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 Fast Forward (FF) Planning System: Approach

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

Definition (STRIPS planning problem)

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

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

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:

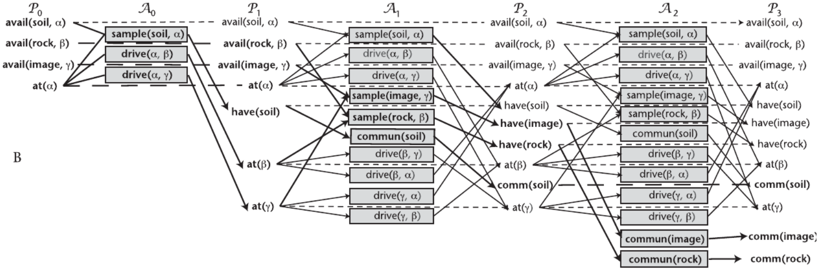
$h_{FF}(s) =$ **"number of actions in a relaxed plan from s"**

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)

function ExtractPlan(plan graph $P_0 A_0 P_1 \cdots A_{n-1} P_n$, goal G)
  for $i = n \ldots 1$ **do**
    $G_i \leftarrow$ goals reached at level i
  end for
  for $i = n \ldots 1$ **do**
    for all $g \in G_i$ not marked TRUE at time i **do**
      Find min-cost $a \in A_{i-1}$ such that $g \in add(A_{i-1})$
      $RP_{i-1} \leftarrow RP_{i-1} \cup \{a\}$
      for all $f \in prec(a)$ **do**
        $G_{layerof(f)} = G_{layerof(f)} \cup \{f\}$
      end for
      for all $f \in add(a)$ **do**
        mark f as TRUE at times $i-1$ and i.
      end for
    end for
  end for
  return RP
end function

“Min-Cost” Actions

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

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

where $level(p)$ is the first layer at which p appears, and $Prec(a)$ 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 if 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)

```
function EHC(initial state I, goal G)
    plan ← EMPTY
    s ← I
    while h(s) ≠ 0 do
        from s, search for s′ such that h(s′) < h(s).
        if no such state is found then
            return fail
        end if
        plan ← plan ∘ "actions on the path to s′"
        s ← s′
    end while
    return plan
end function
```

Breadth-First Search for a New State

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

```
queue ← empty-queue
closed ← {states visited by the plan}
PUSH(queue,{helpful successors of s})
while queue is not empty do
    t ← pop(queue)
    if t ∈ closed then
        continue                      ▷ discard t and continue the iteration
    end if
    if h(t) < h(s) then
        s′ ← t
        break                         ▷ better state found, exit loop
    end if
    PUSH(queue,{helpful successors of t})
    closed ← closed ∪ {t}
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.