Chapter 9
Heuristics in Planning

Dana S. Nau

CMSC 722, AI Planning
University of Maryland, Spring 2008
Planning as Nondeterministic Search

Abstract-search($u$)
   if Terminal($u$) then return($u$)
   $u \leftarrow$ Refine($u$) ;; refinement step
   $B \leftarrow$ Branch($u$) ;; branching step
   $B' \leftarrow$ Prune($B$) ;; pruning step
   if $B' = \emptyset$ then return(failure)
   nondeterministically choose $v \in B'$
   return(Abstract-search($v$))
end
Making it Deterministic

Depth-first-search($u$)
  if Terminal($u$) then return($u$)
  $u \leftarrow$ Refine($u$) ;; refinement step
  $B \leftarrow$ Branch($u$) ;; branching step
  $C \leftarrow$ Prune($B$) ;; pruning step
  while $C \neq \emptyset$ do
    $v \leftarrow$ Select($C$) ;; node-selection step
    $C \leftarrow C - \{v\}$
    $\pi \leftarrow$ Depth-first-search($v$)
    if $\pi \neq$ failure then return($\pi$)
  return(failure)
end
Node-Selection Heuristic

- Suppose we’re searching a tree in which each edge \( (s, s') \) has a cost \( c(s, s') \)
  - If \( p \) is a path, let \( c(p) = \) sum of the edge costs
  - For classical planning, this is the length of \( p \)

- For every state \( s \), let
  - \( g(s) = \) cost of the path from \( s_0 \) to \( s \)
  - \( h^*(s) = \) least cost of all paths from \( s \) to goal nodes
  - \( f^*(s) = g(s) + h^*(s) = \) least cost of all paths from \( s_0 \) to goal nodes that go through \( s \)

- Suppose \( h(s) \) is an estimate of \( h^*(s) \)
  - Let \( f(s) = g(s) + h(s) \)
    - \( f(s) \) is an estimate of \( f^*(s) \)
  - \( h \) is admissible if for every state \( s \), \( 0 \leq h(s) \leq h^*(s) \)
  - If \( h \) is admissible then \( f \) is a lower bound on \( f^* \)
The A* Algorithm

- A* on trees:

  loop
  
  choose the leaf node $s$ such that $f(s)$ is smallest
  
  if $s$ is a solution then return it and exit
  
  expand it (generate its children)

- On graphs, A* is more complicated
  
  - additional machinery to deal with multiple paths to the same node

- If a solution exists (and certain other conditions are satisfied), then:
  
  - If $h(s)$ is admissible, then A* is guaranteed to find an optimal solution
  
  - The more “informed” the heuristic is (i.e., the closer it is to $h^*$), the smaller the number of nodes A* expands
  
  - If $h(s)$ is within $c$ of being admissible, then A* is guaranteed to find a solution that’s within $c$ of optimal
Heuristic Functions for Planning

- $\Delta^*(s,p)$: minimum distance from state $s$ to a state that contains $p$
- $\Delta^*(s,s')$: minimum distance from state $s$ to a state that contains all of the literals in $s'$
  - Hence $h^*(s) = \Delta^*(s,g)$ is the minimum distance from $s$ to the goal
- For $i = 0, 1, 2, \ldots$ we will define the following functions:
  - $\Delta_i(s,p)$: an estimate of $\Delta^*(s,p)$
  - $\Delta_i(s,s')$: an estimate of $\Delta^*(s,s')$
  - $h_i(s) = \Delta_i(s,g)$, where $g$ is the goal
Heuristic Functions for Planning

- $\Delta_0(s,s') = \text{what we get if we pretend that}$
  - Negative preconditions and effects don’t exist
  - The cost of achieving a set of preconditions \( \{p_1, \ldots, p_n\} \)
    is the sum of the costs of achieving each $p_i$ separately

\[
\Delta_0(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\infty, & \text{if } p \not\in s \text{ and } \forall a \in A, p \not\in \text{effects}^+(a) \\
\min_a \{1 + \Delta_0(s, \text{precond}^+(a)) \} | p \in \text{effects}^+(a), \text{otherwise} 
\end{cases}
\]

\[
\Delta_0(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s, \\
\sum_{p \in g} \Delta_1(s, p), & \text{otherwise}
\end{cases}
\]

- $\Delta_0(s,s')$ is not admissible, but we don’t necessarily care
- Usually we’ll want to do a depth-first search, not an A* search
  - This already sacrifices admissibility
Computing $\Delta_0$

- Given $s$, can compute $\Delta_0(s,p)$ for every proposition $p$
  - Forward search from $s$
  - $U$ is a set of sets of propositions

$\operatorname{Delta}(s)$

for each $p$ do: if $p \in s$ then $\Delta_0(s,p) \leftarrow 0$, else $\Delta_0(s,p) \leftarrow \infty$

$U \leftarrow \{s\}$

iterate

for each $a$ such that $\exists u \in U$, precond$(a) \subseteq u$ do

$U \leftarrow \{u\} \cup \text{effects}^+(a)$

for each $p \in \text{effects}^+(a)$ do

$\Delta_0(s,p) \leftarrow \min\{\Delta_0(s,p), 1 + \sum_{q \in \text{precond}(a)} \Delta_0(s,q)\}$

until no change occurs in the above updates

end

- From this, can compute $h_0(s) = \Delta_0(s,g) = \sum_{p \in g} \Delta_0(s,p)$
Heuristic Forward Search

Heuristic-forward-search($\pi, s, g, A$)

if $s$ satisfies $g$ then return $\pi$

$\textit{options} \leftarrow \{a \in A \mid a \text{ applicable to } s\}$

for each $a \in \textit{options}$ do $\Delta_0(\gamma(s,a))$

while $\textit{options} \neq \emptyset$ do

$a \leftarrow \text{argmin}\{\Delta_0(\gamma(s,a),g) \mid a \in \textit{options}\}$

$\textit{options} \leftarrow \textit{options} - \{a\}$

$\pi' \leftarrow \text{Heuristic-forward-search}(\pi.a, \gamma(s,a), g, A)$

if $\pi' \neq \text{failure}$ then return($\pi'$)

return($\text{failure}$)

- This is depth-first search, so admissibility is irrelevant
- This is roughly how the HSP planner works
  - First successful use of an A*-style heuristic in classical planning
Heuristic Backward Search

- HSP can also search backward

\[
\text{Backward-search}(\pi, s_0, g, A) \\
\text{if } s_0 \text{ satisfies } g \text{ then return}(\pi) \\
\text{options} \leftarrow \{a \in A \mid a \text{ relevant for } g\} \\
\text{while } \text{options} \neq \emptyset \text{ do} \\
\quad a \leftarrow \arg\min\{\Delta_0(s_0, \gamma^{-1}(g, a)) \mid a \in \text{options}\} \\
\quad \text{options} \leftarrow \text{options} - \{a\} \\
\quad \pi' \leftarrow \text{Backward-search}(a.\pi, s_0, \gamma^{-1}(g, a), A) \\
\quad \text{if } \pi' \neq \text{failure} \text{ then return}(\pi') \\
\text{return failure}
\]
An Admissible Heuristic

\[ \Delta_1(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\infty, & \text{if } p \notin s \text{ and } \forall a \in A, p \notin \text{effects}^+(a) \\
\min_a \{1 + \Delta_1(s, \text{precond}^+(a)) \mid p \in \text{effects}^+(a)\}, & \text{otherwise} 
\end{cases} \]

\[ \Delta_1(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s, \\
\max_{p \in g} \Delta_1(s, p), & \text{otherwise} 
\end{cases} \]

- \( \Delta_1(s, s') \) = what we get if we pretend that
  - Negative preconditions and effects don’t exist
  - The cost of achieving a set of preconditions \( \{p_1, \ldots, p_n\} \)
    is the max of the costs of achieving each \( p_i \) separately

- This heuristic is admissible; thus it could be used with A*
  - It is not very informed
A More Informed Heuristic

- \( \Delta_2 \): instead of computing the minimum distance to each \( p \) in \( g \), compute the minimum distance to each pair \( \{p,q\} \) in \( g \):
  
  - Analogy to GraphPlan’s mutex conditions

\[
\Delta_2(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\infty, & \text{if } p \notin s \text{ and } \forall a \in A, p \notin \text{effects}^+(a) \\
\min_a \left\{ 1 + \Delta_2(s, \text{precond}^+(a)) \mid p \in \text{effects}^+(a) \right\}, & \text{otherwise}
\end{cases}
\]

\[
\Delta_2(s, \{p,q\}) = \min \left\{ \min_a \left\{ 1 + \Delta_2(s, \text{precond}^+(a)) \mid \{p,q\} \subseteq \text{effects}^+(a) \right\}, \min_a \left\{ 1 + \Delta_2(s, \{q\} \cup \text{precond}^+(a)) \mid p \in \text{effects}^+(a) \right\}, \min_a \left\{ 1 + \Delta_2(s, \{p\} \cup \text{precond}^+(a)) \mid q \in \text{effects}^+(a) \right\} \right\}
\]

\[
\Delta_2(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s, \\
\max_{p,q} \Delta_2(s, \{p,q\}) \mid \{p,q\} \subseteq g \}, & \text{otherwise}
\end{cases}
\]
More Generally, …

Recall that $\Delta^*(s, g)$ is the true minimal distance from a state $s$ to a goal $g$. $\Delta^*$ can be computed (albeit at great computational cost) according to the following equations:

$$\Delta^*(s, g) = \begin{cases} 
0 & \text{if } g \subseteq s, \\
\infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \text{ and } \\
\min_a \{1 + \Delta^*(s, \gamma^{-1}(g, a)) \mid a \text{ relevant for } g\} & \text{otherwise.}
\end{cases} \quad (9.4)$$

- From this, can define $\Delta_k(s, g) = \max$ distance to each $k$-tuple $\{p_1, p_2, \ldots, p_k\}$ in $g$
- Analogy to $k$-ary mutex conditions

$$\Delta_k(s, g) = \begin{cases} 
0 & \text{if } g \subseteq s, \\
\infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \\
\min_a \{1 + \Delta^*(s, \gamma^{-1}(g, a)) \mid a \text{ relevant for } g\} & \text{if } |g| \leq k, \\
\max_{g'} \{\Delta_k(s, g') \mid g' \subseteq g \text{ and } |g'| = k\} & \text{otherwise.}
\end{cases} \quad (9.5)$$
\[ \Delta_2(s, p) = \begin{cases} 
0, & \text{if } p \in s \\
\infty, & \text{if } p \not\in s \text{ and } \forall a \in A, p \not\in \text{effects}^+(a) \\
\min_a \{1 + \Delta_2(s, \text{precond}^+(a)) \mid p \in \text{effects}^+(a)\}, & \text{otherwise} 
\end{cases} \]

\[ \Delta_2(s, \{p,q\}) = \min \begin{cases} 
\min_a \{1 + \Delta_2(s, \text{precond}^+(a)) \mid \{p,q\} \subseteq \text{effects}^+(a)\} \\
\min_a \{1 + \Delta_2(s, \{q\} \cup \text{precond}^+(a)) \mid p \in \text{effects}^+(a)\} \\
\min_a \{1 + \Delta_2(s, \{p\} \cup \text{precond}^+(a)) \mid q \in \text{effects}^+(a)\} 
\end{cases} \]

\[ \Delta_2(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s, \\
\max_{p,q} \Delta_2(s, \{p,q\}) \mid \{p,q\} \subseteq g, & \text{otherwise} 
\end{cases} \]

\[ \Delta_k(s, g) = \begin{cases} 
0 & \text{if } g \subseteq s, \\
\infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \\
\min_a \{1 + \Delta^*(s, \gamma^{-1}(g, a)) \mid a \text{ relevant for } g\} & \text{if } |g| \leq k, \\
\max_{g'} \{\Delta_k(s, g') \mid g' \subseteq g \text{ and } |g'| = k\} & \text{otherwise.} 
\end{cases} \]
Complexity of Computing the Heuristic

- Takes time $\Omega(n^k)$
- If $k \geq \max(|g|, \max\{|\text{precond}(a)| : a \text{ is an action}\})$, then computing $\Delta(s,g)$ is as hard as solving the entire planning problem.
Getting Heuristic Values from a Planning Graph

- Recall how GraphPlan works:
  
  loop
  
  Graph expansion: this takes polynomial time
  
  extend a “planning graph” forward from the initial state
  
  until we have achieved a necessary (but insufficient) condition
  
  for plan existence

  Solution extraction: this takes exponential time
  
  search backward from the goal, looking for a correct plan
  
  if we find one, then return it

  repeat
Using Planning Graphs to Compute $h(s)$

- In the graph, there are alternating layers of ground literals and actions.
- The number of “action” layers is a lower bound on the number of actions in the plan.
- Construct a planning graph, starting at $s$.

More specifically:

- $\Delta^g(s,p) =$ level of the first layer that “possibly achieves” $p$.
- $\Delta^g(s,g)$ is very close to $\Delta_2(s,g)$.
  - $\Delta_2(s,g)$ counts each action individually.
  - $\Delta^g(s,g)$ groups together the independent actions in a layer.
The FastForward Planner

- Use a heuristic function similar to $h(s) = \Delta^g(s,g)$
  - Some ways to improve it (I’ll skip the details)
- Don’t want an A*-style search (takes too much memory)
- Instead, use a greedy procedure:

  until we have a solution, do
  expand the current state $s$
  $s :=$ the child of $s$ for which $h(s)$ is smallest
  (i.e., the child we think is closest to a solution)

- There are some ways to improve this (I’ll skip the details)
- Can’t guarantee how fast it will find a solution, or how good a solution it will find
  - However, it works pretty well on many problems
AIPS-2000 Planning Competition

- FastForward did quite well
- In the this competition, all of the planning problems were classical problems
- Two tracks:
  - “Fully automated” and “hand-tailored” planners
  - FastForward participated in the fully automated track
    » It got one of the two “outstanding performance” awards
  - Large variance in how close its plans were to optimal
    » However, it found them very fast compared with the other fully-automated planners
2002 International Planning Competition

- Among the automated planners, FastForward was roughly in the middle
- LPG (graphplan + local search) did much better, and got a “distinguished performance of the first order” award

- It’s interesting to see how FastForward did in problems that went beyond classical planning
  - Numbers, optimization
- Example: Satellite domain, numeric version
  - A domain inspired by the Hubble space telescope (a lot simpler than the real domain, of course)
    - A satellite needs to take observations of stars
    - Gather as much data as possible before running out of fuel
  - Any amount of data gathered is a solution
    - Thus, FastForward always returned the null plan
2004 International Planning Competition

- FastForward’s author was one of the competition chairs
  - Thus FastForward did not participate
For plan-space planning, refinement = selecting the next flaw to work on
One Possible Heuristic

- Fewest Alternatives First (FAF)
Do Others Work Better?

- Sometimes yes, sometimes no
- Limits to how good *any* flaw-selection heuristic can do
The search space is an AND/OR tree

Deciding what flaw to work on next = serializing this tree (turning it into a state-space tree)

- at each AND branch, choose a child to expand next, and delay expanding the other children
One Serialization
Another Serialization
Why Does This Matter?

● Different refinement strategies produce different serializations
  ◆ the search spaces have different numbers of nodes

● In the worst case, the planner will search the entire serialized search space

● The smaller the serialization, the more likely that the planner will be efficient

● One pretty good heuristic: fewest alternatives first
How Much Difference Can the Refinement Strategy Make?

- Case study: build an AND/OR graph from repeated occurrences of this pattern:

  \[
  \begin{array}{c}
  \text{\ldots} \\
  b
  \end{array}
  \]

- Example:
  - number of levels \( k = 3 \)
  - branching factor \( b = 2 \)

- Analysis:
  - Total number of nodes in the AND/OR graph is \( n = \Theta(b^k) \)
  - How many nodes in the best and worst serializations?
Case Study, Continued

- The best serialization contains $\Theta(b^{2^k})$ nodes
- The worst serialization contains $\Theta(2^k b^{2^k})$ nodes
  - The size differs by an exponential factor
  - But both serializations are *doubly* exponentially large
- To do better, need good node selection, branching, pruning