Intelligent Agents
Heuristic Search Planning

Ute Schmid

Cognitive Systems, Applied Computer Science, Bamberg University

last change: June 25, 2014
Heuristics can reduce search effort dramatically because estimates about success/costs of partial solution paths can restrict (bound) search. Typically, heuristic functions are pre-defined by a human expert. In domain-independent planning, search is independent of domain knowledge, that is, knowledge about the distance of a state to the goal is not available to guide search. How can heuristics be generated automatically for domain-independent planning?

Outline

- Recapitulation
  - Planning as search
  - Node selection heuristics and A*
- Heuristic functions for planning
- Problem relaxation
- Hector Geffner’s HSP planning approach
- Informedness and Admissibility
Planning as Non-deterministic Search

Abstract-search($u$)

\[
\text{if Terminal}(u) \text{ then return } (u)
\]
\[
\text{\textbf{if}} \quad B' = \emptyset \text{ \textbf{then return (failure)}}
\]
\[
\text{\textbf{non-deterministically choose}} \quad v \in B'
\]
\[
\text{return (Abstract-search}(v))
\]

end
Planning as Search

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

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

- 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 \(f^*\)

Be aware of the notation difference: here \(h^*\) is the known optimal least costs from a node \(n\) to the goal and \(h\) is the estimate
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

Estimating the heuristics is based on relaxation of the problem.

Ignoring negative preconditions and effects allows for very fast progression from initial state to goals.

Dana Nau: Lecture slides for Automated Planning
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License:http://creativecommons.org/licenses/by-nc-sa/2.0/
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 $\{p_1, \cdots, p_n\}$ is the sum of the costs of achieving each $p_i$ separately
  $$\Delta_0(s, p) = \begin{cases} 0, & \text{if } p \in s \\ \infty, & \text{if } p \notin s \text{ and } \forall a \in A, p \notin \text{effects}^+(a) \\ \min_a \{1 + \Delta_0(s, \text{precond}^+(a)) \mid p \in \text{effects}^+(a)\}, & \text{otherwise} \end{cases}$$

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

- $\Delta_0(s, s')$ 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 (because DFS does not guarantee optimal solutions)
Computing $\Delta_0$

Delta(s)

\[
\text{foreach } p \text{ do }
\begin{align*}
\text{if } p \in s \text{ then } & \quad \Delta_0(s, p) \leftarrow 0 \\
\text{else } & \quad \Delta_0(s, p) \leftarrow \infty
\end{align*}
\]

end

U ← s;

repeat

\[
A \leftarrow \{a | \text{precond}(a) \subseteq U\};
\]

\[
\text{foreach } a \in A \text{ do }
\begin{align*}
U & \leftarrow U \cup \text{effects}^+(a); \\
\text{foreach } p \in \text{effects}^+(a) \text{ do } & \quad \Delta_0(s, p) \leftarrow \min\{\Delta_0(s, p), 1 + \sum_{q \in \text{precond}(a)} \Delta_0(s, q)\}
\end{align*}
\]

end

until no change occurs in the above updates;

Slightly modified from Dana Nau

Dana Nau: Lecture slides for *Automated Planning*

Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/
Heuristic Forward Search

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

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

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

for each $a \in \text{options}$ do$
\text{Delta}(\gamma(s, a))$

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

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

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

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

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

end

return(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
Using heuristics in forwards and backwards search

Heuristic Backward Search

- HSP can also search backward

\[
\text{Backward-search}(\pi, s_0, g, A)
\]

if \( s_0 \) satisfies \( g \) then return \( \pi \)

\[
\text{options} \leftarrow \{ a \in A \mid a \text{ relevant for } g \}
\]

while \( \text{options} \neq \emptyset \) do

\[
a \leftarrow \text{argmin}\{\Delta_0(s_0, \gamma^{-1}(g, a)) \mid a \in \text{options}\}
\]

\[
\text{options} \leftarrow \text{options} - \{a\}
\]

\[
\pi' \leftarrow \text{Backward-search}(a.\pi, s_0, \gamma^{-1}(g, a), A)
\]

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

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

Dana Nau: Lecture slides for Automated Planning
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/
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

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

$$
\Delta_2(s, \{p, q\}) = \min \left\{ \begin{array}{l}
\min_a \{1 + \Delta_2(s, \text{precond}^+(a))|\{p, q\} \subseteq \text{effects}^+(a)\} \\
\min_a \{1 + \Delta_2(s, \{q\} \cup \text{precond}^+(a))|p \in \text{effects}^+(a)\} \\
\min_a \{1 + \Delta_2(s, \{p\} \cup \text{precond}^+(a))|q \in \text{effects}^+(a)\}
\end{array} \right\}
$$

$$
\Delta_2(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s \\
\max_{p \in g} \Delta_2(s, p)|\{p, q\} \subseteq g\}, & \text{otherwise}
\end{cases}
$$
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:

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

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
Efficient search based on a heuristic function can be applied to domain-independent planning

By relaxation, a (possible non-admissible) heuristic can be estimated

Calculation of the heuristics $\delta$ is based on a polynomial time algorithm (dynamic programming, using memoization)

More informed heuristics are more expensive to calculate