Albert-Ludwigs-Universität, Institut für Informatik

Technical Report 89

Ignoring Irrelevant Facts and Operators in Plan Generation

Bernhard Nebel Yannis Dimopoulos
Jana Koehler

Institut für Informatik, Albert-Ludwigs-Universität
Am Flughafen 17, D-79110 Freiburg, Germany
E-mail: ⟨*last name*⟩@informatik.uni-freiburg.de

April 1997

Abstract

It is traditional wisdom that one should start from the goals when generating a plan in order to focus the plan generation process on potentially relevant actions. The GRAPHPLAN system, however, which is the most efficient planning system nowadays, builds a "planning graph" in a forward-chaining manner. Although this strategy seems to work well, it may possibly lead to problems if the planning task description contains irrelevant information. Although some irrelevant information can be filtered out by GRAPHPLAN, most cases of irrelevance are not noticed.

In this paper, we analyze the effects arising from "irrelevant" information to planning task descriptions for different types of planners. Based on that, we propose a family of heuristics that select relevant information by minimizing the number of initial facts that are used when approximating a plan by backchaining from the goals ignoring any conflicts. These heuristics, although not solution-preserving, turn out to be very useful for guiding the planning process, as shown by applying the heuristics to a large number of examples from the literature.

Contents

1 Introduction

It is traditional wisdom that one should start from the goals when generating a plan in order to focus the plan generation process on potentially relevant actions. The GRAPHPLAN system [Blum and Furst, 1997], however, which is the most efficient planning system nowadays, builds a "planning graph" in a forward-chaining manner, applying all actions that are possible. While GRAPHPLAN works well on most of the examples known from the literature, one might suspect that larger examples containing information that is irrelevant for a particular task lead to performance problems. In fact, irrelevant information can lead to huge planning graphs even when GRAPHPLAN's feature for filtering out irrelevant facts is used.

Although, at first sight, it might seem to be pathological to have task specifications that contain irrelevant information, this situation occurs naturally when one wants to handle larger domains with varying and diverse goals. For instance, in a robotics domain one may want the robot to transport things, to guide people, to clean rooms, etc. In this case, one wants a domain description with all the necessary operators and the static domain specification. For a given goal and initial state, most of the domain might be irrelevant, however. Further, even the toy examples from the literature sometimes contain some form of "irrelevant information," which, if removed, leads to better performance of the planning process. In particular, for McDermott's [1996] *Manhattan world* and for a number of examples used by Kautz and Selman [1996], our heuristics proved to be very effective, reducing GRAPHPLAN's planning time by orders of magnitude.

For traditional planning systems that perform a backward-chaining search from the goal to the initial state, irrelevance is supposedly not a problem because such planning systems consider only actions that are relevant for solving the planning problem. Planning systems that do forward-chaining from the initial state may, however, run into performance problems. They probably explore possibilities that can never contribute to achieve the goal. However, when comparing GRAPHPLAN with a traditional backward-chaining planner, such as UCPOP [Penberthy and Weld, 1992], GRAPHPLAN is so much faster than UCPOP that GRAPHPLAN's larger sensitivity to irrelevance does not matter much. Nevertheless, "irrelevant information" can be a serious problem for GRAPHPLAN.

In order to bring the best of the two worlds together, one might think of mixing the backward-chaining with the forward-chaining approach similar to the way bottom-up and top-down parsing is mixed in the Earley [1970] parsing algorithm. Although parsing for context-free languages is computationally quite different from planning, because the former is a polynomial-time problem and the latter is PSPACE-complete for the propositional case [Bylander, 1994], the general idea of guiding the bottom-up search by top-down expectations seems to be plausible for planning as well.

The basic idea for creating top-down expectations in a planning system is quite simple: do backward-chaining from the goals to the initial facts using the operators and ignoring any conflicts between operator applications. This creates an AND-OR tree, where the AND-nodes are the goals or the preconditions of an (instantiated) operator and the OR-nodes are single ground atoms which can be generated by different (instantiated) operators. However, how long should we grow this tree and what paths are relevant in this tree? Further, if we select paths in the AND-OR tree, what should we do with the nodes on the selected paths? Finally, how can we make sure that growing this tree does not result in an exponential explosion?

In the GRAPHPLAN system, these questions are answered as follows. Grow the tree as long as new ground atoms are produced, consider all paths as relevant and take all ground atoms on these paths as relevant. Finally, since a fixpoint computation is used instead of creating the tree explicitly, the computational costs are bounded polynomially in the number of ground atoms. While this is an efficient and safe strategy, since it does not exclude any solution, it is also very weak. In almost all cases we considered, it does not lead to any reduction in computational costs.

A heuristic guiding the planning process should be *efficient*, i.e., an "almost" polynomial-time algorithm and empirically fast, and *solution preserving*, i.e., it should not exclude possible solutions. Since determining relevant information in planning is usually as hard as planning itself, satisfying both requirements probably leads to quite weak heuristics as in the case of the GRAPHPLAN system. We propose a heuristic based on minimizing the use of initial facts when backchaining from the goals to the initial facts, which is similar to McDermott's *greedy regression graph* heuristic. Based on that, we determine the information which is most likely relevant for the planning process. This heuristic is not solution preserving, but it is computational very efficient and turns out to be quite effective for a large number of domains. Further, it provides a computationally cheap way to identify unsolvable planning tasks and for this reason is very useful in debugging planning task specifications.

The rest of the paper is structured as follows. The next section discusses the notion of "relevance" and "irrelevance" in plan generation. In Section 3, we then show the effects "irrelevant information" can have on different planning systems. A family of heuristics to determine potentially useful information from the description of a planning task is described in Section 4, and the empirical results of applying these heuristics to a large number of examples are given in Section 5. In Section 6 we discuss the results and conclude.

2 What is "Irrelevant" in Plan Generation?

Sometimes it is intuitively obvious that a planning task description contains irrelevant information. For example, if in a blocks-worlds planning task (formalized using STRIPS [Fikes and Nilsson, 1971]) the blocks can have a color, then the colors are irrelevant – provided the goal description does not mention colors or if there are no operators to paint blocks.

There are more subtle cases, however. For example, if a block can be painted while the robot holds it and if the goal description contains information that the color of the block should be the same as in the initial state. Of course, the painting operation is irrelevant. However, it is not obvious how to detect this in a domain-independent way.

An even more subtle case is a blocks-world planning task with a number of additional blocks sitting on the table which are not mentioned in the goal description. These blocks are clearly irrelevant for finding a plan of stacking the relevant blocks, but they are considered by plan generation systems, leading to serious performance problems.

Trying to make the notions of *relevance* and *irrelevance* more concrete, one notes that there are at least two levels on which we can discuss these notions. First of all, there is the *external level*. A planning task description may contain type information, initial facts, and operators that are not needed for a solution.[1] Secondly, we can consider the *internal level* of the planning system. Here, we can consider ground operators or ground facts as relevant or irrelevant. In the following, we refer to operators, type information, initial facts, other ground facts, and instantiated operators as **pieces of information**.

Regardless of the level and the type of information, one can distinguish between different degrees of irrelevance. First of all, a piece of information may never be part of any solution for a given planning task, i.e., it is **completely irrelevant**. Secondly, a piece of information may appear in some plans, but it is not necessary for generating a plan. We call this **solution irrelevance**. If plan length is an issue, one can define the notion of **solution-length irrelevance** in a similar way (provided the notion of *length* of a solution has been defined). Unfortunately, however, these "semantic" notions of irrelevance are computationally as hard as planning itself, at least for the case of propositional STRIPS, which is PSPACE-complete [Bylander, 1994; Bäckström, 1992].

Theorem 1 *For propositional* STRIPS*, deciding* complete irrelevance*,* solution irrelevance*, and* solution-length irrelevance *of a piece of information in a planning task description is* PSPACE*-complete under polynomial Turing-reductions.*

Proof sketch. Membership in PSPACE follows from the following simple facts. A fact or operator is *solution-irrelevant*, if its removal does not change the plan existence property, which is in PSPACE, i.e., two calls to an oracle deciding plan existence suffice to decide solution irrelevance. Further, a piece of information is *solution-length irrelevant* if its removal does not change the bounded plan existence property for minimal length plans, where the bound can be easily computed using binary search. Finally, a fact or operator is *completely irrelevant* if adding

[1] The goal description is, of course, never irrelevant!

something to the planning task that makes the fact or operator necessary for achieving the goals changes the plan existence property.

PSPACE-hardness of *solution irrelevance* and *solution-length irrelevance* follows, since by adding an operator or fact to a planning task description that must be used in any plan, we can decide plan existence and bounded plan existence of the original problem by deciding *solution irrelevance* or *solution-length irrelevance* of the new piece of information in the modified planning task.

PSPACE-hardness of *complete irrelevance* follows because plan existence can be decided by deciding complete irrelevance of all facts or operators. ■

Approaching the notion of irrelevance from a more syntactic point of view, we consider the process of *chaining* between ground facts. We say that a fact φ **generates** ψ iff there is an operator o that can be instantiated to a (type-consistent) ground operator g in a way such that φ is a precondition of g and ψ is an add-effect of g. Based on this definition we say that φ_0 is **reachable by backchaining** from φ_n iff there exists a sequence of facts $\varphi_0, \varphi_1, \ldots, \varphi_n$ such that φ_i generates φ_{i+1}. Similarly, φ_n is **reachable by forward-chaining** from φ_0 iff φ_0 is reachable by backchaining from φ_n.

Using these notions, one can distinguish between *goal irrelevance* and *initial-state irrelevance*. A piece of information is **goal irrelevant** if it cannot be reached by backchaining from the goals, and a piece of information is **initial-state irrelevant** if it cannot be reached by forward-chaining from the initial state. Both notions imply complete irrelevance and are easily computable. GRAPHPLAN can filter out both kinds of irrelevant information. Initial-state irrelevant information is filtered out in building the planning graph, goal irrelevant information can be filtered out by using an option that leads to computing the ground facts reachable by backchaining from the goals. However, as pointed out above, these notions are also very weak and do not cover the more subtle cases mentioned in the beginning of this section.

3 Empirical Effects of Irrelevance

As mentioned in the Introduction, one would expect that traditional backward-chaining planners will have less difficulties with solution-irrelevant pieces of information than the GRAPHPLAN system. In order to test this hypothesis, we set up two sets of modified blocks-world planning tasks.

In the first set, we added colors and a `paint` operation to the blocks world,[2] where the `paint` operation has three parameters, namely the block, its previous color, and its new color. Further, painting can only be performed if the robot

[2]We used the traditional formalization with `pick-up`, `put-down`, `stack`, and `unstack` operators, which is slightly less efficient for UCPOP than the formalization with only one three-parameter `move` operator.

holds the block. Finally, we specify in the initial conditions that all blocks are white and require in the goal description that they are still white. In the second set of examples, we took ordinary blocks-world planning tasks and added superfluous blocks that sit on the table and are clear. In both sets we varied the number of relevant blocks and the number of irrelevant details, i.e., colors and superfluous blocks.

The results[3] shown in Figure 1 seem to confirm the hypothesis that backward-chaining planners such as UCPOP are not as much affected as GRAPHPLAN by solution-irrelevant information.[4] The number of colors in a planning task description do not seem to affect the search process of UCPOP at all, while the colors seem to present a problem for GRAPHPLAN. Further, the effect of superfluous blocks for UCPOP is much less dramatic than for the GRAPHPLAN-system.

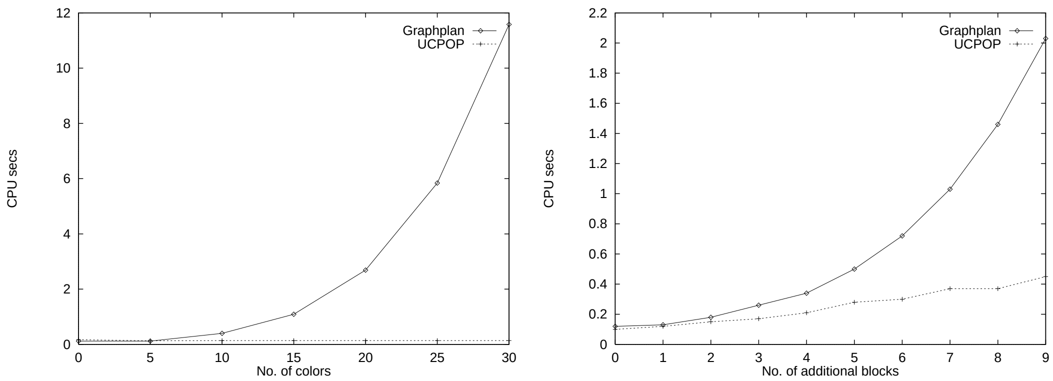


Figure 1: CPU time for stacking 2 blocks when colors or other blocks are present

These results are quite interesting in themselves because they demonstrate that there are cases where UCPOP can outperform GRAPHPLAN. However, if we vary the number of relevant pieces of information as well, the picture changes. In Figure 2, the x-axis measures the number of relevant blocks (note the log-scale for the CPU time).

Although GRAPHPLAN seems to be affected by irrelevance in our examples much more than UCPOP, GRAPHPLAN is so much faster that this does not matter. Nevertheless, GRAPHPLAN's sensitivity to irrelevance can be quite serious. For example, a blocks-world planning task with eight blocks and ten colors cannot be solved by GRAPHPLAN without recompiling the planning system using a larger data structure for storing nodes in the planning graph.

Further, in the examples with irrelevant blocks, the memory requirements grow with number of blocks regardless of whether the blocks are relevant or not

[3]These and the following results were obtained by a single trial for each data point on a SPARCStation4/110 with 64MB main memory.

[4]Note that the colors are not goal-irrelevant because colors are mentioned in the goal description. For this reason, GRAPHPLAN cannot detect their irrelevance.

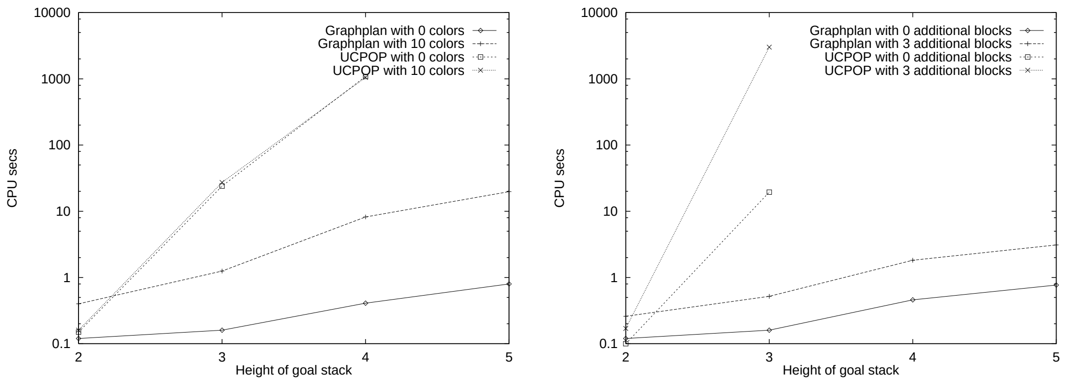


Figure 2: CPU time for stacking n blocks when colors or other blocks are present

(see Figure 3). Since those memory requirements are so severe that they can fill up the memory of ordinary workstations even for planning tasks containing only 15 blocks, any attempt to reduce memory consumption is worthwhile. Finally, the runtime requirements grow with the total number of blocks. In fact, 15 blocks seemed to be the maximum number of blocks GRAPHPLAN could handle in one hour for our example set. So, removing any single irrelevant block can help in making a planning task practically solvable.

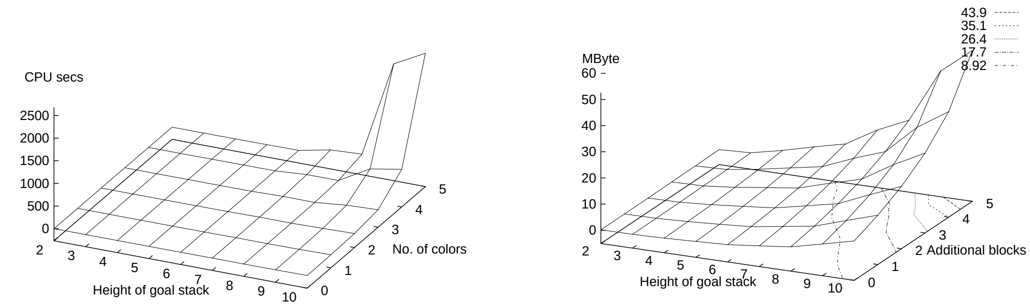


Figure 3: CPU time and memory requirement of GRAPHPLAN for stacking n blocks in the presence of other blocks

4 Selecting Relevant Information by Backchaining and Minimizing the Use of Initial Facts

As sketched above, one can straight-forwardly create top-down expectations in the planning process by backchaining from the goals creating an AND-OR-tree, where the AND-nodes are sets of ground facts (goals or the preconditions of an operator) and the OR-nodes are single ground facts that can be generated by different operators. We call such a tree a **fact-generation tree**.

An OR-node of a fact-generation tree is considered to be **solved** if it is an initial fact or if one of its immediate children is solved. An AND-node is solved if all of its children are solved. The entire fact-generation tree is solved if the AND-node corresponding to the goals is solved. The fact-generation tree can be grown until it is solved or a preset depth is reached without a solution.

This is a very crude approximation to the planning process because the state of the world and delete-effects of operators are completely ignored. However, it provides us with a straight-forward and sound method for deciding that a planning task cannot be solved in d steps (where non-conflicting steps can be executed in parallel as in GRAPHPLAN). In fact, we used this method to debug our planning task specifications, which often turned out to be unsolvable because of trivial typing errors.

Proposition 2 *If a fact-generation tree of depth d does not have a solution, then there does not exist any $\lfloor (d-1)/2 \rfloor$-step plan to solve the corresponding planning task.*

Although the method is straight-forward, it is not by itself computationally efficient. Since the tree grows exponentially with its depth, memory and runtime costs can be quite high even for moderate branching and depth. *Memoizing* the results for each ground fact together with the level in the tree and reusing the result if the new search node is as close to the leave nodes as the memoized results makes the process more efficient.[5] With memoizing, every ground fact will be created at most once per OR-level in the search tree, restricting the number of explored nodes polynomially in the depth of the tree and the size of the instantiated planning task description.

Proposition 3 *Creating a fact-generation tree of depth d can be done in time polynomial in $d \times n \times m$, where n is the number of ground facts and m is the number of ground operators.*

While it suffices to propagate the values *true* and *false* in a fact-generation tree in order to detect unsolvability, we need more sophisticated ways to compute a solution if we want to compute the pieces of information that are probably relevant for the planning process. The idea is to determine a set-inclusion minimal set of initial facts that are necessary to solve the fact-generation tree. Since there are in general different ways to generate a fact, we determine for every AND- and OR-node the set of sets of initial facts that could be used to generate this particular node. We call these sets of sets **possibility sets**.

In order to illustrate this idea, we present a small artificial example. STRIPS-operators will be written as

$$N : P \rightsquigarrow A/D,$$

[5] This means, we effectively create a directed graph instead of tree.

where N is the name of the operator, P is the set of preconditions, A is the set of add-effects, and D is the set of delete-effects. We use the following set of operators:

$$\begin{array}{llcll} \mathit{O1}: & \{a,b\} & \leadsto & \{x,z\} & / \ \{a\} \\ \mathit{O2}: & \{b,c\} & \leadsto & \{x\} & / \ \emptyset \\ \mathit{O3}: & \{c\} & \leadsto & \{y\} & / \ \{c\} \end{array}$$

Further, we assume that $\{a,b,c\}$ is the set of initial facts and $\{x,y\}$ is the set of goals. Then, as demonstrated in Figure 4, the possibility set for the goals is $\{\{a,b,c\},\{b,c\}\}$, and the set-inclusion minimal set of initial facts that solves the fact-generation tree is $\{b,c\}$.

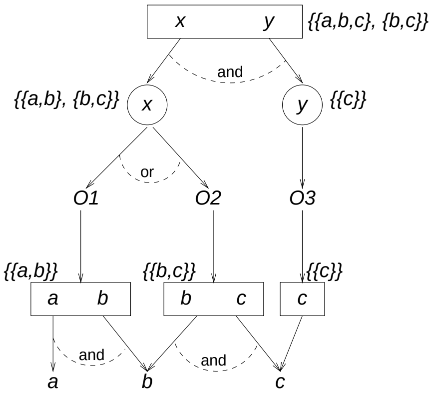


Figure 4: Example for computing the minimal set of initial facts that can be used to solve the fact-generation tree

The computation of such possibility sets is computationally expensive, however. At each AND-node, the elements of the possibility set result from computing the union over all choices of picking elements from the possibility sets of the OR-nodes immediately below the AND-node. For example, the possibility set of the AND-node $\boxed{xy}$ in Figure 4 is obtained by computing $\{a,b\} \cup \{c\}$ and $\{b,c\} \cup \{c\}$. In general, we may get a possibility set at an AND-node with a number of elements that is the product of the cardinalities of the sets at the OR-nodes immediately below the AND-node. This implies that the number of elements of these sets can grow exponentially with the breadth and depth of the fact-generation tree. Further, there does not seem to be an easy way out here. The problem of computing a minimal set of initial facts solving the fact-generation tree is a computationally hard problem.

Theorem 4 *Deciding whether a set of initial facts is a set-inclusion minimal set that solves a fact-generation tree is* NP*-complete for propositional* STRIPS.

Proof sketch. Membership in NP is obvious, because one can decide in polynomial time whether a fact-generation tree has a solution. NP-hardness follows from a straight-forward reduction of the *hitting set* problem to the problem at hand. ■

Since the computation of minimal sets of initial facts that are necessary for solving the fact-generation tree is intended to be a heuristic for determining the relevance of pieces of information, we are not forced to compute all elements of a possibility set, however. Further, even non-minimal sets of initial facts can be useful. For this reason, we use a crude approximation. At every node we store only n cardinality-minimal sets.[6]

Having computed the (approximation of a) possibility set for the goals, there is the question of what to do with the result. First of all, we must decide what to do in case the possibility set has more than one element. Different methods are conceivable:

1. use the union over all elements in the possibility set,
2. use the union over all set-inclusion minimal sets,
3. use the union over all cardinality-minimal sets, or
4. pick a set with a minimal number of elements.

Running the heuristic on all the GRAPHPLAN examples and some other examples revealed that most of the time the fourth method, which we call **one-best-set** method, is a good choice. Sometimes, however, this led to longer plans than necessary, because resources that could have been used were removed. Worse yet, in the *rocket* examples it unfortunately removes one rocket so that the planning task becomes unsolvable. In those cases, the third method, which we call **all-best-sets** method, proved to be a safe way out. Finally, for the examples considered, we were never forced to use the first or second method.

Secondly, we must decide what we want to do with this **set of probably relevant initial facts**. Since the creation of a fact-generation tree is only a very crude approximation of the planning process, initial facts that are necessary for solving the planning task might have been missed. In fact, if changing in the example above the definition of $O2$ to

$$O2\colon\quad \{b,c\} \;\rightsquigarrow\; \{x\} \;\;/\;\; \{c\},$$

it is evident that we cannot solve the planning task using only the initial facts $\{b,c\}$. The entire set of initial facts is now solution-relevant.

We used three methods of selecting "probably relevant" pieces of information with an increasing degree of restriction:

1. Consider all *objects* mentioned in the set of probably relevant initial facts as relevant and filter out initial facts that contain irrelevant objects.

[6] We used $n = 10$, which turns out to be sufficient in the cases we considered.

2. Consider only the *initial facts* in the set of probably relevant initial facts as relevant.

3. Consider only those *ground operators* as relevant that appear in the fact-generation tree and can be generated by the set of probably relevant initial facts.

These methods can be justified as follows. In domains containing a graph that has to be traversed (e.g., McDermott's [1996] *Manhattan world*), where undirected edges are represented by pairs of initial facts, only the first method is useful. The second method is useful if the fact-generation tree is similar to a final plan in that it makes use of all the relevant initial facts. The third method is useful if in building the fact-generation tree all ground operators are used that are necessary for the real plan.

It turned out that for different planning domains different strategies are effective. In the blocks-world domain, for example, a solved fact-generation tree often contains all the necessary ground operators to solve the planning task. In general, it seems to be a good *meta-heuristic* to try the third strategy first, since if it is unsuccessful, GRAPHPLAN fails fast. Then one should try the second and then the first strategy. Since determining the set of "probably relevant initial facts" is also only approximate, this sequence should be interleaved with the *one best set* and *all best sets* strategies described above. In case, we do not get a solution in this way, one may still run the planner on the original problem. In this way, we get (theoretical) completeness although the family of heuristics itself is not solution preserving (see also [Estlin and Mooney, 1996]).

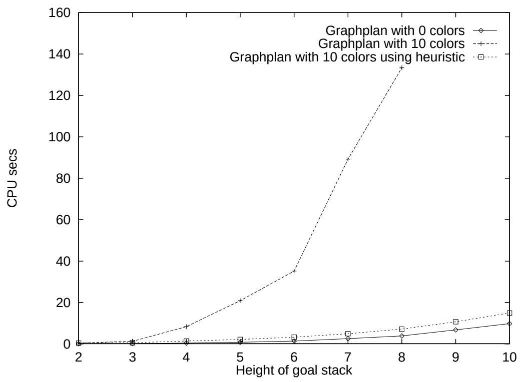
5 Empirical Results

We implemented the family of heuristics described above as a C-program that can be used as a filter for GRAPHPLAN,[7] and tested them on a large set of examples. Applying our heuristics to the examples in Section 3 revealed that they effectively remove the irrelevant information in our blocks-world planning tasks. The CPU time of GRAPHPLAN on the problems and the CPU time of the heuristics combined with GRAPHPLAN are shown in Figure 5.

In order to get an idea of how the heuristics behave on different domains, we ran them on the GRAPHPLAN examples. Runtime is not an issue here, because all problem are solved in a few seconds. More interesting is the question whether the heuristics are *solution-length preserving* or *solution preserving*. An answer to this question is given in Table 1.

In 55% of all examples, it was feasible to use the ground operators that appear in the fact-generation tree basing the selection on the one-best-set method. If the

[7] The program can be obtained from the authors.



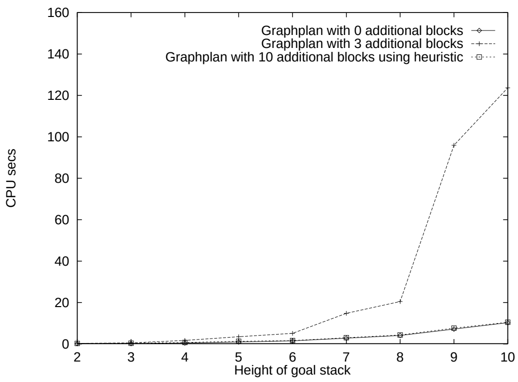


Figure 5: Graphplan CPU time for stacking n blocks when colors or other blocks are present and when using the heuristic

	select ground operators		select initial facts		select objects	
Domain	one best set	all best sets	one best set	all best sets	one best set	all best sets
blocks	(L)	(L)	L	L	L	L
fixit	L	L	L	L	L	L
fridge	L	L	L	L	L	L
fridge	L	L	L	L	L	L
link[a]	S	S	S	S	S	S
logistics	S	L	S	L	S	L
mblocks		S	S	L	S	L
monkey	L	L	L	L	L	L
rocket		(L)		(L)		L
tsp					L	L

[a]The initial fact $g*$ is removed because this fact can be generated by the operator $A*$. For this reason the *link* plans are one step longer after the heuristics have been applied.

Table 1: Solution- and solution-length preserving properties of the different heuristics for different domains. An “S” means that the particular heuristic did not change the solution-existence property for all the examples considered, an “L” means that the plans have the same length. If the letter is in parentheses, it means that some examples did not have the property.

all-best-sets method is used, 86% of the examples could be solved. In case of the *logistics* and *mblocks* domains, however, basing the selection on *one best set* leads to longer plans, because useful resources are removed. In the *rockets* domain, the removal of one of the rockets leads to unsolvability.[8] Even worse is the *tsp* domain. Here, we have to base the selection of relevant information on the *objects* that appear in the set of probably relevant initial facts. The reason is that the

[8]If in the planning task description one could distinguish between usable resources and the initial state information, the heuristic would have a much better way to decide what is relevant.

undirected edges in the graph are represented by two ground facts, standing for two directed edges. In building the fact-generation tree, however, often only one of those facts is used, i.e., we only traverse the edge in one direction, while in the final plan we must also use the other direction.

More interesting planning tasks are, of course, those that are hard for GRAPHPLAN. One such set of hard tasks is the test set that has been used to demonstrate the performance of SATPLAN [Kautz and Selman, 1996]. The results for applying our heuristics are given in Table 2.

	GRAPHPLAN		GRAPHPLAN with heuristic selecting ground operators				GRAPHPLAN with heuristic selecting initial facts			
			one best set		all best sets		one best set		all best sets	
Task	time /act.	CPU secs	time /act.	CPU secs	time /act.	CPU secs	time /act.	CPU secs	time /act.	CPU secs
rocket_ext.a	7/34	295	–	0.5	–	0.6	8/27	1.4	7/34	296
rocket_ext.b	7/30	1339	–	1.0	–	2.0	10/29	72.5	7/30	1340
logistics.a	11/54	5006	13/51	3022	11/54	4569	13/51	9291	11/54	5009
logistics.b	13/45	2429	15/42	746	13/45	2190	15/42	2348	13/45	2432
bw_large.a	12/12	5.7	12/12	1.4	12/12	1.4	12/12	3.1	12/12	3.1
bw_large.b	18/18	556	18/18	2.9	18/18	2.9	18/18	112	18/18	112

Table 2: Time steps and actions of generated plan and CPU time needed for plan generation on SATPLAN examples. A "–" indicates that no solution was found.

Although the planning tasks descriptions were not intentionally designed to have irrelevant information in it, it turned out that the *bw_large* examples contain irrelevant facts. Further, also the *logistics* and *rocket_ext* examples contain more initial facts than necessary to generate a solution. However, in these cases the facts correspond to resources which, if removed, lead to longer plans. In any case, applying the heuristics seems to pay off on this set of examples. If too much information is removed, GRAPHPLAN is very fast in detecting this. Further, overall planning time is reduced in almost all cases, if one uses the meta-heuristics described above.

Another interesting planning task is McDermott's [1996] *Manhattan world* example. GRAPHPLAN cannot handle it because the memory requirements are too high due to too many initial facts, namely 714. Applying our heuristic to it returned a reduced planning task with only 58 initial facts after 6 CPU seconds. Running GRAPHPLAN on this reduced task resulted in a plan with length 40 after another 30 CPU seconds. McDermott [1996] reported that his UNPOP planner needs 30 minutes to find a plan.[9]

[9] McDermott's results were obtained on a SPARCStation2, which is approximately 5–7 times slower than a SPARCStation4. This means that our approach is still 7–10 times faster on this example.

It must be noted, however, that we used a representation of the planning task that is different from McDermott's representation. First of all, we compiled all the conditional operators into sets of unconditional operators, which gives an exponential blowup in the number of operators resulting usually in a very bad performance [Koehler *et al.*, 1997]. In our case, the blowup was moderate, however. Secondly, we represented the grid of the Manhattan world as a graph, where each grid point is a node in the graph. With this representation, the heuristic could identify the grid points that have to be visited to solve the task. In McDermott's representation there are only x and y coordinates, which does not support the identification of relevant grid points by our heuristics. Instead one could identify relevant coordinate values, which is much less restrictive. This shows that our heuristics are very sensitive to the representation of planning task. However, representation is always a very important factor for planning [Kautz *et al.*, 1996].

6 Discussion and Conclusion

Starting from the hypothesis that the GRAPHPLAN system is highly sensitive to irrelevant information in the planning task description, we analyzed the notion of *irrelevance* and compared GRAPHPLAN with UCPOP on planning tasks that contain obviously irrelevant information. Although GRAPHPLAN appears to be much more sensitive to irrelevant information than UCPOP, the former system is so much more efficient that its sensitivity to irrelevance does not seem to matter much. Nevertheless, irrelevant information can lead to high memory and CPU time consumption for GRAPHPLAN.

In order to account for this, we developed a family of heuristics aimed at identifying relevant information. These heuristics are very similar to McDermott's [1996] *greedy regression graph* technique. The main difference is that we base our selection on minimal sets of initial facts necessary to solve the fact-generation tree, while McDermott bases his selection on the minimal number of actions. Minor differences are that AND-nodes are completely instantiated in our case while they may contain variables in McDermott's system. Further, McDermott uses best-first search or discrepancy-search, while we used a simple iterative depth-first search. The most striking difference is that we run our heuristic once before planning starts, while McDermott uses his heuristic each time before extending the plan by an action. Nevertheless, our approach results in quite reliable predictions most of the time.

Despite its simplicity, the heuristics turned out to be very useful. In many domains, the most constraining heuristic gives solution-preserving results and reduces planning time considerably. Further, the heuristics enable GRAPHPLAN to find plans for McDermott's *Manhattan domain*.

Although we used the heuristic only in combination with GRAPHPLAN, it can, of course, be combined with any other planner based on the STRIPS formalism such as SATPLAN [Kautz and Selman, 1996]. We plan to extend the heuristic to deal with ADL operators [Koehler *et al.*, 1997] and to integrate it more tightly into the planner.

References

[AAAI-96, 1996] *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI-96)*, Portland, OR, July 1996. MIT Press.

[Bäckström, 1992] Christer Bäckström. Equivalence and tractability results for SAS^+ planning. In Nebel et al. [1992], pages 126–137.

[Blum and Furst, 1997] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.

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

[Earley, 1970] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 26(1):57–61, 1970.

[Estlin and Mooney, 1996] Tara A Estlin and Raymond J. Mooney. Multi-strategy learning of search control for partial order planning. In AAAI-96 [1996], pages 843–848.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Kautz and Selman, 1996] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In AAAI-96 [1996], pages 1194–1201.

[Kautz *et al.*, 1996] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In L. Carlucci Aiello, J. Doyle, and S. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 5th International Conference (KR-96)*, pages 374–385, Cambridge, MA, November 1996. Morgan Kaufmann.

[Koehler *et al.*, 1997] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. Technical Report 88, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany, 1997.

[McDermott, 1996] Drew McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 142–149, Portland, OR, 1996. AAAI Press, Menlo Park.

[Nebel *et al.*, 1992] B. Nebel, W. Swartout, and C. Rich, editors. *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, Cambridge, MA, October 1992. Morgan Kaufmann.

[Penberthy and Weld, 1992] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Nebel et al. [1992], pages 103–114.