

The FF Planning System

Jorge A. Baier

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

CSC2542 – Topics in Knowledge Representation and Reasoning



The Fast Forward (FF) Planning System

- Was proposed by Hoffmann & Nebel (2001).
- Was the winner of the 2000 planning competition.
- Its novel elements are the following:
 - Heuristic based on *relaxed plans*.
 - Enforced Hill Climbing Used as the Search Strategy.
- Its core ideas have had substantial impact.

The Relaxed Plan Heuristic: Basic Definitions

Definition (STRIPS planning problem)

Let $P = \langle \text{Init}, \text{Ops}, \text{Goal} \rangle$ be a STRIPS planning problem where:

- Init is the initial state.
- Goal is the goal condition.
- Each $o \in \text{Ops}$ of the form $o = (\text{prec}(o), \text{add}(o), \text{del}(o))$

Definition (Delete-Relaxation)

The delete relaxation of P , denoted P^+ , is a instance just like P but in which operators in Ops have an **empty** delete list.

Definition (Relaxed Plan)

A *relaxed plan* for P is any plan for P^+ .

Computing a Relaxed Plan

For a planning state s :

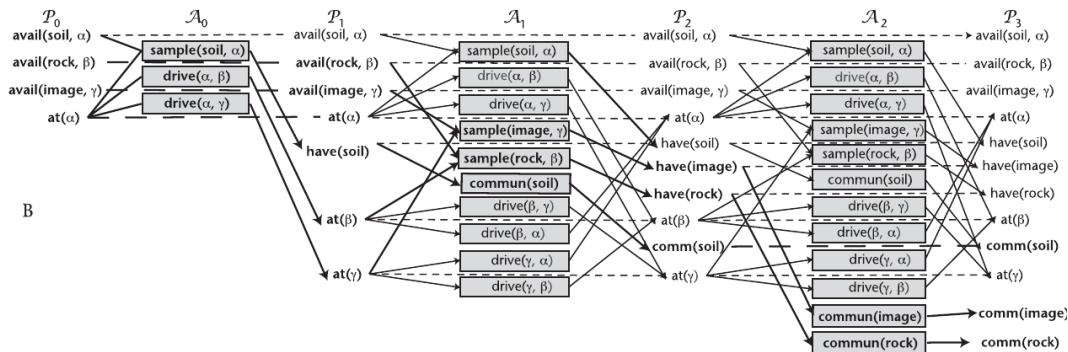
$$h_{FF}(s) = \text{“number of actions in a relaxed plan from } s\text{”}$$

The relaxed plan computed by FF:

- Is obtained using a version of Graphplan on P^+ .
- Is not a shortest relaxed plan (since this is already NP-hard).

Computing a Relaxed Plan: Intuition

For the general picture, we use Bryce & Kambhampati's figure:



Highlights of the relaxed plan extraction algorithm:

- Plan is extracted by regressing the goals (i.e. backwards)
- Iterates from the highest to the lowest level.
- Earliest achievers are always preferred.

Computing a Relaxed Plan: Algorithm

Extraction algorithm (Hoffmann & Nebel, 2001)

```

1: function EXTRACTPLAN(plan graph  $P_0A_0P_1 \dots A_{n-1}P_n$ , goal  $G$ )
2:   for  $i = n \dots 1$  do
3:      $G_i \leftarrow$  goals reached at level  $i$ 
4:   end for
5:   for  $i = n \dots 1$  do
6:     for all  $g \in G_i$  not marked TRUE at time  $i$  do
7:       Find min-cost  $a \in A_{i-1}$  such that  $g \in add(A_{i-1})$ 
8:        $RP_{i-1} \leftarrow RP_{i-1} \cup \{a\}$ 
9:       for all  $f \in prec(a)$  do
10:         $G_{layerof(f)} = G_{layerof(f)} \cup \{f\}$ 
11:      end for
12:      for all  $f \in add(a)$  do
13:        mark  $f$  as TRUE at times  $i - 1$  and  $i$ .
14:      end for
15:    end for
16:  end for
17:  return RP
18: end function

```

“Min-Cost” Actions

The “min-cost” action referred to in line 7 is the one that minimizes the following function:

$$\text{Cost}(a) = \sum_{p \in \text{Prec}(a)} \text{level}(p),$$

where $\text{level}(p)$ is the first layer at which p appears, and $\text{Prec}(a)$ are the preconditions of a .

Helpful Actions

Helpful actions are essential for FF’s performance. Helpful actions are those that appear at the first level of the relaxed plan.

Definition (Helpful action)

An action a of a relaxed plan from s is *helpful* iff it is a member of RP_0 .

Note that helpful actions are a *subset* of the actions executable in s .

Enforced Hill Climbing

Enforced Hill Climbing (EHC) (Hoffmann & Nebel, 2001)

```
1: function EHC(initial state  $I$ , goal  $G$ )
2:    $plan \leftarrow EMPTY$ 
3:    $s \leftarrow I$ 
4:   while  $h(s) \neq 0$  do
5:     from  $s$ , search for  $s'$  such that  $h(s') < h(s)$ .
6:     if no such state is found then
7:       return fail
8:     end if
9:      $plan \leftarrow plan \circ$  "actions on the path to  $s'$ "
10:     $s \leftarrow s'$ 
11:  end while
12:  return  $plan$ 
13: end function
```

Breadth-First Search for a New State

The breadth-first search (line 5) from s is implemented as follows:

```
1:  $queue \leftarrow$  empty-queue
2:  $closed \leftarrow$  {states visited by the plan}
3: PUSH(queue, {helpful successors of  $s$ })
4: while  $queue$  is not empty do
5:    $t \leftarrow pop(queue)$ 
6:   if  $t \in closed$  then
7:     continue ▷ discard  $t$  and continue the iteration
8:   end if
9:   if  $h(t) < h(s)$  then
10:     $s' \leftarrow t$ 
11:    break ▷ better state found, exit loop
12:  end if
13:  PUSH(queue, {helpful successors of  $s$ })
14:   $closed \leftarrow closed \cup \{t\}$ 
15: end while
```

FF's search strategy

EHC is an **incomplete** search algorithm and thus prone to failure. If EHC fails, FF falls back into *best-first search* (A search), in which the evaluation function for a state is:

$$f(s) = h_{FF}(s)$$

Note that this search is complete but greedy since the length of the plan is not considered.

Now let's see how FF works in practice !

References I

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