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.
1 Compile problem into grounded STRIPS.
2 Perform Enforced-Hill-Climbing (EHC) until either solved or no further progress can be made.
   - Sound, not complete.
3 Perform Best-First-Search
   - Sound, complete.
**The Relaxed Plan Heuristic: Basic Definitions**

<table>
<thead>
<tr>
<th>Definition (STRIPS planning problem)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $P = \langle \text{Init}, \text{Ops}, \text{Goal} \rangle$ be a STRIPS planning problem where:</td>
</tr>
<tr>
<td>- \text{Init} is the initial state.</td>
</tr>
<tr>
<td>- \text{Goal} is the goal condition.</td>
</tr>
<tr>
<td>- Each $o \in \text{Ops}$ of the form $o = (\text{prec}(o), \text{add}(o), \text{del}(o))$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition (Delete-Relaxation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The delete relaxation of $P$, denoted $P^+$, is a instance just like $P$ but in which operators in $\text{Ops}$ have an empty delete list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition (Relaxed Plan)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A relaxed plan for $P$ is any plan for $P^+$.</td>
</tr>
</tbody>
</table>
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 \textsc{ExtractPlan}(plan graph } P_0 A_0 P_1 \cdots A_{n-1} P_n, \text{ goal } G \)
2: \hspace{1em} for } i = n \ldots 1 \text{ do}
3: \hspace{2em} G_i \leftarrow \text{goals reached at level } i
4: \hspace{1em} end for
5: \hspace{1em} for } i = n \ldots 1 \text{ do}
6: \hspace{2em} for all } g \in G_i \text{ not marked TRUE at time } i \text{ do}
7: \hspace{3em} \text{Find min-cost } a \in A_{i-1} \text{ such that } g \in add(A_{i-1})
8: \hspace{2em} RP_{i-1} \leftarrow RP_{i-1} \cup \{a\}
9: \hspace{2em} for all } f \in \text{prec}(a) \text{ do}
10: \hspace{3em} \text{G}_{\text{layerof}(f)} = \text{G}_{\text{layerof}(f)} \cup \{f\}
11: \hspace{2em} end for
12: \hspace{2em} for all } f \in \text{add}(a) \text{ do}
13: \hspace{3em} \text{mark } f \text{ as TRUE at times } i - 1 \text{ and } i.
14: \hspace{2em} end for
15: \hspace{1em} end for
16: \hspace{1em} end for
17: \hspace{1em} return RP
18: \hspace{1em} end function
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 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 (EHC) (Hoffmann & Nebel, 2001)

1: function EHC(initial state \( I \), goal \( G \))
2: \( \text{plan} \leftarrow \text{EMPTY} \)
3: \( s \leftarrow I \)
4: \( \text{while } h(s) \neq 0 \text{ do} \)
5: \( \text{from } s, \text{ search for } s' \text{ such that } h(s') < h(s). \)
6: \( \text{if no such state is found then} \)
7: \( \text{return fail} \)
8: \( \text{end if} \)
9: \( \text{plan} \leftarrow \text{plan} \circ \text{“actions on the path to } s'\text{”} \)
10: \( s \leftarrow s' \)
11: \( \text{end while} \)
12: \( \text{return } \text{plan} \)
13: \( \text{end function} \)
Breadth-First Search for a New State

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

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