Intelligent Agents

Heuristic Search Planning

Ute Schmid

Cognitive Systems, Applied Computer Science, Bamberg University

last change: June 25, 2014

Heuristic Search and Domain-Independent Planning

- 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?
- Hector Geffner proposed a method to estimate a heuristics and thereby made efficient search techniques which exploit heuristics available to planning (1998 planning competition, HSP) see Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, /129/(1), 5-33.

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)
    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)
    non-deterministically choose $v \in B'$
    return (Abstract-search(v))
end

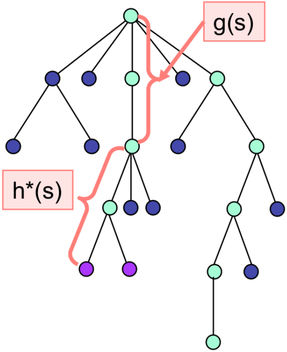
Dana Nau: Lecture slides for *Automated Planning*

Making it Deterministic

```
Depth-first-search(u)
    if Terminal(u) then return (u)
    u ← Refine (u)              ;;  refinement step
    B ← Branch (u)              ;;  branching step
    C ← Prune (B)               ;;  pruning step
    while C ≠ ∅ do
        v ← Select(C)           ;;  node-selection step
        C ← C − {v}
        π ← Depth-first-search(v)
        if π ≠ failure then return (π)
    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 $c(p)$ = 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^*



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

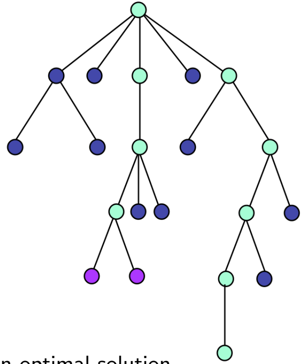
Dana Nau: Lecture slides for *Automated Planning*

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

Dana Nau: Lecture slides for *Automated Planning*

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 $i = 0, 1, 2, \cdots$ 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

Heuristic Functions for Planning

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

$$\Delta_0(s,p) = \begin{cases} 0, & \text{if } p \in s \\ \infty, & \text{if } p \notin s \text{ and } \forall a \in A, p \notin \text{effects}^+(a) \\ min_a\{1 + \Delta_0(s, \text{precond}^+(a)) | p \in \text{effects}^+(a), & \text{otherwise} \end{cases}$$

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

- $\Delta_0(s, s')$ 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)

Dana Nau: Lecture slides for *Automated Planning*

Computing Δ_0

```
Delta(s)
foreach p do
    if p ∈ s then
        Δ0(s, p) ← 0
    else
        Δ0(s, p) ← ∞
    end
end
U ← s;
repeat
    A ← {a | precond(a) ⊂ U};
    foreach a ∈ A do
        U ← U ∪ effects+(a);
        foreach p ∈ effects+(a) do
            Δ0(s, p) ← min{Δ0(s, p), 1 + Σ_{q ∈ precond(a)} Δ0(s, q)};
        end
    end
until no change occurs in the above updates;
```

Slightly modified from Dana Nau

Dana Nau: Lecture slides for *Automated Planning*

Heuristic Forward Search

Heuristic-forward-search(π, s, g, A)
    if s satisfies g then return π
    $options \leftarrow \{a \in A \mid a$ applicable to $s\}$
    for each $a \in options$ do Delta($\gamma(s, a)$)
    while $options \neq \emptyset$ do
        $a \leftarrow \text{argmin}\{\Delta_0(\gamma(s, a), g) \mid a \in options\}$
        $options \leftarrow options - \{a\}$
        $\pi' \leftarrow$ Heuristic-forward-search($\pi.a, \gamma(s, a), g, A$)
        if $\pi' \neq$ failure then return(π')
    return(failure)
end

- 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

Dana Nau: Lecture slides for *Automated Planning*

Heuristic Backward Search

- HSP can also search backward

```
Backward-search(π, s₀, g, A)
    if s₀ satisfies g then return π
    options ← {a ∈ A | a relevant for g}
    while options ≠ ∅ do
        a ← argmin{Δ₀(s₀, γ⁻¹(g, a)) | a ∈ options}
        options ← options − {a}
        π' ← Backward-search(a.π, s₀, γ⁻¹(g, a), A)
        if π' ≠ failure then return(π')
    return(failure)
end
```

Dana Nau: Lecture slides for *Automated Planning*

An Admissible Heuristic

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

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

- $\Delta_1(s, s') =$ 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*

A More Informed Heuristic

- Δ_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

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

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

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

Dana Nau: Lecture slides for *Automated Planning*

More Generally, ...

Recall that $\Delta^*(s,g)$ is the true minimal distance from a state s to a goal g. Δ^* 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

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

Dana Nau: Lecture slides for *Automated Planning*

Summary

- 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 δ is based on a polynomial time algorithm (dynamic programming, using memoization)
- More informed heuristics are more expensive to calculate