Fast Forward Planning

By

J. Hoffmann and B. Nebel
Chronology

“Traditional”

≈ 1995

“Graphplan-based”

≈ 2000

“Satplan-based”

“Heuristic-based”

Optimal

Sub-optimal

faster
Fast Forward (FF)
Winner of AIPS2000

• Forward-chaining heuristic search planner
• Basic principle: Hill-climb through the space of problem states, starting at the initial state.
• Each child state results from apply a single plan operator.
• Always moves to the first child state found that is closer to the goal.
• Records the operators applied along the path.
• The operators leading to the goal constitute a plan.
FF’s Base System Structure

- **Task Specification**
  - **Enforced Hill-climbing**
    - **State**
    - **Goal Distance**
    - **Helpful Actions**
    - **Solution / "Fail"**
  - **Relaxed GRAPHPLAN**
FF Search Strategy

- FF uses a strategy called **enforced hill-climbing**:
  - Obtain heuristic estimate of the value of the current state.
  - Find action(s) transitioning to a better state.
  - Move to the better state.
  - Append actions to plan head.
  - Never backtrack over any choice.
Search: Enforced hill-climbing

- Plain hill-climbing
  - Randomly breaks ties and adds to the path
  - Can wander in plateaus before restarting

- Enforced hill-climbing
  - At a state, perform Breadth First (exhaustive) Search until a state with a better heuristic is found
  - Force search to a better position (if it exists)
  - Add path to that new state to the plan
Enforced Hill-Climbing (cont.)

• The success of this strategy depends on how informative the heuristic is.
  – FF uses a heuristic found to be informative in a large class of benchmark planning domains.

• The strategy is not complete.
  – Never backtracking means that some parts of the search space are lost.

• If FF fails to find a solution using this strategy it switches to standard Best First Search.
\[ h(S1) < h(S4) < h(\text{init}) < h(S2) < h(S3) < h(S5) = h(S6) \]

Maximize Utility \( h \)

Plan Head: A, B
Finding a better state: Plateaus

Perform breadth first search from the current state, to states reachable by action applications, stopping as soon as a strictly better one is found.
FF’s Heuristic Estimate

• The value of a state is a measure of how close it is to a goal state.
• This cannot be determined exactly (too hard), but can be approximated.
• One way of approximating is to solve a relaxed problem.
  – Relaxation is achieved by ignoring the negative effects of the actions.
  – The relaxed action set, $A'$, is defined by:
    \[ A' = \{<\text{pre}(a), \text{add}(a), 0> | a \in A\} \]
Building the Relaxed Plan Graph

• Start at the initial state.
• Repeatedly apply all relaxed actions whose preconditions are satisfied.
  – Assert their (positive) effects in the next layer.
• If all actions are applied and the goals are not all present in the final graph layer,
• Then the problem is unsolvable.
Extracting a Relaxed Soln

• When a layer containing all of the goals is reached, FF searches *backwards* for a plan.
• The first possible achiever found is always used to achieve each goal.
• The relaxed plan might contain many actions happening concurrently at a layer.
• The number of actions in the relaxed plan is an estimate of the true cost of achieving the goals.
Graph-based heuristic

- Problem: $P = <O,I,G>$
- Relaxed problem: $P' = <O',I,G>$
- To compute the heuristic for state $S$:
  - Create the planning graph of $<O',S,G>$ and find a solution
  - Let $O_i$ = set of actions taken at $i$ in plan
  - Then:
    $$h(S) = \sum_{i=0,...,m-1} |O_i|$$
    ... an estimate of the length of this solution
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
</table>
| Move from room x to room y | - pre: robot is in x, door open  
- add: robot is in y  
- del: robot is in x |
| Polish from room x to room y | - pre: door open  
- add: floor polished |
| Open door     | - pre: door is closed  
- add: door is open  
- del: door is closed |
| Close door    | - pre: door is open  
- add: door is closed  
- del: door is open |

**Initial State**
- In(A)  
- Closed

**Final State**
- In(B)  
- Closed  
- Polished
Distance Estimate Extracted From A Relaxed Plan Graph

- Current: In(A), Closed
- Goal: In(B)

Layers correspond to successive time points,
# layers indicate minimum time to achieve goals.
How FF Uses the Heuristic

- FF uses the heuristic to estimate how close each state is to a goal state
  - any state satisfying the goal propositions.
- The actions in the relaxed plan are used as a guide to which actions to explore when extending the plan.
  - All actions in the relaxed plan at the 1st layer that achieves at least one of the (sub) goals required at the 2nd layer are considered helpful.
- FF restricts attention to the helpful actions when searching forward from a state.
Distance Estimate Extracted From A Relaxed Plan Graph

- Current: In(A), Closed

- Goal: In(B)

- Useful actions: Open
Properties of the Heuristic

- The relaxed plan that is extracted is not guaranteed to be the optimal relaxed plan.
- The heuristic is not admissible.
  - FF can produce non-optimal solutions.
- Focusing only on helpful actions is not completeness preserving.
- Enforced hill-climbing is not completeness preserving.
Getting Out of Deadends

• Because FF does not backtrack, FF can get stuck in dead-ends.
• This arises when an action cannot be reversed, thus, having entered a bad state there is no way to improve.
• When no search progress can be made, FF switches to Best First Search from the initial state.
  – Detecting a dead-end can be expensive if the plateau is large.
Relaxed Graph

Node 1

In(A) Closed Estimate ?

Goal: In(B), Closed, Polished

In(A) Closed

In(A) Closed
Open

In(A) Closed
Open

In(A)

In(B)
Closed

Open

Polished

Move(A,B)

Close

noop

noop

noop

noop

Polish

Note: For simplicity, operator only drawn in first layer in which it is introduced.

Estimate: 3
• # actions in plan
Useful Actions: Open
• actions used in 1st layer, used to create children
Goal: In(B), Closed, Polished

Node 1: In(A) Closed
        ↓ Open
        Node 2: In(A) Opened

Relaxed Graph:

- In(A) Opened
- In(A) Closed
- In(B) Closed
- In(B) Opened

Estimate: 3 = Plateau
perform BFS

Useful Actions:
Move, Close, Polish
Goal: In(B), Closed, Polished

Node 1
In(A)
Closed
Open

Node 2
In(A)
Opened
Estimate 3

Move(A,B)

Node 3
In(B)
Opened
Estimate ?

Relaxed Graph

In(B)
Opened

In(B)
In(A)
Closed
Opened
Polished

Estimate: 2 = Off Plateau

Useful Actions:
Close, Polish
Goal: In(B), Closed, Polished

Node 1: In(A) Closed
   - Open
   - Estimate 3

Node 2: In(A) Opened
   - Move(A,B)
   - Close
   - Estimate 3

Node 3: In(B) Opened
   - Close
   - Polish
   - Estimate 2

Node 4: In(B) Closed

Relaxed Graph

Estimate: 2
No improvement
Goal: In(B), Closed, Polished

Node 1: In(A) Closed
Open
Node 2: In(A) Opened
Estimate 3
Move(A,B)
Close
Polish
Node 3: In(B) Opened
Estimate 2
Close
Polish
Node 4: In(B) Closed
Estimate 2
Node 5: In(B) Opened
Polished
Estimate ?

Relaxed Graph

Estimate: 1
Useful Actions:
Close
Goal: In(B), Closed, Polished

Plan: Open, Move(A,B), Polish, Close

Relaxed Graph

Estimate: 0
Runtime Curves on large Logistic instances
FF versus HSP

- FF and HSP are forward chaining planners that use a hill-climbing strategy based on relaxed distance estimates.
- FF uses a heuristic evaluation based on the number of actions in an explicit relaxed plan.
- HSP uses weight values which approximate (but overestimate) the length of a relaxed plan.
FF versus HSP

- FF uses a number of pruning heuristics that can be very powerful (especially in the simply structured propositional benchmark domains).
- FF terminates **Breadth First Search** for a successor state as soon as an improvement is found. HSP selects successors randomly from the set of best states.
- FF defaults to complete **Best First Search** from the current state if the enforced hill-climbing strategy fails.