FF: The Fast-Forward Planning System

Jörg Hoffmann
Institute for Computer Science
Albert Ludwigs University
Georges-Köhler-Allee, Geb. 52
79110 Freiburg, Germany
hoffmann@informatik.uni-freiburg.de

Abstract

Fast-Forward, abbreviated FF, was the most successful automatic planner in the AIPS-2000 planning systems competition. Like the well known HSP system, FF relies on forward search in the state space, guided by a heuristic that estimates goal distances by ignoring delete lists. It differs from HSP in a number of important details. This article describes the algorithmic techniques used in FF in comparison to HSP, and evaluates their benefits in terms of runtime and solution length behavior.

Introduction

Fast-Forward, abbreviated FF, was the most successful automatic planner in the AIPS-2000 planning systems competition. Though its performance clearly distinguished it from the other planners, the idea behind the approach is not new to the planning community. In fact, the basic principle is that of the HSP system, first introduced by Bonet et al. (Bonet, Loerincs, & Geffner 1997). Planning problems are attacked by forward search in state space, guided by a heuristic function that is automatically extracted from the domain description. To arrive at such a function, both planning systems *relax* the planning problem by ignoring parts of its specification, i.e., the delete lists of all actions.

FF can be seen as an advanced successor of the HSP system, which differs from its predecessor in a number of important details.

1. A more sophisticated method for heuristic evaluation, taking into account positive interactions between facts.
2. A novel kind of local search strategy, employing systematic search for escaping plateaus and local minima.
3. A method that identifies those successors of a search node that seem to be—and usually are—most helpful in getting to the goal.

We describe those methods in the subsequent sections. Afterwards, we overview the results of an empirical investigation determining which of the techniques yields which benefits in terms of runtime and solution length performance. We reflect on an experiment we have made, and, starting from this, outline the avenue of research we are currently focussing on.

Heuristic

Trying to attack domain independent planning as heuristic search, the main difficulty lies in the automatic derivation of the heuristic function. For human algorithm designers, a common approach to deriving a heuristic is to *relax* the problem $\mathcal{P}$ at hand into a simpler problem $\mathcal{P}'$, which can be solved efficiently. Facing a search state in $\mathcal{P}$, one can then use the solution length of the same state in $\mathcal{P}'$ to estimate its difficulty.

Bonet et al. (Bonet, Loerincs, & Geffner 1997) have proposed a way of applying this idea to domain independent planning. They relax the high-level problem description by simply ignoring delete lists. In the relaxed problem, all actions only add new atoms to the state, but don't remove any. During the execution of a relaxed action sequence, states thus only grow, and the problem is solved as soon as each goal has been added by some action. Let us illustrate this. Say we have an action that moves a robot from some point A to another point B. The precondition contains a fact stating that the robot needs to be at location A for the action to be applicable. After applying the action, the add list produces a fact stating the robot stands at location B, and the delete list removes the fact stating it stands at A. In the relaxation, the delete is ignored, so the precondition fact is not removed—after executing the relaxed action, the robot is located at A and B simultaneously. In a similar fashion, a relaxed planner can solve the n-discs Tower-of-Hanoi problem in n steps, and simultaneously assign the truth values TRUE and FALSE to a variable in a Boolean satisfiability problem. Nevertheless, the relaxation can be used to derive heuristics that are quite informative on a lot of benchmark planning problems.

The length of an *optimal* relaxed solution is an admissible—underestimating—heuristic, which could, theoretically, be used to find optimal solution plans by applying the A^* algorithm. However, computing the length of an optimal relaxed solution is NP-hard (Bylander 1994). Considering this, Bonet et al. (Bonet, Loerincs, & Geffner 1997) introduced the following way of *approximating* relaxed solution length from a search state S, based on computing *weight* values for all facts, which estimate their distance to S. First, initialize $weight(f) := 0$ for all facts $f \in S$, and $weight(f) := \infty$ for all others. Then apply all actions. For each action with preconditions $\mathrm{pre}(o)$ that adds a fact f, update the weight of f to

$$weight(f) := min(weight(f), weight(\mathrm{pre}(o)) + 1)$$

To determine the weight of an action's preconditions, one needs to define the weight of a *set* of facts. Bonet et al. assume facts to be achieved independently.

$$weight(F) := \sum_{f \in F} weight(f)$$

The updates are iterated until weight values don't change anymore. The difficulty of the state is then estimated as

$$h_{HSP}(S) := weight(\mathcal{G}) = \sum_{g \in \mathcal{G}} weight(g)$$

Here, $\mathcal{G}$ denotes the goal state of the problem at hand. The heuristic function obtained that way can be computed reasonably fast, and is often quite informative. Bonet and Geffner therefore used it in their first version of HSP, as it entered the AIPS-1998 planning competition.

The crucial observation leading to FF's heuristic method is this. While computing optimal relaxed solution length is NP-hard, deciding relaxed solvability is in P (Bylander 1994). Therefore, there exist polynomial-time decision algorithms. If such an algorithm constructs a witness, one can use that witness for heuristic evaluation. An algorithmic method that accomplishes this is the very well known Graphplan algorithm (Blum & Furst 1997). Started on a solvable relaxed problem, Graphplan finds a solution plan in polynomial time (Hoffmann & Nebel 2001). Facing a search state S, we therefore run a relaxed version of Graphplan starting out from S, and use the generated output for heuristic evaluation.

Relaxed Graphplan can be described as follows. First, build the planning graph until all goals are reached. The graph consists of alternating fact- and action-layers. The first fact layer is identical to S. The first action layer contains all actions that are applicable in S. The union of all those action's add effects with the facts that are already there forms the second fact layer. To this layer, again all actions are applied, and so on, until a fact layer is reached that contains all goals. This process corresponds quite closely to the computation of the weight values in HSP, as described above. Once the goals are reached, one can extract a relaxed plan in the following manner. Start at the top graph layer m, working on all goals. At each layer i, if a goal is present in layer $i-1$, then insert it into the goals to be achieved at $i-1$. Else, select an action in layer $i-1$ that adds the goal, and insert the action's preconditions into the goals at $i-1$. Once all goals at i are worked on, continue with the goals at $i-1$. Stop when the first graph layer is reached. The process results in a relaxed plan $\langle O_0, \dots, O_{m-1} \rangle$, where each O_i is the set of actions selected at time step i. We estimate solution length by counting the actions in that plan.

$$h_{FF}(S) := \sum_{i=0,\dots,m-1} |O_i|$$

The estimation values obtained this way are usually lower than HSP's estimates, as extracting a plan takes account of positive interactions between facts. Consider a planning problem where the initial state is empty, the goals are $\{G_1, G_2\}$, and there are the following three actions.

$\mathbf{op}G_1$: $P \Rightarrow$ ADD G_1
$\mathbf{op}G_2$: $P \Rightarrow$ ADD G_2
$\mathbf{op}P$: $\emptyset \Rightarrow$ ADD P

The meaning of the notation should be clear intuitively. HSP's heuristic estimate of the goal's distance to the initial state is four: Each single goal has weight two. The actions $\mathbf{op}G_1$ and $\mathbf{op}G_2$ share the precondition P, however. Relaxed plan extraction recognizes this, and selects $\mathbf{op}P$ only once, yielding a plan containing only *three* actions.

Search

While the heuristics presented in the preceding section can be computed in polynomial time, heuristic evaluation of states is still costly in the HSP as well as in the FF system. It is therefore straightforward to choose hill-climbing as the search method, in the hope to reach the goal by evaluating as few states as possible. HSP, in its AIPS-1998 version, used a common form of hill-climbing, where a best successor to each state was chosen randomly, and restarts took place whenever a path became too long. FF uses an *enforced* form of hill-climbing instead.

Facing a search state S, FF evaluates all of its direct successors. If none of those has a better heuristic value than S, it goes one step further, i.e., search then looks at the successor's successors. If none of those two-step successors looks better than S, FF goes on to the three-step successors, and so on. The process terminates when a state S' with better evaluation than S is found. The path to S' is then added to the current plan, and search continues with S' as the new starting state. In short, each search iteration performs complete breadth first search for a state with strictly better evaluation. If a planning problem does not contain dead end situations, then this strategy is guaranteed to find a solution (Hoffmann & Nebel 2001).

It has been recognized in the SAT community that the behavior of a local search method depends crucially on the structure of the problem it is trying to solve (Frank, Cheeseman, & Stutz 1997). Important features here are the number and distribution of solutions, as well as the size of local minima and plateaus. Our observation is that plateaus and local minima, when evaluating states with FF's or HSP's heuristic, tend to be small in many benchmark planning problems. It is therefore an adequate approach trying to find an exit state to such regions by complete breadth first search. We come back to this later.

Helpful Actions

The relaxed plan that FF extracts for each search state can not only be used to estimate goal distance, but also to identify the successors that seem to be most useful, and to detect goal ordering information (Hoffmann & Nebel 2001). Here, we explain the identification of a set of useful successors, generated by what we call the *helpful actions*. Consider the following small example, taken from the *Gripper* domain, as it was used in the AIPS-1998 competition. There are two rooms, A and B, and two balls, which shall be moved from room A to room B, using a robot. The robot changes rooms via the **move** operator, and controls two grippers which can **pick** or **drop** balls. Say the robot is in room A, and has picked up both balls. The relaxed solution that our heuristic extracts is this.

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

This is a relaxed plan consisting of two action sets. Looking at the first set yields our set of helpful actions: moving to room B is the only action that makes sense in the situation at hand. The two other applicable actions drop balls into room A, which is useless. To the human solver, this is obvious. It can automatically be detected by restricting any state's successors to those generated by the first action set in its relaxed solution. However, this is too restrictive in some cases. To a search state S, we therefore define the set $H(S)$ of helpful actions as follows.

$$H(S) := \{o \mid \text{pre}(o) \subseteq S, \text{add}(o) \cap G_1 \neq \emptyset\}$$

Here, G_1 denotes the set of goals that relaxed plan extraction constructs one level ahead of the initial graph layer. We thus consider as helpful those applicable actions that add at least one goal at the lowest layer of the relaxed solution. These are the actions that *could be* selected for the first set in the relaxed solution. The successors of any state S in breadth first search are then restricted to $H(S)$. While not completeness preserving, this approach works well in most of the current planning benchmarks. If enforced hill-climbing using that pruning technique fails to find a solution, we simply switch to a complete weighted A^* algorithm.

Performance Evaluation

A question of particular interest is, if FF is so closely related to HSP-1.0, then why does it perform so much better? We have conducted the following experiments to give an answer.

The three major differences of FF in comparison to HSP are relaxed plan extraction vs. weight values computation, enforced hill-climbing vs. hill-climbing, and helpful actions pruning vs. no such pruning technique. We have implemented experimental code where each of these differences is attached to a switch, which can be turned *on* or *off*. This yields eight planners, where *(off,off,off)* is an imitation of HSP-1.0, and *(on,on,on)* corresponds to FF. Each of these planners was run on a large set of benchmark planning problems, taken from 20 different domains. The collected data was then examined with the intention of assessing the impact that each single switch has on performance. For a detailed description, we refer the reader to our longer article (Hoffmann & Nebel 2001). Here, we overview the results. Data is subdivided into three parts, where we vary on each single switch in turn, keeping the others fixed.

FF Distance Estimates versus HSP Distance Estimates

Have a look at Figure 1. There are three tables, each one corresponding to a single switch. The four columns in each table stand for the four alignments of the respective other switches. In each column, the alignment's behavior with one setting of the table's switch is compared to the behavior with the other setting. Entries in a row show the number of planning domains in our test suite, where the corresponding setting of the switch leads to significantly better performance than the other setting, in terms of running time, and in terms of solution length.

Distance Estimate	Hill-climbing				Enforced Hill-climbing			
	All Actions		Helpful Actions		All Actions		Helpful Actions	
	time	length	time	length	time	length	time	length
HSP distance	2	2	1	2	2	0	1	0
FF distance	12	2	12	5	11	9	9	11

Search Strategy	All Actions				Helpful Actions			
	HSP distance		FF distance		HSP distance		FF distance	
	time	length	time	length	time	length	time	length
Hill-climbing	5	1	9	1	3	2	1	2
Enforced Hill-climbing	9	8	8	10	16	6	16	9

Pruning Technique	Hill-climbing				Enforced Hill-climbing			
	HSP distance		FF distance		HSP distance		FF distance	
	time	length	time	length	time	length	time	length
All Actions	2	0	3	0	2	1	2	0
Helpful Actions	13	7	14	8	15	5	15	3

Figure 1: Comparison of related planners when varying on goal distance estimates, search strategies, or pruning technique, from top to bottom. Performance is compared in terms of number of domains in our 20-domain test suite where one alternative leads to significantly better performance than the other one.

Let us focus on the topmost table, comparing the behavior of HSP goal distance estimates to that of FF estimates, when they are used by four different planners, obtained from aligning the other switches. The time entries in the leftmost column, for example, tell us that in 2 of our 20 planning domains hill-climbing without helpful actions had shorter running times when using HSP estimates than it did when using FF estimates. On the other hand, the alingment succeeded faster with FF estimates in 12 of the cases. In the remaining domains, both estimates lead to roughly the same runtime performance. We observe:

1. FF's estimates improve runtime performance in around half of our domains across all switch alignments.

2. With enforced hill-climbing in the background, FF's estimates have clear advantages in terms of solution length.

We remark that, in many of the domains with improved runtime performance, FF's estimates improve runtime across all our problem instancess reliably, but only by a small amount (Hoffmann & Nebel 2001). In some domains, however, HSP's heuristic overestimates goal distances quite drastically because it ignores positive interactions. In those domains, FF's estimates yield clear advantages.

For the second observation, we have no good explanation. It seems that the greedy way in which enforced hill-climbing builds its plans is just better suited when distance estimates are cautious, i.e., low.

Enforced Hill-climbing versus hill-climbing

Consider the table in the middle of Figure 1, comparing all combinations of the estimates and pruning technique switches, when used by hill-climbing in contrast to usage by enforced hill-climbing. We observe the following:

1. Without helpful actions in the background, enforced hill-climbing degrades performance almost as many times as it improves it, but with helpful actions, enforced hill-climbing is faster in 16 of our 20 domains.

2. Enforced hill-climbing often finds better solutions.

Whether one or the other search strategy is adequate depends on the domain. The advantage of enforced hill-climbing when helpful actions are in the background is due to the kind of interaction that the pruning technique has with the different search strategies. In hill-climbing, helpful actions saves running time proportional to the length of the paths encountered. In the enforced method, it cuts the branching factor during breadth first search, yielding *exponential* savings.

When enforced hill-climbing enters a plateau in the search space, it performs complete search for an exit, and adds the shortest path to that exit to its current plan prefix. When hill-climbing enters a plateau, on the other hand, it strolls around more or less randomly, until it hits an exit state. All the actions on its journey to that state are kept in the final plan. The former search method therefore often finds shorter plans than the latter.

Helpful Actions versus All Actions

We finally focus on the bottom table in Figure 1. It comprises one column for each variation of distance estimate and search strategy, comparing the behavior with helpful actions pruning to that without. Our observations are the following:

1. Helpful actions pruning improves runtime performance significantly in about three out of four of our domains across all switch alignments.
2. Only in one single domain is there a significant increase in solution length when one turns on helpful actions pruning.

On the 20 domains from our test suite, there is quite some variation with respect to the degree of restriction that helpful actions pruning exhibits. At the lower side of the scale, 5% of any state's successors are not considered helpful, while at the upper side, that percentage rises to 99%, i.e., only one out of a hundred successors is considered helpful there. In two domains from the middle of the scale, the restriction is inadequate, i.e., solutions get cut out of the state space. A moderate degree of restriction already leads to significantly improved runtime behavior. This is especially the case for enforced hill-climbing.

Our second observation strongly indicates that the actions that really lead towards the goal are usually considered helpful. Looking at Figure 1, there are some domains where solution length even *decreases* by not looking at all successors, especially when solving problems by hill-climbing. When that search strategy enters a plateau, it can only stroll around randomly in the search for an exit. If the method is additionally focussed into the direction of the goals, by helpful actions pruning, finding that exit might well take less steps.

Outlook

Put short, FF is a simple but effective algorithmic method, at least for solving the current planning benchmarks. An intuition is that those benchmarks are often quite simple in structure, and that it is this simplicity which makes them solvable so fast by such a simple algorithm as FF. To corroborate this, we ran FF on a set of problems with a more complicated search space structure. We generated random SAT instances according to the fixed clause-length model with 4.3 times as many clauses as variables (Mitchell, Selman, & Levesque 1992), and translated them into a PDDL encoding. The instances have a growing number of variables, from 5 up to 30. We ran the three planners FF, IPP, and Blackbox on those planning problems. In contrast to the behavior observed on almost any of the classical planning benchmarks, FF was clearly outperformed by the two other approaches. Typically, it immedeately found its way down to a state with only few unsatisfied clauses, and then got lost in the large local minimum it was in, which simply couldn't be escaped by systematic search. The other planners did much better due to the kind of inference algorithms they employ, which can rule out many partial truth assignments quite early.

Following (Frank, Cheeseman, & Stutz 1997), we have investigated the state space structures of the planning benchmarks, collecting empirical data about the density and size of local minima and plateaus. This has lead us to a taxonomy for planning domains, dividing them by the degree of complexity that the respective task's state spaces exhibit with respect to relaxed goal distances. Most of the current benchmark domains apparently belong to the "simpler" parts of that taxonomy (Hoffmann 2001). We also approach our hypotheses from a theoretical point of view, where we measure the degree of interaction that facts in a planning task exhibit, and draw conclusions on the search space structure from that. Our goal in that research is to devise a method that automatically decides which part of the taxonomy a given planning task belongs to.

Acknowledgements
The author wishes to thank Bernhard Nebel for discussions and continued support. Thanks also go to Blai Bonet and Hector Geffner for their help in comparing FF with HSP.

References

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

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, 714–719. MIT Press.

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

Frank, J.; Cheeseman, P.; and Stutz, J. 1997. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research* 7:249–281.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. To appear in *Journal of Artificial Intelligence Research.*

Hoffmann, J. 2001. Local search topology in planning benchmarks: An empirical analysis. To appear in *Proc. IJCAI-01*. Seattle, Washington, USA: Morgan Kaufmann.

Mitchell, D.; Selman, B.; and Levesque, H. J. 1992. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, 459–465. San Jose, CA: MIT Press.