AI Magazine Volume 22 Number 3 (2001) (© AAAI)

FF
The Fast-Forward Planning System

Jörg Hoffmann

■ FAST-FORWARD (FF) was the most successful automatic planner in the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00) 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 run-time and solution-length behavior.

FAST-FORWARD, abbreviated FF, was the most successful automatic planner in the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00) planning systems competition. Although 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, Loerincs, and 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, that is, 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: First, it has a more sophisticated method for heuristic evaluation, taking into account positive interactions between facts. Second, it uses a novel kind of local search strategy, employing systematic search for escaping plateaus and local minima. Third, it uses a method that identifies those successors of a search node that seem to be—and usually are—most helpful in getting to the goal.

I describe these methods in the subsequent sections. Afterwards, I overview the results of an empirical investigation determining which of the techniques yields which benefits in terms of run-time and solution-length performance. I reflect on an experiment that I have made and outline the avenue of research that I am currently focusing on.

Heuristic

In 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 $\mathcal{P}'$ that 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, Loerincs, and Geffner (1997) 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 only grow, and the problem is solved as soon as each goal has been added by some action. To illustrate, 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.

… 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.*

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 that 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, Loerincs, and Geffner (1997) introduced the following way of approximating a 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 $pre(o)$ that adds a fact *f*, update the weight of *f* to

$$weight(f) := min(weight(f), weight(pre(o)) + 1)$$

To determine the weight of an action's preconditions, one needs to define the weight of a set of facts. Bonet et al. (1997) 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

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

Here, *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 when it entered the AIPS'98 planning competition.

The crucial observation leading to FF's heuristic method is that although computing optimal relaxed solution length is NP hard, deciding relaxed solvability is in P (Bylander 1994). Therefore, polynomial-time decision algorithms exist. If such an algorithm constructs a witness, one can use this witness for heuristic evaluation. An algorithmic method that accomplishes this is the very well-known GRAPHPLAN algorithm (Blum and Furst 1997). Started on a solvable relaxed problem, GRAPHPLAN finds a solution plan in polynomial time (Hoffmann and Nebel 2001). Facing a search state *S*, I 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 add effects of these actions 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 earlier. 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 $\langle O_0, \ldots, O_m-1\rangle$, 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.

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

The estimation values obtained this way are usually lower than HSP's estimates because 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.

opG_1: $P \Rightarrow$ ADD G_1
opG_2: $P \Rightarrow$ ADD G_2
op$P$: $\varnothing \Rightarrow$ 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 **op**G_1 and **op**G_2 share the precondition *P*, however. Relaxed plan extraction recognizes this, and selects **op**P only once, yielding a plan containing only three actions.

Search

Although 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 of reaching the goal by evaluating as few states as possible. HSP, in its AIPS'98 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 its direct successors. If none of these successors has a better heuristic value than S, it goes one step further; that is, search then looks at the successor's successors. If none of these 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 and 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, and Stutz 1997). Important features here are the number and distribution of solutions as well as the size of local minima and plateaus. My 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. I come back to this later.

Helpful Actions

The relaxed plan that FF extracts for each search state cannot 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 and Nebel 2001). Here, I explain the identification of a set of useful successors, generated by what I call the *helpful actions*. Consider the following small example, taken from the gripper domain, as it was used in the AIPS'98 competition. There are two rooms, A and B, and two balls, which will be moved from room A to room B, using a robot. The robot changes rooms using the move operator and controls two grippers that 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

< { 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, a useless action, which, to the human solver, 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. Thus, we can 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 $H(S)$. Although not completeness preserving, this approach works well in most of the current planning benchmarks. If enforced hill climbing using this 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, then why does it perform so much better? To give an answer, I conducted the following experiment.

The three major differences between FF and HSP 1.0 are relaxed plan extraction versus weight value computation, enforced hill climbing versus hill climbing, and helpful action pruning versus no such pruning technique. I have implemented experimental code where each of these differences is attached to a switch, which can be turned on or off. Thus, I have 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

… 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.

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*
HSP Distance	2	0	3	0	2	1	2	0
FF Distance	13	7	14	8	15	5	15	3

Table 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.

data were then examined to assess the impact that each single switch has on performance. For a detailed description, I refer the reader to Hoffmann and Nebel (2001). Here, I overview the results. Data are subdivided into three parts, where I vary on each single switch, in turn, keeping the others fixed.

FF Distance Estimates versus HSP Distance Estimates

Look at table 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 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 solution length.

Let us focus on the topmost portion of the 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. However, the alignment succeeded faster with FF estimates in 12 of the cases. In the remaining domains, both estimates lead to roughly the same run-time performance.

We can make two observations: First, FF's estimates improve run-time performance in about half of our domains across all switch alignments. In many of the domains with improved run-time performance, FF's estimates improve run time across all our problem instances reliably but only by a small amount (Hoffmann and Nebel 2001). In some domains, however, HSP's heuristic overestimates goal distances quite drastically because it ignores positive interactions. In these domains, FF's estimates yield clear advantages.

Second, with enforced hill climbing in the background, FF's estimates have clear advan-

tages in terms of solution length. However, I 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, that is, low.

Enforced Hill Climbing versus Hill Climbing

Consider the middle portion of table 1, comparing all combinations of the estimates and pruning technique switches when used by hill climbing versus enforced hill climbing. Observe the following:

First, 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. Second, 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 the result of the kind of interaction that the pruning technique has with the different search strategies. In hill climbing, helpful actions save running time proportional to the length of the paths encountered. In the enforced method, helpful actions cut 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 this exit to its current plan prefix. When hill climbing enters a plateau, however, it strolls around more or less randomly until it hits an exit state. All the actions on its journey to this state are kept in the final plan. Enforced hill climbing, therefore, often finds shorter plans than hill climbing.

Helpful Actions versus All Actions

Now we focus on the bottom portion of table 1. It comprises one column for each variation of distance estimate and search strategy, comparing the behavior with helpful actions pruning to those without. I observe the following:

First, helpful action pruning improves runtime performance significantly in about three of four of our domains across all switch alignments. Second, only in one single domain is there a significant increase in solution length when one turns on helpful action pruning.

On the 20 domains from our test suite, there is quite some variation with respect to the degree of restriction that helpful action pruning exhibits. At the lower side of the scale, 5 percent of any state's successors are not considered helpful, but at the upper side, this percentage rises to 99 percent; that is, only 1 of 100 successors is considered helpful there. In two domains from the middle of the scale, the restriction is inadequate; that is, solutions get cut out of the state space. A moderate degree of restriction already leads to significantly improved run-time behavior, which is especially the case for enforced hill climbing.

The second observation strongly indicates that the actions that really lead toward 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 this search strategy enters a plateau, it can only stroll around randomly in the search for an exit. If the method is additionally focused into the direction of the goals, by helpful actions pruning, finding this exit might well take fewer steps.

Outlook

Briefly, FF is a simple but effective algorithmic method, at least for solving the current planning benchmarks. My intuition is that these benchmarks are often quite simple in structure and that it is this simplicity that makes them solvable so fast by such a simple algorithm as FF. To corroborate this conclusion, I ran FF on a set of problems with a more complicated search-space structure. I generated random SAT instances according to the fixed clause-length model with 4.3 times as many clauses as variables (Mitchell, Selman, and Levesque 1992) and translated them into a PDDL encoding. The instances have a growing number of variables, from 5 to 30. I ran the three planners, FF, IPP, and BLACKBOX on these 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 immediately found its way down to a state with only a 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 because of the kind of inference algorithms they use, which can rule out many partial truth assignments quite early.

Following Frank, Cheeseman, and Stutz (1997), I investigated the state-space structures of the planning benchmarks, collecting empirical data about the density and size of local minima and plateaus. This investigation has led to a taxonomy for planning domains,

… FF is a simple but effective algorithmic method, at least for solving the current planning benchmarks. My intuition is that these benchmarks are often quite simple in structure and that it is this simplicity that makes them solvable so fast by such a simple algorithm as FF.

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 this taxonomy (Hoffmann 2001). I also approach my intuitions from a theoretical point of view, where I measure the degree of interaction that facts in a planning task exhibit, and draw conclusions on the search space structure from that. The goal in this research is to devise a method that automatically decides which part of the taxonomy a given planning task belongs to.

Acknowledgments

I want 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 Proceedings of the Fourteenth National Conference on Artificial Intelligence, 714–719. Menlo Park, Calif.: American Association for Artificial Intelligence.

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. 2001. Local Search Topology in Planning Benchmarks: An Empirical Analysis. Paper presented at the Seventeenth International Joint Conference on Artificial Intelligence, 4–10 August, Seattle, Washington.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Mitchell, D.; Selman, B.; and Levesque, H. J. 1992. Hard and Easy Distributions of SAT Problems. In Proceedings of the Tenth National Conference on Artificial Intelligence, 459–465. Menlo Park, Calif.: American Association for Artificial Intelligence.

Jörg Hoffmann received a diploma in computer science from Freiburg University in March 1999. From January 1997 until September 1999, he was a member of the IPP project headed by Jana Koehler. Since April 1999, he has been working on a Ph.D. in planning as heuristic search, supervised by Bernhard Nebel. He was a member of the graduate program on "Human and Machine Intelligence" in Freiburg from April 1999 until December 2000 and is now working on a project financed by DFG. His e-mail address is hoffmann@informatik-uni.freiburg.de.