FastForward: Heuristics For Planning

Chad Hogg

2007-02-27
Outline

1. State-Space Planning
   - Review
   - Vanillalce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion
Outline

1. State-Space Planning
   - Review
     - VanillaIce
     - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion
State-Space Representation

- Back to classical planning for a moment ...
State-Space Representation

- Back to classical planning for a moment ...
- A state $S$ is a set of ground literals.
Back to classical planning for a moment ...

A state $S$ is a set of ground literals.

An action $A = (h, p, e^+, e^-)$ consists of a head and 3 sets of literals – preconditions, positive effects, and negative effects.
Back to classical planning for a moment ...

A state $S$ is a set of ground literals.

An action $A = (h, p, e^+, e^-)$ consists of a head and 3 sets of literals – preconditions, positive effects, and negative effects.

The application of an action $A$ to a state $S$ is another state

$$\gamma(S, A) = (S - e^-) \cup e^+.$$
Forward Search in State Space

Forward search applies actions to an initial state until a goal state is reached.
Forward Search in State Space

- Forward search applies actions to an initial state until a goal state is reached.
- Depth-first search expands until dead-end, then backtracks.
Forward Search in State Space

- Forward search applies actions to an initial state until a goal state is reached.
- Depth-first search expands until dead-end, then backtracks
  - Sound, but not complete or admissible in general case
Forward Search in State Space

- Forward search applies actions to an initial state until a goal state is reached.
- Depth-first search expands until dead-end, then backtracks
  - Sound, but not complete or admissible in general case
- Breadth-first search checks all 1-action sequences, then all 2-action sequences, etc.
Forward Search in State Space

- Forward search applies actions to an initial state until a goal state is reached.
- Depth-first search expands until dead-end, then backtracks
  - Sound, but not complete or admissible in general case
- Breadth-first search checks all 1-action sequences, then all 2-action sequences, etc.
  - Sound, complete, and admissible
Forward Search in State Space

- Forward search applies actions to an initial state until a goal state is reached.
- Depth-first search expands until dead-end, then backtracks
  - Sound, but not complete or admissible in general case
- Breadth-first search checks all 1-action sequences, then all 2-action sequences, etc.
  - Sound, complete, and admissible
  - Very memory intensive
Forward Search in State Space

- Forward search applies actions to an initial state until a goal state is reached.
- Depth-first search expands until dead-end, then backtracks
  - Sound, but not complete or admissible in general case
- Breadth-first search checks all 1-action sequences, then all 2-action sequences, etc.
  - Sound, complete, and admissible
  - Very memory intensive
- Best-first, Iterative Deepening are specialized
Our Familiar Example

State-Space Planning
Heuristics
Hill-Climbing
Pruning

Review
Vanillalce
FastForward

FastForward: Heuristics For Planning
Outline

1. **State-Space Planning**
   - Review
   - VanillaIce
   - FastForward

2. **Heuristics**
   - Goal Distance
   - Graphplan Relation

3. **Hill-Climbing**
   - Standard Form
   - Enforced

4. **Pruning**
   - Helpful Actions
   - Goal Deletion
VanillasIce Properties

- Written by me last week, largely untested
Vanillalce Properties

- Written by me last week, largely untested
- A breadth-first, forward-searching state-space classical planner
Vanillalce Properties

- Written by me last week, largely untested
- A breadth-first, forward-searching state-space classical planner
- Sound, complete, and admissible (assuming no bugs)
Vanillalce Properties

- Written by me last week, largely untested
- A breadth-first, forward-searching state-space classical planner
- Sound, complete, and admissible (assuming no bugs)
- Much, much slower than state-of-the-art systems
VanillaIce Usage

- Binary is located at /home/cmh204/vanilla_ice/vanilla_ice1.0/vanilla_ice on vega.cc.lehigh.edu
VanillaIce Usage

- Binary is located at /home/cmh204/vanillaIce/vanillaIce1.0/vanillaIce on vega.cc.lehigh.edu
- Source is available upon request
Vanillalce Usage

- Binary is located at 
  /home/cmh204/vanilla_ice/vanilla_ice1.0/vanilla_ice on vega.cc.lehigh.edu
- Source is available upon request
- All relevant information and instructions are in README.txt file
VanillaIce Usage

- Binary is located at /home/cmh204/vanilla_ice/vanilla_ice1.0/vanilla_ice on vega.cc.lehigh.edu
- Source is available upon request
- All relevant information and instructions are in README.txt file
- Sample domain and problem input files in PDDL are provided
VanillaIce Usage

- Binary is located at 
  `/home/cmh204/vanilla_ice/vanilla_ice1.0/vanilla_ice` on 
  `vega.cc.lehigh.edu`
- Source is available upon request
- All relevant information and instructions are in `README.txt` file
- Sample domain and problem input files in PDDL are provided
- Demonstration ...
Outline

1. State-Space Planning
   - Review
   - Vanillalce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion

FastForward: Heuristics For Planning
Written by Jörg Hoffmann in 2000
FastForward

- Written by Jörg Hoffmann in 2000
- Winner of many contemporary planning competitions
FastForward:

- Written by Jörg Hoffmann in 2000
- Winner of many contemporary planning competitions
- Another breadth-first, forward-searching state-space classical planner
FastForward

- Written by Jörg Hoffmann in 2000
- Winner of many contemporary planning competitions
- Another breadth-first, forward-searching state-space classical planner
- Sound and complete but not admissible
FastForward

- Written by Jörg Hoffmann in 2000
- Winner of many contemporary planning competitions
- Another breadth-first, forward-searching state-space classical planner
- Sound and complete but not admissible
- Much faster than Vanillalce due to heuristics, hill-climbing, and pruning strategies
A heuristic in general is a procedure that often performs well but has no guarantees.
A heuristic in general is a procedure that often performs well but has no guarantees.

Usually involves solving a relaxed version of problem.
Heuristic Properties

- A *heuristic* in general is a procedure that often performs well but has no guarantees.
- Usually involves solving a relaxed version of problem.
- In search problems, a heuristic is an estimate of how likely a node is to quickly lead to a solution.
Heuristic Properties

- A *heuristic* in general is a procedure that often performs well but has no guarantees.
- Usually involves solving a relaxed version of problem.
- In search problems, a heuristic is an estimate of how likely a node is to quickly lead to a solution.
- Such a heuristic allows the most promising nodes to be explored first.
A *heuristic* in general is a procedure that often performs well but has no guarantees.

Usually involves solving a relaxed version of problem.

In search problems, a heuristic is an estimate of how likely a node is to quickly lead to a solution.

Such a heuristic allows the most promising nodes to be explored first.

Must be reasonably accurate and significantly cheaper to compute than solution to problem.
Outline

1 State-Space Planning
   • Review
   • Vanillalce
   • FastForward

2 Heuristics
   • Goal Distance
   • Graphplan Relation

3 Hill-Climbing
   • Standard Form
   • Enforced

4 Pruning
   • Helpful Actions
   • Goal Deletion
So what does this mean for forward-search state-space planning?
So what does this mean for forward-search state-space planning?

- How many actions must be applied to a state to achieve the goals
So what does this mean for forward-search state-space planning?

- How many actions must be applied to a state to achieve the goals
- Why not compute this exactly?
So what does this mean for forward-search state-space planning?
- How many actions must be applied to a state to achieve the goals
- Why not compute this exactly?
  - Only known way would give you the plan, meaning you have to solve the problem.
We shall use $h^*(s)$ to mean the actual minimum number of actions between a state and the goals.
Terminology

- We shall use \( h^*(s) \) to mean the actual minimum number of actions between a state and the goals
- \( \Delta(s, p) \) is an estimated minimum distance from state \( s \) to a state containing predicate \( p \)
Terminology

- We shall use $h^*(s)$ to mean the actual minimum number of actions between a state and the goals.
- $\Delta(s, p)$ is an estimated minimum distance from state $s$ to a state containing predicate $p$.
- $\Delta(s, g)$ is an estimated distance from state $s$ to set $g$ of predicates.
We shall use \( h^*(s) \) to mean the actual minimum number of actions between a state and the goals.

\( \Delta(s, p) \) is an estimated minimum distance from state \( s \) to a state containing predicate \( p \).

\( \Delta(s, g) \) is an estimated distance from state \( s \) to set \( g \) of predicates.

\( h(s) \) is an estimated heuristic value for state \( s \).
Relaxing Goal Distance

What makes this so computationally expensive?
What makes this so computationally expensive?

- Selecting actions is a combinatorial problem because of interactions and deletions among methods.
Relaxing Goal Distance

- What makes this so computationally expensive?
  - Selecting actions is a combinatorial problem because of interactions and deletions among methods
- How could we relax the problem but still have an informative heuristic?
  - Ignore negative effects of actions, so state is constantly expanding towards goals
Relaxing Goal Distance

- What makes this so computationally expensive?
  - Selecting actions is a combinatorial problem because of interactions and deletions among methods

- How could we relax the problem but still have an informative heuristic?
  - Ignore negative effects of actions, so state is constantly expanding towards goals
Another relaxation is assuming goals are achieved independently.
Another relaxation is assuming goals are achieved independently.

\[
\Delta_0(s, p) = \begin{cases} 
0, & p \in s \\
\min_a \{1 + \Delta_0(s, \text{prec}(a)) | p \in \text{eff}^+(a)\}, & p \notin s 
\end{cases}
\]
Relaxing Goal Independence

Another relaxation is assuming goals are achieved independently.

\[ \Delta_0(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\min_a \{1 + \Delta_0(s, \text{prec}(a)) | p \in \text{eff}^+(a)\}, & \text{if } p \notin s 
\end{cases} \]

Recursive backward count from predicate to state.
Another relaxation is assuming goals are achieved independently.

\[ \Delta_0(s, p) = \begin{cases} 0, & p \in s \\ \min_a \{1 + \Delta_0(s, \text{prec}(a)) \mid p \in \text{eff}^+(a)\}, & p \notin s \end{cases} \]

- Recursive backward count from predicate to state

\[ \Delta_0(s, g) = \sum_{p \in g} \Delta_0(s, p) \]
Relaxing Goal Independence

- Another relaxation is assuming goals are achieved independently

\[ \Delta_0(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\min_a \{1 + \Delta_0(s, \text{prec}(a)) | p \in \text{eff}^+(a)\}, & \text{if } p \notin s 
\end{cases} \]

- Recursive backward count from predicate to state

\[ \Delta_0(s, g) = \sum_{p \in g} \Delta_0(s, p) \]

- Simple summation of distances to each goal
Relaxing Goal Independence

- Another relaxation is assuming goals are achieved independently

\[ \Delta_0(s, p) = \begin{cases} 
0, & p \in s \\
\min_a \{1 + \Delta_0(s, \text{prec}(a))|p \in \text{eff}^+(a)\}, & p \notin s
\end{cases} \]

- Recursive backward count from predicate to state

\[ \Delta_0(s, g) = \sum_{p \in g} \Delta_0(s, p) \]

- Simple summation of distances to each goal

\[ h_0(s) = \Delta_0(s, g) \]
Related Heuristics

What if goals are not independent?
What if goals are not independent?

- $\Delta_1(s, g) = \max_{p \in g} \Delta_0(s, p)$
Related Heuristics

- What if goals are not independent?
  - $\Delta_1(s, g) = \max_{p \in g} \Delta_0(s, p)$
  - $\Delta_2(s, g) = \text{shortest path to most difficult pair of goals}$
Related Heuristics

What if goals are not independent?
- $\Delta_1(s, g) = \max_{p \in g} \Delta_0(s, p)$
- $\Delta_2(s, g) =$ shortest path to most difficult pair of goals
- $\Delta_k(s, g) =$ shortest path to most difficult $k$ goals
Related Heuristics

What if goals are not independent?

- \( \Delta_1(s, g) = \max_{p \in g} \Delta_0(s, p) \)
- \( \Delta_2(s, g) = \) shortest path to most difficult pair of goals
- \( \Delta_k(s, g) = \) shortest path to most difficult \( k \) goals

Higher-order heuristics are more informative but harder to compute
Outline

1. State-Space Planning
   - Review
   - VanillaIce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion
Graphplan As Heuristic

- Expanding into larger states disregarding negative effects should look familiar
Graphplan As Heuristic

- Expanding into larger states disregarding negative effects should look familiar
- Graphplan does nearly the same thing, including but marking negative effects
Graphplan As Heuristic

- Expanding into larger states disregarding negative effects should look familiar.
- Graphplan does nearly the same thing, including but marking negative effects.
- Building planning graph to search in works similarly to exploring with heuristics to guide search.
Graphplan In FastForward

- FastForward calls a special version of Graphplan to compute a heuristic value
Graphplan In FastForward

- FastForward calls a special version of Graphplan to compute a heuristic value
- Domain given to Graphplan contains no negative effects
FastForward calls a special version of Graphplan to compute a heuristic value.

- Domain given to Graphplan contains no negative effects.
  - Thus, no mutual exclusion among actions or literals.
Graphplan In FastForward

- FastForward calls a special version of Graphplan to compute a heuristic value.
- Domain given to Graphplan contains no negative effects.
  - Thus, no mutual exclusion among actions or literals.
  - Thus, no need for backtracking.
Graphplan In FastForward

- FastForward calls a special version of Graphplan to compute a heuristic value.
- Domain given to Graphplan contains no negative effects:
  - Thus, no mutual exclusion among actions or literals
  - Thus, no need for backtracking
- Graphplan always chooses no-ops first
FastForward calls a special version of Graphplan to compute a heuristic value.

Domain given to Graphplan contains no negative effects:
- Thus, no mutual exclusion among actions or literals
- Thus, no need for backtracking

Graphplan always chooses no-ops first:
- Thus, actions appear only once
FastForward calls a special version of Graphplan to compute a heuristic value.

Domain given to Graphplan contains no negative effects:
- Thus, no mutual exclusion among actions or literals
- Thus, no need for backtracking

Graphplan always chooses no-ops first:
- Thus, actions appear only once

Length of Graphplan’s solution to relaxed problem from a node is heuristic value for that node.
Relaxed Graphplan Complexity

- On relaxed problems, Graphplan runs in polynomial time in the number of actions, size of largest positive effects set, and size of initial state.
Relaxed Graphplan Complexity

- On relaxed problems, Graphplan runs in polynomial time in the number of actions, size of largest positive effects set, and size of initial state.
- Refer to the paper for proof
On relaxed problems, Graphplan runs in polynomial time in the number of actions, size of largest positive effects set, and size of initial state.

Refer to the paper for proof.

Building the planning graph sounds expensive, but if this polynomial-time algorithms restricts the exponential state space, it will be a win.
We take advantage of lack of mutual exclusion to use a compact notation for large planning graphs.
Example Notation

- We take advantage of lack of mutual exclusion to use a compact notation for large planning graphs.
- Literals are only listed in the first level where they are true.
Example Notation

- We take advantage of lack of mutual exclusion to use a compact notation for large planning graphs.
- Literals are only listed in the first level where they are true.
- Actions are only listed in the first level where they are applicable.
We take advantage of lack of mutual exclusion to use a compact notation for large planning graphs.

- Literals are only listed in the first level where they are true.
- Actions are only listed in the first level where they are applicable.
- Only new effects of actions are shown, in dashed lines.
Example

(ontable B)
(ontable C)
(on A B)
(clear A)
(clear C)
(hand-empty)
Example

(ontable B)
(ontable C)
(on A B)
(clear A)
(clear C)
(hand-empty)

unstack(A B)
pickup(C)
Example

(ontable B) unstack(A B) (clear B)
(ontable C) pickup(C) (holding A)
(on A B) (clear C) (holding C)
(clear A)
(clear C)
(hand-empty)
State-Space Planning
Heuristics
Hill-Climbing
Pruning

Example

(ontable B)
(ontable C)
(on A B)
(clear A)
(clear C)
(hand-empty)

unstack(A B) -> (clear B)
pickup(C) -> (holding A)

putdown(A)
putdown(C)
(stack(A B)
(stack(A C)
(stack(C A)
(stack(C B)
pickup(B)
Example

State-Space Planning
Heuristics
Hill-Climbing
Pruning

Goal Distance
Graphplan Relation

Chad Hogg
FastForward: Heuristics For Planning
Example
Example

State-Space Planning
Heuristics
Hill-Climbing
Pruning

Goal Distance
Graphplan Relation

Chad Hogg
FastForward: Heuristics For Planning
Example
Example

(State-Space Planning)
(Heuristics)
(Hill-Climbing)
(Pruning)

Goal Distance
(Graphplan Relation)

Chad Hogg

FastForward: Heuristics For Planning
Example

('ontable B')
('ontable C')
('on A B')
('clear A')
('clear C')
('hand-empty')

unstack(A B)
pickup(C)

('clear B')
('holding A')
('holding C')

putdown(A)
putdown(C)
stack(A B)
stack(A C)
stack(C A)
stack(C B)
pickup(B)

('ontable A')
('on A C')
('on C A')
('on C B')
('holding B')

stack(B A)
stack(B C)
Example

The diagram illustrates a planning problem using the domain of blocks. The states and actions are represented as nodes and edges in the graph. The initial state is (on B A), (clear A), (clear C), (on table B), (on table C), and (hand-empty). The actions include:

- **unstack(A B)**
- **unstack(A C)**
- **pickup(C)**
- **putdown(A)**
- **putdown(C)**
- **stack(A B)**
- **stack(A C)**
- **stack(C A)**
- **stack(C B)**
- **stack(B C)**
- **(clear B)**
- **(clear C)**
- **(holding A)**
- **(holding C)**

The goal state desired is (ontable A), (on C B), (on C A), (clear C), (clear B), (on A C), and (holding B). The heuristic function is used to evaluate the states and guide the search towards the goal.
Example

State-Space Planning
Heuristics
Hill-Climbing
Pruning

Goal Distance
Graphplan Relation

Example:

- (ontable B)
- (ontable C)
- (on A B)
- (clear A)
- (clear C)
- (hand-empty)

- unstack(A B) → (clear B)
- pickup(C) → (holding A)
- (holding C)

- putdown(A) → stack(A B)
- putdown(C) → stack(A C)
- stack(A C) → stack(C A)
- stack(C A) → stack(C B)
- stack(C B) → pickup(B)

- (ontable A) → (on A C)
- (on A C) → (on C A)
- (on C A) → (on C B)
- (on C B) → (holding B)

- (on B A) → putdown(B) → stack(B A)
- (on B C) → stack(B C)

Chad Hogg
FastForward: Heuristics For Planning
Example

Chad Hogg

FastForward: Heuristics For Planning
State-Space Planning
Heuristics
Hill-Climbing
Pruning

Example

(ontable B)
(ontable C)
(on A B)
(clear A)
(clear C)
(hand-empty)

unstack(A B) → (clear B)
ipickup(C) → (holding A)

putdown(A)
putdown(C)
stack(A B)
stack(A C)
stack(C A)
stack(C B)
ipickup(B)

(on B A)
(on B C)

putdown(B)
stack(B A)
stack(B C)

(ontable A)
(on A C)
(on C A)
(on C B)
(holding B)
Graphplan’s solution has three action levels
Example

- Graphplan’s solution has three action levels
- Each level has one action: \texttt{unstack(A B)}, \texttt{pickup(B)}, \texttt{stack(B C)}
Example

- Graphplan’s solution has three action levels
- Each level has one action: \textit{unstack}(A B), \textit{pickup}(B), \textit{stack}(B C)
- Thus, $h(s_0) = 3$
Graphplan’s solution has three action levels

- Each level has one action: `unstack(A, B)`, `pickup(B)`, `stack(B, C)`

Thus, $h(s_0) = 3$

This “plan” makes little sense, but it contains a sense of how much needs to be done.
When we return, we will discuss how FastForward uses this heuristic.
A hill-climbing algorithm is a greedy search strategy that moves to the neighbor with highest value.
Hill-Climbing Defined

- A *hill-climbing* algorithm is a greedy search strategy that moves to the neighbor with highest value.
- Often finds a good local maxima, but not the optimal solution.
Hill-Climbing Defined

- A *hill-climbing* algorithm is a greedy search strategy that moves to the neighbor with highest value.
- Often finds a good local maxima, but not the optimal solution.
- We want to minimize a heuristic, so perhaps “valley descending” would be more appropriate.
Outline

1. State-Space Planning
   - Review
   - VanillaIce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion
HSP uses standard hill-climbing and a heuristic based on the relaxations previously discussed.
Hill-Climbing in State Space

- HSP uses standard hill-climbing and a heuristic based on the relaxations previously discussed.
- When no successor is a better state, and goals are not met, make arbitrary choice.
Hill-Climbing in State Space

- $HSP$ uses standard hill-climbing and a heuristic based on the relaxations previously discussed.
- When no successor is a better state, and goals are not met, make arbitrary choice.
- No backtracking, so a bad choice can make the problem unsolvable.
Hill-Climbing in State Space

- HSP uses standard hill-climbing and a heuristic based on the relaxations previously discussed.
- When no successor is a better state, and goals are not met, make arbitrary choice.
- No backtracking, so a bad choice can make the problem unsolvable.
- Sound, but not complete or admissible.
Outline

1. State-Space Planning
   - Review
   - VanillaIce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion
Enforced Hill-Climbing

- FF uses a slightly modified hill-climbing algorithm
Enforced Hill-Climbing

- FF uses a slightly modified hill-climbing algorithm
- Instead of choosing the best successor, perform breadth-first search for the first strictly better descendent
Enforced Hill-Climbing

- FF uses a slightly modified hill-climbing algorithm
- Instead of choosing the best successor, perform breadth-first search for the first strictly better descendent
- Less likely to randomly wander around plateaus
**Enforced Hill-Climbing**

- FF uses a slightly modified hill-climbing algorithm
- Instead of choosing the best successor, perform breadth-first search for the first strictly better descendent
- Less likely to randomly wander around plateaus
- Also sound but neither complete nor admissible
Outline

1. State-Space Planning
   - Review
   - VanillaIce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion
Outline

1. State-Space Planning
   - Review
   - Vanillalce
   - FastForward

2. Heuristics
   - Goal Distance
   - Graphplan Relation

3. Hill-Climbing
   - Standard Form
   - Enforced

4. Pruning
   - Helpful Actions
   - Goal Deletion