A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm

Jörg Hoffmann[1]

Abstract. We present a new heuristic method to evaluate planning states, which is based on solving a relaxation of the planning problem. The solutions to the relaxed problem give a good estimate for the length of a real solution, and they can also be used to guide action selection during planning. Using these informations, we employ a search strategy that combines Hill-climbing with systematic search. The algorithm is complete on what we call *deadlock-free* domains. Though it does not guarantee the solution plans to be optimal, it does find close to optimal plans in most cases. Often, it solves the problems almost without any search at all. In particular, it outperforms all state-of-the-art planners on a large range of domains.

1 INTRODUCTION

The standard approach to obtain a heuristic is to relax the problem $\mathcal{P}$ at hand into some easier problem $\mathcal{P}'$. The optimal solution length to a situation in $\mathcal{P}'$ can then be used as an admissible estimate for the optimal solution length of the same situation in $\mathcal{P}$. An application of this idea to domain independent planning was first used in the HSP system [3]. The planning problem $\mathcal{P}$ is relaxed by simply ignoring the delete lists of all operators. However, computing the optimal solution length for a planning problem without delete lists is still NP-hard, as was first shown by Bylander [4]. Therefore, the HSP heuristic is only a rough estimate of the optimal relaxed solution length. In short, it is obtained by summing up the minimal distances of all atomic goals.

In this paper, we go one step further. We introduce a method that computes *some*, not necessarily optimal, solution to the relaxed problem. These solutions are helpful in two ways:

- their length provides an informative estimate for the difficulty of a situation;
- one can use them as a guidance for action selection.

The solution length estimates are used to control a local search strategy similar to Hill-climbing, which is combined with systematic breadth first search in order to escape local minima or plateaus. The guidance information is employed to cut down the branching factor during systematic search. The method shows good behavior over all domains that are commonly used in the planning community. In particular, we will see that it is complete on the class of problems we call *deadlock-free*. Performing local search, the method can not guarantee its solution plans to be optimal. In spite of this, it finds close to optimal plans in most cases. As a benefit from the severe restriction of its search space, it shows very competitive runtime behavior. For example, *logistics* problems are solved faster than by any other domain independent planning system known to the author at the time of writing.

[1] Institute for Computer Science, Albert Ludwigs University, Georges-Köhler-Allee, Geb. 52, 79110 Freiburg, Germany, email: hoffmann@informatik.uni-freiburg.de

2 BACKGROUND

Throughout the paper, we consider simple STRIPS domains. We briefly review two standard notations. An *action* o has the form

$$o = \langle\, pre(o) \;\Rightarrow\; add(o),\; del(o) \,\rangle$$

where $pre(o)$, $add(o)$ and $del(o)$ are sets of ground facts. Plans P are sequences $P = \langle o_1, \ldots, o_n \rangle$ of actions, i.e., we consider only linear plans.

3 HEURISTIC

In this section, we introduce a method for heuristically evaluating planning states S. Basically, the method consists of two parts.

1. First, the *relaxed fixpoint* is built on S. This is a forward chaining process that determines in how many steps, at best, a fact can be reached from S, and with which actions.
2. Then, a *relaxed solution* is extracted from the fixpoint. This is a sequence of parallel action sets that achieves the goal from S, if their delete effects are ignored.

The first part corresponds directly to the heuristic method that is used in HSP [3]. The second part goes one step further: while in HSP, the heuristic is extracted as a side effect of the fixpoint, we invest some extra effort to find a relaxed plan, and use the plan to determine our heuristic value. The fixpoint process is depicted in Figure 1.

The algorithm can be seen as building a layered graph structure, where fact and action layers are interleaved in an alternating fashion. The process starts with the initial fact layer, which are the facts that are TRUE in S. Then, the first action layer comprises the actions whose preconditions are contained in S. The effects of these actions lead us to the second fact layer, which, in turn, determines the next action layer and so on. The process terminates, and remembers the number max of the last layer, if all goals are reached or if the new fact layer is identical to the last one.

The crucial information that the fixpoint process gives us are the *levels* of all facts and actions. These are defined as the number of the first fact- or action layer they are members of.

$$\text{level}(f) := \begin{cases} min\{i \mid f \in F_i\} & \text{ex. } i : f \in F_i \\ \infty & \text{otherwise} \end{cases}$$

```
$F_0 := S$
$k := 0$
**while** $\mathcal{G} \not\subseteq F_k$ **do**
    $O_k := \{o \in \mathcal{O} \mid pre(o) \subseteq F_k\}$
    $F_{k+1} := F_k \cup \bigcup_{o \in O_k} add(o)$
    **if** $F_{k+1} = F_k$ **then**
        **break**
    **endif**
    $k := k + 1$
**endwhile**
$max := k$
```

Figure 1. Computing the relaxed fixpoint on a planning state S. $\mathcal{O}$ and $\mathcal{G}$ denote the action set and goal state of the problem at hand, respectively.

$$\text{level}(o) := \begin{cases} min\{i \mid o \in O_i\} & \text{ex. } i : o \in O_i \\ \infty & \text{otherwise} \end{cases}$$

We now show how to extract a relaxed plan from the fixpoint structure. This is done in a backward chaining manner, where we simply use any action with minimal level to make a goal TRUE. The exact algorithm is depicted in Figure 2. Note that we *do not need to search*, we can proceed right away to the initial state and are guaranteed to find a solution.

```
**for** $i := 1, \ldots, max$ **do**
    $G_i := \{g \in \mathcal{G} \mid \text{level}(g) = i\}$
**endfor**
$h := 0$
**for** $i := max, \ldots, 1$ **do**
    **for** all $g \in G_i,\ g$ not TRUE at $i$ **do**
        select $o$ with $g \in add(o)$ such that level$(o) = i - 1$
        $h := h + 1$
        **for** all $f \in pre(o),\ f$ not TRUE at $i - 1$ **do**
            $G_{\text{level}(f)} := G_{\text{level}(f)} \cup \{f\}$
        **endfor**
        **for** all $f \in add(o)$ **do**
            mark $f$ as TRUE at $i - 1$ and $i$
        **endfor**
    **endfor**
**endfor**
```

Figure 2. The algorithm that extracts a relaxed solution to a state S after the fixpoint has been built.

Before plan extraction starts, an array of goal sets G_i is initialized by inserting all goals with corresponding level. The mechanism then proceeds down from layer max to layer 1, and selects an action o for each goal g at the current layer i, incrementing the plan length counter h. No actions are selected for goals that are marked TRUE at the time being, as they are already added. The achiever o is required to have level$(o) = i - 1$. This is minimal as the goal g has level i, i.e., the first action that achieved g in the fixpoint came in at level $i - 1$. The preconditions of o are inserted as new goals into their corresponding goal sets. If the current layer is i, then the levels of o's preconditions are at most $i - 1$, so these new goals will be made TRUE later during the process.

3.1 Goal Distance

To obtain the heuristic goal distance value $h(S)$ of a given planning state S, we now simply chain the two algorithms together. First, we perform the fixpoint computation from Figure 1. If the process terminates without reaching the goals, we set $h(S) := \infty$. Otherwise, we extract a relaxed plan, Figure 2, and use the plan length for evaluation, i.e., $h(S) := h$.

The overall structure of the relaxed planning process is quite similar to planning with planning graphs [1]. It amounts to a very special case, as no negative interactions at all occur between facts or actions in the relaxed problem.

3.2 Helpful Actions

We can also use the extracted plan to determine a set of actions that seem to be helpful in reaching the goal. To do this, we turn our look on the actions that are contained in the *first time step* of the relaxed solution, i.e., the actions that are selected at level 0. These are often the actions that are useful in the given situation. Let us see a simple example for that, taken from the *gripper* domain, as it was used in the 1998 AIPS planning systems competition. We do not repeat the exact definition of the domain here, as it is easily understood intuitively. There are two rooms, A and B, and a certain number of balls, which shall be moved from room A to room B. The planner changes rooms via the **move** operator, and controls two grippers which can **pick** or **drop** balls. Each gripper can only hold one ball at a time. We look at a small problem where 2 balls must be moved into room B. A relaxed solution to the initial state that our heuristic might extract is

$<$ { **pick** ball1 A left,
pick ball2 A left,
move A B },
{ **drop** ball1 B left,
drop ball2 B left } $>$

This is a parallel relaxed plan consisting of two time steps. Note that the **move** A B action is selected parallel to the **pick** actions, as the relaxed planner does not notice that it can not **pick** balls in room A anymore once it has moved into room B. In a similar fashion, both balls are picked with the left gripper. Nevertheless, two of the three actions in the first step are helpful in the given situation: both **pick** actions are starting actions of an optimal sequential solution. Thus, one might be tempted to define the set $H(S)$ of helpful actions as only those that are contained in the first time step of the relaxed plan. However, this is too restrictive in some cases. We therefore define our set $H(S)$ as follows.

$$H(S) := \{o \in O_0 \mid add(o) \cap G_1 \neq \emptyset\}$$

After plan extraction, O_0 contains the actions that are applicable in S, and G_1 contains the facts that were goals or subgoals at level 1. Thus, we consider as helpful those actions which add at least one fact that was a (sub)goal at the lowest time step of our relaxed solution.

4 SEARCH

We now introduce a search algorithm that makes effective use of the heuristics we defined in the last section. The key observation that leads us to the method is the following. On some domains, like the *gripper* problems from the 1998 competition and Russel's *tyreworld*, it is sufficient to use our heuristic in a naive *Hill-climbing* strategy. In these problems, one can simply start in the initial state, pick, in each state, a best valued successor, and ends up with an optimal solution plan. This strategy is very efficient on the problems where it finds plans.

However, the naive method does *not* find plans on most problems. Usually, it runs into an infinite loop. To overcome this problem, one could employ standard Hill-climbing variations, like restarts, limited plateau moves, or a memory for repeated states. We use an *enforced* Hill-climbing method instead, see the definition in Figure 3.

```
initialize the current plan to the empty plan $<>$
$S := \mathcal{I}$
obtain $h(S)$ by evaluating $S$
**if** $h(S) = \infty$ **then**
  output ”No Solution”, stop
**endif**
**while** $h(S) \neq 0$ **do**
  breadth first search for a state $S'$ with $h(S') < h(S)$
  **if** no such state can be found **then**
    output ”No Solution”, stop
  **endif**
  add the actions on the path to $S'$ at the end of the current plan
  $S := S'$
**endwhile**
```

Figure 3. The Enforced Hill-climbing algorithm. $\mathcal{I}$ denotes the initial state of the problem to be solved.

The algorithm combines Hill-climbing with systematic breadth first search. Like standard Hill-climbing, it picks some successor of the current state at each stage of the search. Unlike in standard Hill-Climbing, this successor does not need to be a direct one, and, unlike in standard Hill-Climbing, we do not pick any best valued successor, but *enforce* the successor to be one that is *better* than our current state.

More precisely, at each stage during search a successor state is found by performing breadth first search starting out from the current state S. For each search state S', all successors are generated and evaluated heuristically. Doubly occuring states are pruned from the search by keeping a hashtable of past states in memory, and the search stops as soon as it has found a state S' that has a better heuristic value than S. This way, the Hill-climbing search escapes plateaus and local minima by simply performing exhaustive search for an exit, i.e., a state with strictly better heuristic evaluation.

4.1 Helpful Actions

So far, we have only used the goal distance heuristic. We integrate the helpful actions heuristic into our search algorithm as follows. During breadth first search, we do not generate *all* successors of any search state S' anymore, but consider *only those* that are obtained by applying actions from $H(S')$. This way, the branching factor for the search is cut down. However, the helpful actions heuristic is not completeness-preserving, i.e., considering only the actions in $H(S')$ might make the search miss a goal state. If this happens, i.e., if the search can not reach any new states anymore when restricting the successors to $H(S')$, we simply switch back to complete breadth first search starting out from the current state S and generating *all* successors of search nodes.

5 COMPLETENESS

The Enforced Hill-climbing algorithm is complete on *deadlock-free* planning problems. We define a *deadlock* to be a state S that is reachable from the initial state $\mathcal{I}$, and from which the goal can not be reached anymore. A planning problem is called *deadlock-free*, if it does not contain any deadlock state. We remark that a deadlock-free problem is also solvable, cause otherwise the initial state itself would already be a deadlock.

Theorem 1 *Let $\mathcal{P}$ be a planning problem. If $\mathcal{P}$ is deadlock-free, then the* Enforced Hill-climbing *algorithm, as defined in Figure 3, will find a solution.*

Due to space restrictions, we do not show the (easy) proof of Theorem 1 here and refer the reader to [5]. In short, if the complete breadth first search starting from a state S can not reach a better evaluated state, then, in particular, it can not reach a goal state, which implies that the state S is a deadlock in contradiction to the assumption.

In [5], it is also shown that most of the currently used benchmark domains are in fact *deadlock-free*. Any solvable planning problem that is *invertible* in the sense that one can find, for each action sequence P, an action sequence $\overline{P}$ that undoes P's effects, does not contain deadlocks. One can always go back to the initial state first and execute an arbitrary solution thereafter. Moreover, planning problems that contain an *inverse action* $\overline{o}$ to each action o *are* invertible: simply undo all actions in the sequence P by executing the corresponding inverse actions. Finally, most of the current benchmark domains *do* contain inverse actions. For example in the *blocksworld*, we have **stack** and **unstack**. Similarly in domains that deal with logistics problems, for example *logistics*, *bulldozer*, *gripper* etc., one can often find inverse pairs of actions. If an action is not invertible, its role in the domain is often quite limited. A nice example is the **inflate** operator in the *tyreworld*, which can be used to inflate a spare wheel. Obviously, there is not much point in defining something like a **deflate** operator. More formally speaking, the operator does not destroy a goal or a precondition of any other operator in the domain. In particular, it does not lead into deadlocks.

As one of the anonymous reviewers pointed out to us, deadlock-free domains might be an artificially dominant group because of the simplicity of the current benchmarks. Any domain with consumable resources will contain non-invertible actions. This is certainly true to some extent. We have one theoretical and one practical answer.

- In theory, one can make Enforced Hill-climbing complete on *any* planning problem by simply adding an operator that is applicable in any situation, and reproduces the initial state. That way, search always has the opportunity to go back to the start. In practice, this is not likely to be an effective approach, as it would force complete breadth first search to go all the way down to a state S' with $h(S') < h^{min}$, where h^{min} is the evaluation of the best state seen so far.
- From a more practical point of view, our experience is that Enforced Hill-climbing usually fails quite quickly on the problems which it can not solve. One can then simply switch to a complete heuristic search algorithm, like greedy best-first or weighted A^*.

In the subsequent empirical investigation, we show results for a large collection of benchmark planning problems. All of them but one—a simple *sokoban* instance—are deadlock-free. This is not because we concentrated on solving problems that are deadlock-free, but because there are very few benchmarks available that are not.

Anyhow, Enforced Hill-climbing finds solutions to *all* of these problems, including the *sokoban* instance containing deadlocks.

6 EMPIRICAL RESULTS

For empirical evaluation, we implemented the Enforced Hill-climbing algorithm, using relaxed plans to evaluate states and to determine helpful actions, in C. We call the resulting planning system FF, which is short for FAST-FORWARD planning system. All running times for FF are measured on a Sparc Ultra 10 running at 350 MHz, with a main memory of 256 M Bytes. Where possible, i.e., for those planners that are publicly available, the running times of other planners were measured on the same machine. We indicate run times taken from the Literature in the text. All planners were run with the default parameters, unless otherwise stated in the text, and all benchmark problems are the standard examples taken from the Literature. Some benchmark problems have been modified in order to show how planners scale to bigger instances. We explain the modifications made, if any, in the text. Dashes indicate that the corresponding planner failed to solve that problem within half an hour.

6.1 The *logistics* Domain

This is a classical domain, involving the transportation of packets via trucks and airplanes. There are two well known test suites. One has been used in the 1998 AIPS planning systems competition, the other one is part of the BLACKBOX distribution. The problems in the competition suite are very hard. In fact, they are so hard that, up to date, no planner has been reported to solve them all. FAST-FORWARD is the first one that does. See Figure 4, showing also the results for GRT [12] and HSP-r [2], which are—as far as the author knows—the two best other domain independent *logistics* planners at the time being.[2]

	HSP-r		GRT		FF	
problem	time	steps	time	steps	time	steps
prob-01	0.36	35	0.28	30	0.06	27
prob-02	3.13	36	1.32	34	0.19	32
prob-03	25.45	64	5.55	60	0.71	54
prob-04	50.13	63	19.28	69	0.98	58
prob-05	0.62	27	0.39	26	0.08	22
prob-06	293.60	83	14.39	80	1.95	73
prob-07	6.20	37	1.76	37	0.38	36
prob-08	-	-	16.37	48	2.04	41
prob-09	371.03	97	50.48	98	2.08	91
prob-10	287.64	121	23.13	117	3.20	103
prob-11	4.58	34	1.54	36	0.21	30
prob-12	-	-	43.06	48	2.01	41
prob-13	-	-	85.58	79	7.73	67
prob-14	-	-	60.20	104	6.97	98
prob-15	19.52	120	67.50	106	1.27	93
prob-16	92.75	69	31.58	62	1.23	55
prob-17	29.35	61	12.19	53	0.63	44
prob-18	-	-	335.05	193	50.76	167
prob-19	-	-	238.98	174	16.26	151
prob-20	-	-	324.12	169	24.40	139
prob-21	-	-	294.23	120	8.93	102
prob-22	-	-	-	-	246.05	282
prob-23	100.67	145	16.86	118	3.84	126
prob-24	-	-	98.54	49	4.17	40
prob-25	-	-	-	-	106.23	181
prob-26	-	-	-	-	71.15	183
prob-27	-	-	-	-	71.26	141
prob-28	-	-	-	-	679.43	265
prob-29	-	-	-	-	589.75	323
prob-30	-	-	-	-	62.4	131

Figure 4. Results of the three domain independent planners best suited for *logistics* problems on the competition suite. Times are in seconds, *steps* counts the number of actions in a sequential plan. For HSP-r, the weighting factor W is set to 5, as was done in the experiments described by Bonet and Geffner in [2].

[2] It is important to distinct the results shown in Figure 4 for HSP-r from those reported by Bonet and Geffner [2]. Those results were taken on the problems from the BLACKBOX distribution, while our results are taken on the competition test suite.

The times for GRT in Figure 4 are from the paper by Refanidis and Vlahavas [12], where they are measured on a Pentium 300 with 64 M Byte main memory. FF outperforms both HSP-r and GRT by an order of magnitude. Also, it finds shorter plans than the other planners.

We also ran FF on the benchmark problems from the BLACKBOX distribution suite, and it solved all of them in less than half a second. Compared to the results shown by Bonet and Geffner [2] for these problems, FF was between 2 and 10 times faster than HSP-r, finding shorter plans in all cases.

6.2 Mixed classical Problems

FAST-FORWARD shows competitive behavior on all commonly used benchmark domains. To exemplify this, we show a table of running times on a variety of different domains in Figure 5, comparing FF against a collection of state-of-the-art planning systems: IPP [8], STAN [9], BLACKBOX [7], and HSP [3].

In Figure 5, the planning problems shown are the following. The *tyreworld* problem was originally formulated by Russell, and asks the planner to replace a flat tire. The problem is modified in a natural way so as to make the planner replace n flat tires. FF is the only planner that is capable of replacing more than three tires, scaling up to much bigger problems.

The *hanoi* problems make the planner solve the well known *Towers of Hanoi* problem, with n discs to be moved. FF also outperforms the other planners on these problems.

The *sokoban* problem encodes a small instance of a well known computer game, where a single stone must be pushed to its goal position. Although the problem contains deadlocks, FF has no difficulties in solving it.

The *manhattan* domain was first introduced by McDermott [10]. In these problems, the planner controls a robot which moves on a $n \times n$ grid world, and has to deal with different kinds of keys and locks. The original problem taken from [10] corresponds to the *mh-11* entry in Tabular 5, where the robot moves on a 11×11 grid. The other entries refer to problems that have been modified to encode 7×7, 15×15 and 19×19 grid worlds, respectively. FF easily handles all of them, finding slightly suboptimal plans.

Finally, the *blocksworld* problems in Figure 5 are benchmark examples taken from [6]. FF outperforms the other planners in terms of running time as well as in terms of solution length.

7 RELATED WORK

The closest relative to the work described in this paper is, quite obviously, the HSP system [3]. In short, HSP does Hill-climbing search, with the heuristic function

$$h(S) := \sum_{g \in \mathcal{G}} weight_S(g)$$

The weight of a fact with respect to a state S is, roughly speaking, the minimum over the sums of the precondition weights of all actions that achieve it. The weights are obtained as a side effect of doing exactly the same fixpoint computation as we do. The main problem in HSP is that the heuristic needs to be recomputed for each single search state, which is very time consuming. Inspired by HSP, a few approaches have been developed that try to cope with this problem, like HSP-r [2] and the GRT-planner [12].

The authors of HSP themselves handle the problem by sticking to their heuristic, but changing the search direction, going backwards

		IPP		STAN		BLACKBOX		HSP		FF	
domain	problem	time	steps	time	steps	time	steps	time	steps	time	steps
tyreworld	fixit-1	0.04	19	0.10	19	0.43	19	0.35	23	0.04	19
tyreworld	fixit-2	11.29	30	1.25	30	114.32	30	-	-	0.09	30
tyreworld	fixit-3	-	-	-	-	933.14	41	-	-	0.20	41
tyreworld	fixit-4	-	-	-	-	-	-	-	-	0.42	52
hanoi	tower-3	0.03	7	0.03	7	0.23	7	0.31	7	0.01	7
hanoi	tower-5	0.11	31	0.27	31	680.6	31	2.04	31	0.09	31
hanoi	tower-7	1.93	127	6.10	127	-	-	23.18	163	0.52	127
hanoi	tower-9	39.31	511	230.20	511	-	-	-	-	6.45	511
sokoban	sokoban-1	1.15	25	1.51	25	1283.29	25	13.87	29	0.22	25
manhattan	mh-7	4.82	35	20.04	35	-	-	1.12	35	0.09	38
manhattan	mh-11	65.12	40	1013.96	40	-	-	13.31	40	0.26	43
manhattan	mh-15	-	-	-	-	-	-	-	-	0.64	59
manhattan	mh-19	-	-	-	-	-	-	-	-	1.53	87
blocksworld	bw-large-a	0.47	10	0.57	10	10.30	10	0.78	11	0.04	7
blocksworld	bw-large-b	2.20	14	4.04	14	160.14	14	1.54	13	0.10	10
blocksworld	bw-large-c	88.17	25	267.08	26	-	-	4.34	20	0.56	16
blocksworld	bw-large-d	362.19	33	-	-	-	-	11.36	27	1.42	20

Figure 5. Running times and quality (in terms of number of actions) of plans for FF and state-of-the-art planners on various classical domains. All planners are run with the default parameters, except HSP, where loop checking needs to be turned on.

from the goal in HSP-r instead of forward from the initial state in HSP. This way, they need to compute a weight value for each fact only once, and simply sum the weights up for a state later during search.

The authors of [12] invert the direction of the HSP *heuristic* instead. While HSP computes distances by going towards the goal, GRT goes from the goal to each fact, and estimates its distance. The function that then extracts, for each state during forward search, the state's heuristic estimate, uses the pre-computed distances as well as some information on which facts will probably be achieved simultaneously.

For the FAST-FORWARD planning system, a somewhat paradoxical extension of HSP has been made. Instead of avoiding the major drawback of the HSP strategy, we even worsen it, at first sight: the heuristic keeps being fully recomputed for each search state, and we even put some extra effort on top of it, by extracting a relaxed solution. However, the overhead for extracting a relaxed solution is marginal, and the relaxed plans can be used to prune unpromising branches from the search tree.

To verify where the enormous run time advantages of FF compared to HSP come from, we ran HSP using Enforced Hill-climbing search with and without helpful actions pruning, as well as FF without helpful actions on the problems from our test suite. Due to space restrictions, we can not show our findings in detail here. It seems that the major steps forward are our variation of Hill-climbing search in contrast to the restart techniques employed in HSP, as well as the helpful actions heuristic, which prunes most of the search space on many problems. Our different heuristic distance estimates seem to result in shorter plans and slightly, about a factor two, better running times, when one compares FF to a version of HSP that uses Enforced Hill-climbing search and helpful actions pruning. We did not yet find the time to do these experiments the other way round, i.e., integrate our heuristic into the HSP search algorithm, as this would involve modifying the original HSP code, which means a lot of implementation work.

There has been at least one more approach in the Literature where goal distances are estimated by ignoring the delete lists of the operators. In [10], Greedy Regression-Match Graphs are introduced. In a nutshell, these estimate the goal distance of a state by backchaining from the goals until facts are reached that are TRUE in the current state, and then counting the estimated minimal number of steps that are needed to achieve the goal state.

To the best of our understanding, the action chains that lead to a state's heuristic estimate in [10] are similar to the relaxed plans that we extract. However, the backchaining process seems to be quite costly. For example, building the Greedy Regression-Match Graph for the initial state of the *manhattan* world 11×11 grid problem is reported to take 25 seconds on a Sparc 2 station. For comparison, we ran FF on a Sparc 4 station. Finding a relaxed plan for the initial state takes less than one hundredth of a second, i.e., the time measured is 0.00 CPU seconds.

The helpful actions heuristic shares some similarities with what is known as relevance from the literature [11]. The main difference is that relevance in the usual sense refers to what is useful for solving the whole problem. Being helpful, on the other hand, refers to something that is useful *in the next step*.

8 CONCLUSION AND OUTLOOK

In this paper, we presented two heuristics for domain independent STRIPS planning, one estimating the distance of a state to the goal, and one collecting a set of promising actions. Both are based on an extension of the heuristic that is used in the HSP system. We showed how these heuristics can be used in a variation of Hill-climbing search, and we have seen that the algorithm is complete on the class of deadlock-free domains. We collected empirical evidence that the resulting planning system is among the fastest planners in existence nowadays, outperforming the other state-of-the-art planners on quite a range of domains, like the *logistics*, *manhattan* and *tyreworld* problems.

To the author, the most exciting question is this: *Why* is the heuristic information obtained in this simple manner so good? It is not really difficult to construct abstract examples where the approach produces arbitrarily bad plans, or uses arbitrarily much time, so why does it almost never go wrong on the benchmark problems? Why is the relaxed solution always so close to a real solution, except for the *Tower of Hanoi* problems? Is it possible to define a notion of "simple" planning domains, where relaxed solutions have desirable properties?

First steps into that direction seem to indicate that, in fact, there might be some underlying theory in that sense. In particular, it can be proven that the Enforced Hill-climbing algorithm finds optimal solutions when the heuristic used is *goal-directed* in the following sense:

$$h(S) < h(S') \;\Rightarrow\; min(S) < min(S')$$

Here, $min(S)$ denotes the length of the shortest possible path from state S to a goal state, i.e., Enforced Hill-climbing is optimal when heuristically better evaluated states are really closer to the goal.

It can also be proven that the length of an *optimal* relaxed solution *is*, in fact, a goal-directed heuristic in the above sense on the problems from the *gripper* domain that was used in the 1998 planning systems competition. We have not yet, however, been able to identify some general structural property that implies goal-directedness of optimal relaxed solutions.

Apart from these theoretical investigations, we want to extend the algorithms to handle richer planning languages than STRIPS, in particular ADL and resource constrained problems.

ACKNOWLEDGEMENTS

The author thanks Bernhard Nebel for helpful discussions and suggestions on designing the paper. Thanks also go the the referees for their comments which helped improve the paper.

REFERENCES

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

[2] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of the 5th European Conference on Planning*, pages 359–371, 1999.

[3] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence*, pages 714–719, 1997.

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

[5] J. Hoffmann. A heuristic for domain independent planning and its use in a fast greedy planning algorithm. Technical Report 133, Albert-Ludwigs-University Freiburg, 2000.

[6] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence*, pages 1194–1201, 1996.

[7] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 318–325, 1999.

[8] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proceedings of the 4th European Conference on Planning*, pages 273–285, 1997.

[9] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[10] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems*, pages 142–149, 1996.

[11] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *Proceedings of the 4th European Conference on Planning*, pages 338–350, 1997.

[12] I. Refanidis and I. Vlahavas. GRT: A domain independent heuristic for strips worlds based on greedy regression tables. In *Proceedings of the 5th European Conference on Planning*, pages 346–358, 1999.