, to load/unload packages onto/from trucks, and to load/unload packages onto/from planes. To generate our test suite, as in *Grid* we used the random generator provided on the FF homepage. Our instances all have 10 cities with 10 locations each, and 10 planes. We scaled the instances by the number of packages (which must all be transported). Table 6 provides our data for the examples with 40, 42, 44, 46, 48, and 50 packages, 20 random instances per size.

The solution percentage/runtime performance of both FF versions is improved significantly by the landmarks technique, at the cost of longer plans. For LPG, the solution percentage performance is a bit inconclusive but the average runtime over those instances solved by both LPG and LPG+L is better for LPG+L. The plans become longer.

Let us take some time to explain the landmarks and orderings found in *Logistics*, in comparison to other transportation domains. The goal in transportation domains is always a set of *at*-relations for transportable objects (sometimes also for vehicles). To achieve the *at* goal for an object, we must unload it from some vehicle, for which the object must be *in* the vehicle. If there is only a single vehicle, like in *Ferry* and *Miconic-STRIPS*, then

%solved						
FFv1.0	15	20	0	0	0	0
FFv1.0+L	100	100	100	100	100	100
FFv2.3	20	20	10	15	0	0
FFv2.3+L	100	100	100	100	100	100
LPG	100	80	90	65	55	35
LPG+L	100	100	80	70	60	35
time						
FFv1.0	238.11	267.20	-	-	-	-
FFv1.0+L	34.29	39.30	48.34	57.51	72.25	93.25
FFv2.3	207.11	233.54	226.60	228.82	-	-
FFv2.3+L	65.96	74.21	89.94	89.25	116.87	140.11
LPG	124.05	170.75	189.95	204.73	246.80	202.65
LPG+L	59.08	66.21	74.44	85.49	104.57	131.33
length						
FFv1.0	351.00	351.00	-	-	-	-
FFv1.0+L	429.00	439.00	466.60	484.65	504.75	523.90
FFv2.3	339.25	368.00	377.00	385.67	-	-
FFv2.3+L	424.75	449.75	469.00	480.67	506.50	526.50
LPG	447.80	468.31	483.50	500.00	517.88	547.50
LPG+L	468.95	484.38	508.36	522.89	556.88	568.00

Table 6: Experimental Results in *Logistics*.

the *in*-relation for the object is a shared precondition and is thus detected as a landmark, with a necessary order before the *at*-goal. If there are several vehicles, like in *Driverlog*, *Logistics*, *Mprime*, *Mystery*, and *Zenotravel*, then our lookahead technique, as described in Section 4.3, comes into the play: there is no shared precondition to the unload actions, but the *in*-relations for all vehicles form an intermediate fact set for which all earliest achievers (loading actions, namely) in the RPG share the "earliest loading-*at*-relation of the object". Now, for all the listed domains except *Logistics* the earliest possible loading-*at*-relation of the object simply is its initial location; in *Logistics*, for actions that load the object onto a plane, the earliest possible loading-*at*-relation is the airport of the object's (package's) origin city. So in *Logistics* we get a useful $\leq^a_{ln}$ lookahead order between the object being *at* its origin and its destination *airport*, while in the other cases we get a non-useful lookahead order between the object being *at* its initial and goal *location*. (A similar observation can be made in *Gripper* where there are several gripper hands.) Overall, the landmarks tree we get in *Logistics* is, for each package *p*, a sequence of the form *at(p initial-location)* $\leq^a_{gn}$ *in(p origin-city-truck)* $\leq^a_{gn}$ *at(p origin-city-airport)* $\leq^a_{ln}$ *at(p destination-city-airport)* $\leq^a_{gn}$ *in(p destination-city-truck)* $\leq^a_{gn}$ *at(p goal-location)*.[15] With our disjunctive landmarks control, the decomposition we get divides the package transportation into very small steps and thus helps to improve runtime. The point where we get longer plans is probably the $\leq^a_{ln}$ order between the package's origin and destination airports: posing the *at*-relations at destination airports as a disjunctive goal, the planner transports the packages between cities one-by-one, without trying to use the same plane for several packages. (As we indicated in Section 5.3, this increase in plan length can be avoided, at the cost of longer runtimes, by posing the

15. Note here that there is just one truck in each city in the standard *Logistics* domain. With several trucks in a single city, we would get a sequence of $\leq^a_{ln}$ orderings between the package's initial location, origin city airport, destination city airport, and goal location.

%solved						
FFv1.0	45	20	10	5	5	0
FFv1.0+L	100	100	100	100	75	0
FFv2.3	45	30	20	20	0	0
FFv2.3+L	100	100	100	100	100	100
LPG	100	100	100	100	100	35
LPG+L	95	95	100	100	85	65
time						
FFv1.0	183.39	235.81	158.33	162.55	280.89	-
FFv1.0+L	7.32	8.90	8.86	9.32	11.36	-
FFv2.3	159.88	237.87	209.92	225.01	-	-
FFv2.3+L	7.89	9.11	9.02	11.01	16.71	20.20
LPG	39.94	45.83	73.75	86.72	88.19	105.96
LPG+L	28.43	31.60	46.74	53.87	57.39	68.29
length						
FFv1.0	153.44	176.75	147.00	133.00	182.00	-
FFv1.0+L	164.00	195.25	155.00	141.00	206.00	-
FFv2.3	167.33	185.00	166.50	190.00	-	-
FFv2.3+L	183.56	201.50	177.75	207.25	279.10	286.30
LPG	240.42	261.26	269.25	261.75	294.82	290.29
LPG+L	228.89	250.37	258.50	246.60	276.65	269.71

Table 7: Experimental Results in *Rovers*.

leaf landmarks as a DNF goal of maximal consistent subsets – all *at*-relations at destination airports are consistent and the DNF we get is a single conjunction.)

6.7 Rovers

The *Rovers* domain was introduced by Long and Fox in the AIPS-2002 planning competition. The problem is to find a plan for an interplanetary rovers mission. A set of rovers can navigate along road maps of way points (i.e. the viable road map can be different for different rovers), take soil or rock samples, take images, and communicate their observations back to landers. The goal is to gather together a set of observations. There are 9 operators to navigate the rovers, to take soil/rock samples, to drop such samples (they occupy storage space), to calibrate cameras, to take images, and to communicate soil/rock/image data to landers. We generated our random instances with the software provided by Long and Fox. This software takes as inputs the number of rovers, the number of waypoints, the number of cameras, the number of "modes" (different types of images that can be taken), and the number of objectives (of which images can be taken). We scaled the instances via a parameter n. Each instance has n rovers, $25+n$ waypoints, n cameras, 5 modes, and $5+n$ objectives. Table 7 shows our data for the n values 25, 27, 29, 31, 33, and 35, 20 random instances per size.

We get various kinds of landmarks and orderings in *Rovers*. The goal is a set of *communicated-data*-relations, where the data can be rock or soil analysis data, or images. The actions that communicate such data from a rover to a lander all require, of course, that the rover has the respective piece of data, so we get *have-data* landmarks. For a rock or soil piece of data, the data can be gathered only by taking a sample at a specific waypoint *w*; if there is only a single rover *r* that is equipped for the right kind of analysis and that can navigate to *w* then we get *at(r w)* as a landmark. For taking an image of an objective o in a specific mode m, we need a camera c that supports m, on a rover r that can navigate to

a point w from which o is visible. If there is only one appropriate combination of c, r, and w, then we get *at(r w)* and *calibrated(c)* as landmarks. In that case, if there is only one appropriate calibration target for c, we also get the respective rover position as a landmark. Reasonable orderings arise, e.g., if there are several landmarks involving the position of the same rover. (E.g., if a camera on a rover r must be calibrated at a waypoint w, and be used to take an image at a waypoint w', then we get a reasonable order *at(r w)* $\leq_r^a$ *at(r w')*.)

The solution percentage/runtime performance of both FF versions improves dramatically when using our landmarks control, the plans become slightly longer. LPG's solution percentage performance is a bit inconclusive in the examples at hand, where LPG is not yet at its limit of scalability, so we ran further experiments. We generated larger instances, with n values 37, 39, 41, 43, and 45, 3 random examples per size, and ran LPG and LPG+L on these. The solution percentage values we got are (LPG/LPG+L) 0/100, 33.3/100, 0/100, 0/33.3, and 0/0. This suggests that LPG scales further up when using landmarks. In the single example solved by LPG in the larger test suite, LPG took 79.31 seconds to find a plan with 270 steps, while LPG+L took 101.13 seconds to find a plan with 249 steps. In the smaller examples, LPG's runtime in solved instances becomes better, while at the same time the found plans become shorter.

6.8 Tyreworld

The *Tyreworld* domain was first designed by Stuart Russel (Russell & Norvig, 1995). Flat wheels on a car must be exchanged with spare tyres, which involves a number of different working steps. Problem size scales with the number of flat wheels. There are 13 operators to open/close the boot, to fetch/put away something from/into the boot, to loosen/tighten nuts, to jack up/jack down hubs, to do up/undo nuts, to put on/remove wheels, and to inflate wheels. To exchange a single wheel, one has to open the boot, fetch the spare wheel and the pump and the jack and the wrench, inflate the spare wheel, loosen the nuts on the right hub, jack up the hub, undo the nuts, remove the flat wheel, put on the spare wheel, do up the nuts, jack down the hub, and tighten the nuts. The tools must be put back into the boot at the end. There is only a single *Tyreworld* instance per size, and we used the generator provided on the FF homepage to generate examples with up to 30 flat wheels.[16] The results are very easy to interpret, so we do not include a detailed table here. FFv1.0 can solve tasks with up to 22 wheels within our 300 second time limit. FFv1.0+L comfortably scales up to the task with 30 wheels, which it solves in 14.28 seconds. For FFv2.3, which easily scales up to the task with 30 wheels (solved in 7.57 seconds) landmarks do not bring a runtime improvement (indeed, the runtimes become somewhat worse). For LPG, which also easily scales up to 30 wheels (14.55 seconds), the runtime behaviour is similar. As for plan length, this becomes somewhat longer for FFv1.0 and LPG when using landmarks; for FFv2.3, plan length remains the same.

Let us consider the (sub-)goals and orderings in *Tyreworld* instances. Such instances have rather many top level goals. There are *in-boot* goals for all tools, and a *closed-boot* goal. For each flat wheel, we have an *on* goal for the respective spare wheel and hub, an

16. It certainly sounds funny to encode a problem involving a car with 30 wheels all flat at the same time. But of course, what we are interested in here is the underlying *structure* of the problem, which does not depend on the names given to the predicates and operators.

inflated goal for the spare wheel, a *tight* goal for the nuts on the hub, and an *in-boot* goal for the flat wheel. Reasonable orderings between these goals tell us that, first, we should inflate the spare wheels, then mount these on their respective hubs, then tighten the nuts, then put the flat wheels and the tools into the boot, then close the boot. We do get some landmarks and orderings beyond this, stating that at some point we must hold each spare wheel, and that at some point the hubs with flat wheels on must be made free. But this additional information is apparently not enough to make a significant difference over the Goal Agenda technique (which works with reasonable orderings on top level goals). This explains the behaviour of FFv1.0 and FFv2.3: remember that these planners are basically the same except that FFv2.3 uses the Goal Agenda. Using landmarks, FFv1.0 becomes roughly as efficient as FFv2.3, at the cost of longer plans. For FFv2.3, in contrast, nothing much changes by using landmarks on top of the Goal Agenda, except an additional overhead for the landmarks computation. For LPG, it seems that this planner is very efficient in *Tyreworld* even without any goal ordering techniques.

7. Discussion

Various attempts have been made in the past to detect goal ordering constraints and use them for speeding up planning. All of these approaches differ considerably from our work. We list the most closely related ones. Irani and Cheng (1987) order a goal B after a goal A if, roughly speaking, A must be achieved before B can be achieved. The same authors later extended their notions to *sets* of goals (Cheng & Irani, 1989). In difference to our work, only top-level goals are considered. The same holds true for the work of Hüllem et al. (1999) who define a number of different ordering relations between top-level goals, which they approximate using (non-instantiated) operator graphs (Smith & Peot, 1993). A similar idea is explored in the PRECEDE system of McCluskey and Porteous (1997). They look at identifying necessary orders between top level goals but the focus of their approach is different: they develop a method for the construction of planning domain models and, within this, orders are identified *once* only for a given domain (rather than being problem instance specific) and are then exploited both for further validation of the domain model and also at planning time. Knoblock (Knoblock, 1994) describes the ALPINE system, which *does* consider problem sub-goals to compute an "abstraction hierarchy" for the Prodigy planner (Fink & Veloso, 1994). In ALPINE, a sub-goal A is ordered before a sub-goal B if A must *possibly* be achieved first in order to achieve B. This is different from our work where orderings of this kind are inserted when A must *necessarily* be achieved first in order to achieve B (also, the way these orderings are used to structure the search is very different). Further, in difference to our approach, ALPINE does not consider the possible *negative* interactions between sub-goals, which we use to extract reasonable orderings.

Koehler and Hoffmann (2000) provide the formal notion of reasonable orderings between top-level goals, which they approximate and use for (potentially) splitting a planning task into several smaller sub-tasks. We extend this approach by the new dimension of necessary sub-goals, landmarks. By doing so, we obtain (potentially) more ordering information, e.g. in domains such as *Grid*, *Freecell*, *Logistics*, and *Rovers*, where the approach previously could do nothing to structure the search (because the achievement of one top level goal does not, or only very subtly, affect the achievement of another one). We provide a natural

formal extension of reasonable orderings to landmarks, clarify the computational complexity of our notions, and describe domain-independent and planner-independent algorithmic methods to approximate our definitions and to exploit the resulting ordering information. Our experiments show that our technology often speeds FF up significantly, sometimes dramatically, even when already using Koehler and Hoffmann's goal ordering techniques. A preliminary implementation of landmarks to control LPG shows that runtime gains are obtainable for this planner, too.

Various interesting research questions are open. For example, it is unclear to us why our landmarks technique often has less runtime impact on LPG than on FF; it ought to be investigated how our techniques can be better integrated with the LPG code. It might also be that, overall, there are alternative ways of using landmarks (e.g. as a plan initialisation technique rather than as a disjunctive search control framework) that work better for LPG. As another open topic, as said before our current technology removes, prior to planning, all cycles in the ordering relations. Instead, it could be investigated if these cycles can contain useful structural information, and how to exploit this information. It also remains open how to lift our techniques to more powerful planning paradigms, in particular ADL, numeric, and temporal planning. As for ADL, lifting our definitions is probably straightforward, but extending the approximation techniques (e.g. the computation of "shared precondition facts" when preconditions are complex formulas) seems difficult. It probably makes more sense to look at an extension of STRIPS with conditional action effects only (planners such as FF (Hoffmann & Nebel, 2001) and IPP (Koehler et al., 1997) compile full-scale ADL down into such a format prior to planning). As for numeric planning, we believe that our definitions and approximation techniques can be naturally extended based on the "monotonicity" concept introduced with Metric-FF by Hoffmann (2003). Such techniques are likely to be useful in domains like *Settlers* (Long & Fox, 2003) where the numeric variables carry most of the information of what is needed to solve the task. In temporal planning, it seems our techniques could, in principle, be used with no extensions whatsoever (except adjusting them to the new syntax). It might be possible, however, to enrich the landmarks graph (the LGT) with information about the *time* that is at least needed to get from one landmark (a shared precondition, e.g.) to another one. Such additional information can probably play a useful role in the temporal planning process.

Finally, staying within the STRIPS setting, there are two interesting topics opened up by our preliminary handling of "lookahead necessary" orderings. Currently we order $L \leq_{ln} L'$ if L' is a landmark, and there is an intermediate fact set $\{L_1, \ldots, L_m\}$ such that all achievers of L' have (at least) one fact L_j as precondition, and all achievers of any of the facts L_j share the precondition L. Now:

1. The intermediate set $\{L_1, \ldots, L_m\}$ we use is a *disjunctive landmark* in the sense that $L_1 \vee \ldots \vee L_m$ must be true at some point on every solution plan. While currently we use $\{L_1, \ldots, L_m\}$ only to possibly find new landmarks L, disjunctive landmarks really are a generalisation of our approach. The landmarks we considered in this paper are the special case where the disjunctive landmarks are singleton sets. It remains to explore the general case. Interesting issues are to be resolved concerning the choice of disjunction candidates, and the definition of, e.g., reasonable orderings.

2. While currently we look only one step ahead, in general one can look any number k of steps ahead: find k intermediate sets by backchaining over possible achievers, such that at the kth level there is a shared precondition. Obviously, doing so it would be critical to find good heuristics for selecting intermediate fact sets, and to trade the effort invested for the extended backchaining against the value of the obtained ordering information.

Acknowledgments

The authors wish to thank Maria Fox and Derek Long for providing the TIM API. We are also very thankful to Alfonso Gerevini and Ivan Serina for their help with running/modifying the LPG system. We finally thank Malte Helmert for pointing out some NP-reductions. This article is an extended and revised version of a paper (Porteous, Sebastia, & Hoffmann, 2001) that was published at ECP-01.

Appendix A. Proofs

This appendix contains two kinds of proofs: first, statements about the computational complexity of deciding about our ordering relations; second, statements about our sufficient criteria for reasonable orderings (i.e., implications from necessary orderings and inconsistency to reasonable orderings). Each kind of proofs is presented in one subsection.

A.1 Computational Complexity

Theorem 1 *Let* LANDMARK *denote the following problem: given a planning task* (A, I, G)*, and a fact* L*; is* L *a landmark?*

Deciding LANDMARK *is PSPACE-complete.*

Proof: Hardness is proven by polynomially reducing the complement of PLANSAT– the decision problem of whether there exists a solution plan to a given arbitrary STRIPS planning task (Bylander, 1994) – to the problem of deciding LANDMARK. Given a planning task (A, I, G), obtain the modified action set A' by adding the three actions

$$\begin{array}{rrr} (pre, & add, & del) \\ (\{F\}, & I, & \{F\}), \\ (\{F\}, & \{L\}, & \{F\}), \\ (\{L\}, & G, & \emptyset) \end{array}$$

and obtain the modified initial and goal states by

$$I' := \{F\},\ G' := G$$

Here, F and L are new facts. In words, attach to (A, I, G) an artificial way of by-passing the task. On that by-pass, L must be added. L is a landmark in the modified task iff (A, I, G) is not solvable.

Membership follows because a fact is a landmark if and only if it is an initial- or goal-fact, or the task becomes unsolvable when ignoring all actions that add the fact. Deciding unsolvability – the complement of PLANSAT– is in PSPACE. ■

Theorem 2 *Let* NECESSARY-ORD *denote the following problem: given a planning task* (A, I, G)*, and two facts* L *and* L'*; does* $L \leq_n L'$ *hold?*

Deciding NECESSARY-ORD *is PSPACE-complete.*

Proof: Hardness is proven by polynomially reducing the complement of PLANSAT to the problem of deciding NECESSARY-ORD. Given a planning task (A, I, G), obtain the modified action set A' by adding the three actions

$$\begin{array}{rrr} (pre, & add, & del) \\ (\emptyset, & \{L\}, & \emptyset), \\ (\{L\}, & \{L'\}, & \emptyset) \\ (G, & \{L'\}, & \emptyset) \end{array}$$

and obtain the modified initial and goal states by

$$I' := I,\ G' := \{L, L'\}$$

Here, L and L' are new facts. The only chance of achieving L' without having L true a priori is when the original goal G has been achieved. Thus, there is the necessary ordering $L \leq_n L'$ if and only if the original task is unsolvable.

A non-deterministic algorithm that decides the complement of NECESSARY-ORD in polynomial space is the following. Iteratively guess actions, keeping in memory always only the current state and its predecessor. If n is the total number of different facts in (A, I, G), 2^n such guessing steps suffice to visit all reachable states (one can use a counter to keep track of the number of guessing steps already made). $L \leq_n L'$ does not hold if and only if L' is true initially, or there is a state that contains L' such that the state's predecessor does not contain L. Finished with NPSPACE = co-PSPACE = PSPACE. ■

Note that L and L' are landmarks in the hardness part to the proof of Theorem 2.

Theorem 3 *Let* GREEDY-NECESSARY-ORD *denote the following problem: given a planning task* (A, I, G)*, and two facts* L *and* L'*; does* $L \leq_{gn} L'$ *hold?*

Deciding GREEDY-NECESSARY-ORD *is PSPACE-complete.*

Proof: Hardness is proven by exactly the same argument as in the proof to Theorem 2. For membership, one can use the same algorithm except that now one must also remember whether or not L' has been true yet, and check only the predecessors of those states where L' is true the first time. ■

Theorem 4 *Let* AFTERMATH *denote the following problem: given a planning task* (A, I, G)*, and two facts* L *and* L'*; is* L' *in the aftermath of* L*?*

Deciding AFTERMATH *is PSPACE-complete.*

Proof: Hardness is proven by reduction of the complement of PLANSAT. Given a planning task (A, I, G), obtain the modified action set A' by adding the actions

$$\begin{array}{rrr} (pre, & add, & del) \\ (\{F\}, & \{L'\}, & \{F\}), \\ (\{L'\}, & \{L\}, & \{L'\}), \\ (\{L\}, & I, & \{L\}), \\ (\{L\}, & \{L'\}, & \emptyset), \\ (\{L, L'\}, & G, & \emptyset) \end{array}$$

and obtain the modified initial and goal states by

$$I' := \{F\},\ G' := G$$

Here, F, L, and L' are new facts. In words, what happens is this. The only state in $s \in S_{(L', \neg L)}$ is the one that results from applying the first new action to the initial state. One must then add L. Afterwards, there is the choice to either solve the original task, or achieve L' and solve the task using the last of the new actions. There is a plan from s on which L' is not true at some point after L if and only if the original task is solvable.

A non-deterministic algorithm that decides the complement of AFTERMATH in polynomial space is the following. Iteratively guess 2^n actions, where n is the total number of different facts in (A, I, G). L' is not in the aftermath of L if and only if there is an action sequence that contains a state $s \in S_{(L', \neg L)}$ and that achieves the goal, such that either L is not true between s and the first goal state, or L' is not true between L and the first goal state (checking $s \in S_{(L', \neg L)}$ and points where L, L', or the goal are true the first time can be done by keeping appropriate flags). Finished with NPSPACE = co-PSPACE = PSPACE. ■

Theorem 5 *Let* REASONABLE-ORD *denote the following problem: given a planning task (A, I, G), and two facts L and L' such that L' is in the aftermath of L; does $L \leq_r L'$ hold?*

Deciding REASONABLE-ORD *is PSPACE-complete.*

Proof: The hardness proof is exactly the same that Koehler and Hoffmann (Koehler & Hoffmann, 2000) use for proving PSPACE-hardness of deciding about reasonable orderings, proceeding by a polynomial reduction of (the complement of) PLANSAT. Given an arbitrary planning task (A, I, G), introduce new atoms L, L', F and F' and obtain the modified action set A' by adding the three actions

$$\begin{array}{rrr} (pre, & add, & del) \\ (\{F\}, & \{L', F'\}, & \{F\}), \\ (\{L', F'\}, & I, & \{F'\}), \\ (G, & \{L\}, & \emptyset) \end{array}$$

and modify the initial and goal states by

$$I' := \{F\},\ G' := G \cup \{L, L'\}$$

Here, F, F', L, and L' are new facts. L and L' are goals in the modified task so in particular L' is in the aftermath of L. In the modified task, $S_{(L',\neg L)}$ contains the single state $\{L', F'\}$. From there, all plans that achieve L delete L' if and only if there is no such plan, i.e., if and only if the original task is unsolvable.

A non-deterministic algorithm that decides the complement of REASONABLE-ORD in polynomial space is the following. Iteratively guess 2^n actions, where n is the total number of different facts in (A, I, G), and check whether there is an action sequence that contains a state $s \in S_{(L',\neg L)}$, such that, after s, L' does not get deleted before L gets achieved (the necessary information about the sequence can be stored in flags). Finished with NPSPACE = co-PSPACE = PSPACE. ∎

A.2 Sufficient Criteria

Lemma 1 *Given a planning task (A, I, G), and two landmarks L and L'. If either*

1. *$L' \in G$, or*
2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq_{gn} L_{i+1}$ for $1 \leq i \leq n$, and $L' \leq_{gn} L_{n+1}$,*

then L' is in the aftermath of L.

Proof: The case where $L' \in G$ is trivial. We prove the other case. Let $s \in S_{(L',\neg L)}$, $s = \mathit{Result}(I, P)$, be a state where L' has been achieved strictly before L on the action sequence P. As $L = L_1$ has never been true on the path P from the initial state to s, neither has L_2 been true, or we have a contradiction to $L_1 \leq_{gn} L_2$: L_2 is not true initially so it would have to be made true the first time at some point; immediately before this point, L_1 would need to be true. The same argument applies iteratively upwards to all pairs L_i and L_{i+1}, implying that none of $L_1, \ldots, L_n$ have been true yet on the path to s. Likewise, L_{n+1} can not have been true yet on the path to s.

We first prove that L_n and L' must be true together at some point on any solution plan from s. As L_{n+1} is a landmark that has not yet been true on the path to s, any solution plan $P' = \langle a_1, \ldots, a_m \rangle$ from s, $G \subseteq \mathit{Result}(s, P')$, fulfills $L_{n+1} \in \mathit{Result}(s, \langle a_1, \ldots, a_j \rangle)$ for some $1 \leq j \leq m$. Assume j is the lowest such index. Then L_{n+1} is made true the first time at the end of the path $P \circ \langle a_1, \ldots, a_j \rangle$. Therefore, with $L_n \leq_{gn} L_{n+1}$, $L' \leq_{gn} L_{n+1}$, and $L_{n+1} \notin s$, we have $j \geq 1$ and $L_n, L' \in \mathit{Result}(s, \langle a_1, \ldots, a_{j-1} \rangle)$.

We now arrive at our desired property by a simple backward chaining on the sequence $L_1, \ldots, L_n$. We know that L_n and L' are true together on some point $j-1$ on every solution plan $P' = \langle a_1, \ldots, a_m \rangle$ from s. Knowing that L_n has not been true on the path from the initial state to s, there must be a point $j' \leq j-1$ in P' such that L_n is true the first time at the end of the path $P \circ \langle a_1, \ldots, a_{j'} \rangle$. But then, with $L_{n-1} \leq_{gn} L_n$, and $L_n \notin s$, we have $j' \geq 1$ and $L_{n-1} \in \mathit{Result}(s, \langle a_1, \ldots, a_{j'-1} \rangle)$. We know that L_{n-1} has not been true on the path from the initial state to s, so the same argument can be applied downwards to L_1, giving us

$L_1 = L \in Result(s, \langle a_1, \ldots, a_{j''-1} \rangle)$ for some $j'' < j'$. With $L' \in Result(s, \langle a_1, \ldots, a_{j-1} \rangle)$, and $j'' - 1 < j - 1$, we have that L' is in the aftermath of $L = L_1$. ■

Theorem 6 *Given a planning task (A, I, G), and two landmarks L and L'. If L interferes with L', and either*

1. *$L' \in G$, or*

2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq_{gn} L_{i+1}$ for $1 \leq i \leq n$, and $L' \leq_{gn} L_{n+1}$,*

then there is a reasonable ordering between L and L', $L \leq_r L'$.

Proof: By Lemma 1, we know that L' is in the aftermath of L. Say $s \in S_{(L', \neg L)}$. We need to prove that any action sequence achieving L from s deletes L'. We need to cover four reasons for interference.

1. L is inconsistent with L': with inconsistency, from *any* reachable state were L' is true we must delete L' in order to achieve L.

2. there is a fact $x \in \bigcap_{a \in A, L \in add(a)} add(a)$ such that x is inconsistent with L': x will necessarily be true in the state after we achieved L, so L' must be deleted by some action on the way to that state.

3. $L' \in \bigcap_{a \in A, L \in add(a)} del(a)$: obvious, the action adding L will delete L'.

4. Say there is an x that is inconsistent with L', and $x \leq_{gn} L$. We know that s is reachable, $s = Result(I, P)$, $L' \in s$, and L is not true at any point on the path to s. As $L' \in s$, we have $x \notin s$ with inconsistency. Say $P' = \langle a_1, \ldots, a_n \rangle \in A^*$ is an action sequence such that $L \in Result(s, P')$, i.e., $L \in Result(s, \langle a_1, \ldots, a_i \rangle)$ such that i is minimal. Because L is achieved the first time on the sequence $P \circ \langle a_1, \ldots, a_i \rangle$, we can apply the greedy necessary ordering $x \leq_{gn} L$, so we know that $x \in Result(I, P \circ \langle a_1, \ldots, a_{i-1} \rangle)$ ($i \geq 1$ because $L \notin s$). With inconsistency, $L' \notin Result(I, P \circ \langle a_1, \ldots, a_{i-1} \rangle)$, so $i - 1 > 0$ and $L' \in del(a_j)$ for some $1 \leq j \leq i - 1$.

■

Lemma 2 *Given a planning task (A, I, G), a set O of ordering constraints, and two landmarks L and L'. If either*

1. *$L' \in G$, or*

2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq L_{i+1}$ is in O for $1 \leq i \leq n$, $L' \leq_{gn} L_{n+1}$, and for at least one $1 \leq i \leq n$ there is no action $a \in A$ with $L_i \in add(a)$ and $L_{i+1} \in add(a)$,*

then L' is in the obedient aftermath of L.

Proof: The case where $L' \in G$ is trivial. We prove the other case. Let $s \in S^O_{(L',\neg L)}$ be a state where L' has been achieved strictly before L; let P be an obedient (in I) path from the initial state to s on which L has not yet been true, and L' has just been achieved. As $L = L_1$ has never been true on the path, neither has L_2 been true, or we have a contradiction to the constraint $L_1 \leq L_2$. The same argument applies iteratively upwards to all pairs L_i and L_{i+1}, implying that none of $L_1, \ldots, L_n$ have been true yet on the path to s. Likewise, L_{n+1} can not have been true yet on the path to s if the second case of the prerequisite holds. In particular, none of these facts is contained in s itself.

Now, we first observe that, on any solution plan P' from s, L' is true one step before L_{n+1} is true the first time. This simply follows from the greedy necessary ordering between the two facts. Say $P' = \langle a_1, \ldots, a_m \rangle$ is a solution plan from s, $G \subseteq \mathit{Result}(s, P')$, such that P' is obedient in s. As L_{n+1} is a landmark, and we have $L_{n+1} \notin s$, P' fulfills $L_{n+1} \in \mathit{Result}(s, \langle a_1, \ldots, a_j \rangle)$ for some $1 \leq j \leq n$. Assume j is the lowest such index. Then L_{n+1} is made true the first time at the end of the path $P \circ \langle a_1, \ldots, a_j \rangle$. Therefore, with $L' \leq_{gn} L_{n+1}$ and $L_{n+1} \notin s$, we have $j \geq 1$ and $L' \in \mathit{Result}(s, \langle a_1, \ldots, a_{j-1} \rangle)$. So the first occurrence of L_{n+1} is at a point j while an occurrence of L' is at $j - 1$. Now, we can do a backward chaining on the sequence $L_1, \ldots, L_{n+1}$. P' is obedient in s and none of the facts is true in s. So the facts must be achieved at points $j_1, \ldots, j_{n+1} = j$ such that $j_i \leq j_{i+1}$ for $1 \leq i \leq n$. Here we make use of the second part of the prerequisite: there is at least one $1 \leq i \leq n$ such that there is no action a with $L_i \in \mathit{add}(a)$ and $L_{i+1} \in \mathit{add}(a)$. This implies that, at this point, the ordering relation between j_i and j_{i+1} is strict, $j_i < j_{i+1}$: the two facts can not be achieved simultaneously so L_i must be achieved strictly before L_{i+1} to obey the constraint $L_i \leq L_{i+1}$. Thus we got $j_1 < j$ where j_1 is the first occurrence of $L = L_1$ and $j - 1$ is an occurrence of L'. This concludes the argumentation. ■

Theorem 7 *Given a planning task (A, I, G), a set O of ordering constraints, and two landmarks L and L'. If L interferes with L', and either*

1. *$L' \in G$, or*
2. *there are landmarks $L = L_1, \ldots, L_{n+1}$ such that $L_i \leq L_{i+1}$ is in O for $1 \leq i \leq n$, $L' \leq_{gn} L_{n+1}$, and for at least one $1 \leq i \leq n$ there is no action $a \in A$ with $L_i \in \mathit{add}(a)$ and $L_{i+1} \in \mathit{add}(a)$,*

then there is an obedient reasonable ordering between L and L', $L \leq^O_r L'$.

Proof: By Lemma 2, we know that L' is in the obedient aftermath of L. Say $s \in S^O_{(L',\neg L)}$. We need to prove that any action sequence achieving L from s deletes L'. For the four reasons for interference, this can be proven with exactly the same arguments as in the proof to Theorem 6. ■

Appendix B. Experimental LPG+L Implementation

In this appendix we discuss our experimental implementation of landmarks control around LPG. The implementation is preliminary in that it does not integrate landmarks control

directly with LPG's code, but rather implements a simple control loop around an LPG executable. This causes a large runtime overhead due to the repetition of LPG's pre-processing machinery that computes fact (and action) inconsistencies. Our approach is to simply ignore this unnecessary overhead. This way, our experimental code produces the same results as a direct implementation that would do the pre-process only once – *provided LPG's pre-process computes the same information throughout the control loop.* We show that the latter is indeed the case in all our 8 testing domains except in *Freecell.* As explained in Section 6 already, because of these results in *Freecell* we measure total running time of our implementation, while in the other 7 domains we count the runtime for only the first one of LPG's inconsistency pre-processes.

Let us first discuss LPG's overall architecture. As far as relevant for us here, it proceeds as follows:

1. Perform a pre-process that detects inconsistent facts and actions.

2. Build a planning graph where the mutex reasoning is replaced by inserting the pre-computed inconsistencies.

3. Perform a local search in a space of sub-graphs of the resulting planning graph.

We need a little more detail about steps 1 and 2. We summarise the algorithms described by Gerevini et al. (2003). Step 1 first computes inconsistent pairs of facts. The algorithm keeps track of sets F and A of reached facts and actions, and of a set M of potential inconsistencies (persistent mutex relations, in LPG's terminology). F is initialised with the initial state of the task that LPG is run on, A and M are initialised with the empty set. Then a fixpoint procedure iterates until there are no more changes to F and M. In each iteration, all actions are considered whose preconditions are in F without an inconsistency in M. Such actions are used to identify new potential inconsistencies (e.g., between the action's add and delete effects). Potential inconsistencies are removed when there are actions in A that either add both facts, or add one fact without deleting the other. When the fixpoint is reached, the fact pairs in M are proved to be inconsistent. The inconsistent actions are then computed as those that either interfere (delete each others add effects or preconditions), or have competing needs (inconsistent preconditions). Step 2 in LPG's architecture then builds a planning graph without mutex reasoning. Starting from the initial state, fact layers F_i and action layers A_i are built alternatingly, where A_i contains all actions whose preconditions are in F_i, and F_{i+1} contains all add effects of the actions in A_i. The process stops with a layer where no new facts come in, and the mutex pairs in the resulting graph are the inconsistent pairs computed by step 1.

As said, our experimental implementation repeatedly calls an LPG executable inside the landmarks control. So steps 1 and 2 are done over again, producing a large runtime overhead. In an implementation integrating the landmarks control directly into LPG's code, one could avoid much of this overhead by using an overall architecture that computes the inconsistencies only once, and (only) does steps 2 and 3 of LPG's architecture for every sub-task in the control loop. We henceforth refer to such a hypothetical implementation as *direct-LPG+L*, and to our experimental implementation as *LPG+L*. The idea is to simulate direct-LPG+L's behaviour by running LPG+L, and counting the runtime taken by LPG in

step 1 only for the first iteration of the control loop.[17] The subtlety to be taken care of is that the result of step 1 depends on the start (initial) state that LPG is run on. Remember that the search control loop in LPG+L calls LPG on a sequence of consecutive start states s, where the first s is the initial state, and each successor start state s' results from the previous start state s by applying the plan that LPG found for the last sub-task. Now, different found inconsistencies (for different start states) in step 1 can result in different planning graphs at the end of step 2, and different planning graphs can result in different search behaviour in step 3. So we can only safely simulate direct-LPG+L in cases where the planning graph resulting from step 2 is always the same in LPG+L as it would be in direct-LPG+L. We found that the latter is indeed the case in 7 of our 8 testing domains. The reason for this is that state reachability is (largely, in some cases) invertible in these domains.

Consider a state s' that is reachable from a state s. Denote by $F(s)$, $A(s)$ and $M(s)$ the sets F, A, and M at the end of LPG's fact inconsistency fixpoint computation when started with initial state s, similarly for s'. Then $F(s') \subseteq F(s)$, $A(s') \subseteq A(s)$, and $M(s') \supseteq M(s) \cap (F(s') \times F(s'))$ hold. The reason is simply that everything that is reachable from s' is also reachable from s. The fixpoint process from s will eventually "execute" the action sequence that leads from s to s', i.e., include these actions into the set A. At this point, s' will be contained in the set F, with no potential inconsistencies. It follows that every fact or action that the process reaches when started in s' will also be reached when started in s. It also follows that every potential inconsistency the process removes from M when started in s' will also be removed when started in s. This proves the claim. (Actually, $M(s') \supseteq M(s) \cap (F(s') \times F(s'))$ already follows because every fact pair that is inconsistent below s must also be inconsistent below s'. We include the longer proof sketch to provide a better background for the proof arguments below.)

If LPG's step 1 finds the same fact inconsistencies for two states, then it trivially follows that the action inconsistencies identified will also be the same. Thus, with the above, if states s and s' are both reachable from each other, then the outcome of step 1 is the same for both states. This immediately shows that in invertible domains – domains where to each action a there is an inverse action $\overline{a}$ that undoes exactly a's effects – the outcome of LPG's step 1 remains the same throughout our landmarks control loop. *Blocksworld-arm*, *Blocksworld-no-arm*, *Depots*, and *Logistics* are all invertible.

The other three of our experimental domains where we can safely ignore the overhead caused by step 1 repetition are *Grid*, *Rovers*, and *Tyreworld*. For each of these domains, we show that the outcome of step 2 in direct-LPG+L is the same as the outcome of step 2 in LPG+L, for every iteration of the landmarks control loop. Let us start with the *Tyreworld*. Consider the information computed by LPG's steps 1 and 2 in states s and s', where s' is reachable from s. *Tyreworld* is not invertible only because one can not "deflate" spare wheels that one has already inflated. From s' one can get to a state s'' that subsumes s except for, possibly, a set of *not-inflated(?spare-wheel)* facts. Such facts serve only as precondition for the actions that inflate *?spare-wheel*, which actions only achieve

17. The LPG implementation interleaves the computations for step 1 and step 2, which is why the individual runtimes can not be separated, and our LPG+L implementation in fact ignores the overhead for both step 1 *and* step 2 except in the first iteration of the control loop. The time taken for step 2 is typically not significant. See also the discussion at the end of this appendix.

inflated(?spare-wheel). These *inflated(?spare-wheel)* facts are already contained in s'. So the fact inconsistency fixpoint process can reach everything from s' that it can reach from s, except the inflate *?spare-wheel* actions. It follows that the only difference of the outcome of step 1 in s' compared to its outcome in s is that from s there can be inconsistencies involving the inflate *?spare-wheel* actions. But when step 2 is done in s', these actions are never reached anyway as their *not-inflated(?spare-wheel)* preconditions are not reachable. So steps 2 in both LPG+L and direct-LPG+L end up with the same planning graphs for the state s'.

A similar argument as in *Tyreworld* applies to states s and s' in *Grid* tasks. The only thing that can not be undone in *Grid* tasks is the opening of locked locations. From s' one can get to a state s'' that subsumes s except, possibly, a set of *locked(?location)* facts. Such facts serve only as precondition for actions that unlock *?location*, which actions only achieve *unlocked(?location)*. These *unlocked(?location)* facts are already contained in s', and the only difference of the outcome of step 1 in s' compared to its outcome in s is that from s there can be inconsistencies involving actions that unlock *?location*. When direct-LPG+L does step 2 in s', these actions are never reached anyway.

Finally, let's consider the *Rovers* domain. When reaching a state s' from a state s in *Rovers*, one can in general not get back to s, because once a soil or rock sample has been taken, it can not be put back to the waypoint it's been taken from. That is, from s' one can get to a state s'' that subsumes s except, possibly, a set of *at-soil/rock-sample* facts.[18] This, in turn, can only affect the reachability of *sample-soil/rock* actions, *have-soil/rock-sample* facts, *full-storage-space* facts, *communicate-soil/rock-data* actions, and *communicated-rock/soil-data* facts. So the only difference of the outcome of step 1 in s' compared to its outcome in s is that from s there can be inconsistencies involving these facts and actions. But when direct-LPG+L does step 2 in s', these facts and actions are never reached anyway.

We finally remark the following. The LPG implementation interleaves the computations for step 1 and step 2, and LPG outputs the runtime taken by step 1 *together with* step 2. So our LPG+L implementation in fact ignores the overhead for both step 1 *and* step 2 except in the first iteration of the control loop. In personal communication, Alfonso Gerevini and Ivan Serina informed us that step 2 typically takes only a small fraction (around 5%) of the time spent in pre-processing – which seems reasonable considering that step 2 is basically the same as building a relaxed planning graph, i.e., a single heuristic evaluation in, e.g., FF. Nevertheless, we reiterate that our LPG+L implementation is preliminary, and that the runtime results should be treated with care. In a direct implementation of landmarks in LPG, step 2 would have to be done for every iteration of the control loop, which runtime overhead we could not count due to the above implementational difficulties. On the other hand, in a direct implementation one could exploit various ideas regarding the computation of inconsistencies for different start states. Alfonso Gerevini suggested that, e.g., one could modify the inconsistency computation to only take account of the *changes* in a new state. One could also derive state invariants, like DISCOPLAN (Gerevini & Schubert, 2000) does, that are computed only once (independently of the initial state), and then pruned accordingly every time a new start state comes up (pruning amounts to

18. Taking an image deletes the calibration of the camera; but one can always re-calibrate the camera with the same calibration target as was used before – no camera is calibrated initially.

removing those invariants that do not hold given a particular initial state). Finally, one could try to replace LPG's inconsistency reasoning with the TIM inconsistency function, thereby avoiding step 1 altogether. As we said before, exploring such possibilities is a topic for future work.

References

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1-2), 279–298.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*(1–2), 165–204.

Cheng, J., & Irani, K. B. (1989). Ordering problem subgoals. In Sridharan, N. S. (Ed.), *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 931–936 Detroit, MI. Morgan Kaufmann.

Fikes, R. E., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fink, E., & Veloso, M. (1994). Prodigy planning algorithm. Technical report CMU-94-123, Carnegie Mellon University.

Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, *9*, 367–421.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

Gazen, B. C., & Knoblock, C. (1997). Combining the expressiveness of UCPOP with the efficiency of Graphplan.. In Steel, & Alami (Steel & Alami, 1997), pp. 221–233.

Gerevini, A., & Schubert, L. (2000). Inferring state constraints in DISCOPLAN: Some new results. In *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI-00)*, pp. 761–767 Austin, TX. MIT Press.

Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*. Special issue on the 3rd International Planning Competition, to appear.

Hoffmann, J. (2000). A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS-00)*, pp. 216–227. Springer-Verlag.

Hoffmann, J. (2002). Local search topology in planning benchmarks: A theoretical analysis. In Ghallab, M., Hertzberg, J., & Traverso, P. (Eds.), *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, pp. 92–100 Toulouse, France. Morgan Kaufmann.

Hoffmann, J. (2003). The Metric-FF planning system: Translating "ignoring delete lists" to numerical state variables. *Journal of Artificial Intelligence Research*. Special issue on the 3rd International Planning Competition, to appear.

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

Hüllen, J., Muñoz-Avila, H., & Weberskirch, F. (1999). Extracting goal orderings to improve partial-order and Graphplan-based planning. Technical report, University of Kaiserslautern.

Irani, K. B., & Cheng, J. (1987). Subgoal ordering and goal augmentation for heuristic problem solving. In McDermott, J. (Ed.), *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 1018–1024 Milan, Italy. Morgan Kaufmann.

Knoblock, C. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, *68*(2), 243–302.

Koehler, J. (1998). Solving complex planning tasks through extraction of subproblems. In Simmons, R., Veloso, M., & Smith, S. (Eds.), *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pp. 62–69. AAAI Press, Menlo Park.

Koehler, J., & 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.

Koehler, J., Nebel, B., Hoffmann, J., & Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset.. In Steel, & Alami (Steel & Alami, 1997), pp. 273–285.

Long, D., & Fox, M. (2003). The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*. Special issue on the 3rd International Planning Competition, to appear.

McCluskey, T. L., & Porteous, J. M. (1997). Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, *95*(1), 1–65.

McDermott, D., et al. (1998). *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee.

Porteous, J., & Sebastia, L. (2000). Extracting and ordering landmarks for planning. In *Proceedings UK Planning and Scheduling SIG Workshop*.

Porteous, J., Sebastia, L., & Hoffmann, J. (2001). On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., & Borrajo, D. (Eds.), *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, pp. 37–48 Toledo, Spain. Springer-Verlag.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ.

Slaney, J., & Thiebaux, S. (2001). Blocks world revisited. *Artificial Intelligence*, *125*, 119–153.

Smith, D., & Peot, M. (1993). Postponing threats in partial-order planning. In *Proceedings of the 11th National Conference of the American Association for Artificial Intelligence (AAAI-93)*, pp. 500–506 Washington, DC. MIT Press.

Steel, S., & Alami, R. (Eds.). (1997). *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, Vol. 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France. Springer-Verlag.