Planning, Execution & Learning

1. Heuristic Search Planning

Reid Simmons
Heuristic Search Planning

• Basic Idea
  – *Automatically* Analyze Domain/Problems to Derive Heuristic Estimates to Guide Search

• Decisions
  – How to evaluate search states
  – How to use the evaluations to guide search
  – How to generate successor states

• *Resurgence in Total-Order, State-Space Planners*
  – Best such planner (FF) dominates other types
  – Still a hot topic for research
Search Heuristics

• Admissible
  – What?
  – Why Important?

• Informed
  – What?
  – Why Important?
Evaluating Search States

• Basic Idea
  – *Solve a Relaxed Form of the Problem; Use as Estimate for Original Problem*

• Approaches
  – Assume *complete* subgoal independence
  – Assume no *negative* interactions
  – Assume *limited* negative interactions
**HSP (Bonet & Geffner, 1997)**

- Heuristic State-Space Planner
  - Can Do Either Progression or Regression

- Relax Problem by Eliminating “Delete” Lists
  - Essentially compute transitive closure of actions, starting at initial state
  - Cost of literal is stage/level at which first appears
  - Continue until no new literals are added
  - Similar to *GraphPlan’s* forward search
Computing Costs of Literals

\[ \begin{align*}
0 \text{On}(C, A) & \quad 0 \text{On}(A, \text{Table}) & \quad 0 \text{On}(B, \text{Table}) & \quad 0 \text{Handempty} & \quad 0 \text{Clear}(C) & \quad 0 \text{Clear}(B) \\
\text{Pick}(C, A) & \quad \text{PickT}(B) & \\
0 \text{On}(C, A) & \quad 0 \text{On}(A, \text{Table}) & \quad 0 \text{On}(B, \text{Table}) & \quad 0 \text{Handempty} & \quad 0 \text{Clear}(C) & \quad 0 \text{Clear}(B) \\
1 \text{Holding}(C) & \quad 1 \text{Holding}(B) & \quad 1 \text{Clear}(A) \\
\text{PutT}(C) & \quad \text{Put}(C, A) & \quad \text{Put}(C, B) & \quad \text{PutT}(B) & \quad \text{Put}(B, A) & \quad \text{Put}(B, C) & \quad \text{PickT}(A) \\
2 \text{On}(C, \text{Table}) & \quad 2 \text{On}(C, B) & \quad 2 \text{On}(B, A) & \quad 2 \text{On}(B, C) & \quad 2 \text{Holding}(A) \\
\text{PickT}(C) & \quad \text{Pick}(C, B) & \quad \text{Pick}(B, A) & \quad \text{Pick}(B, C) & \quad \text{Put}(A, B) & \quad \text{Put}(A, C) \\
3 \text{On}(A, B) & \quad 3 \text{On}(A, C) \\
\text{On}(A, B) & \quad \& \quad \text{On}(B, C) & \quad \text{Estimate}: 5 \quad \text{Actually}: 6 \\
\text{On}(A, C) & \quad \& \quad \text{On}(C, B) & \quad \text{Estimate}: 5 \quad \text{Actually}: 4
\end{align*} \]
HSP Heuristics

• **Max**
  - Cost of action is *maximum* over costs of preconditions
  - Admissible, but not very informed

• **Sum**
  - Cost of action is *sum* of precondition costs
  - Informed, but not admissible

• **H²**
  - Solve for *pairs* of literals
  - Take maximum cost over all pairs
  - Informed, and claimed to be admissible
Heuristic Search Strategies

- Best-First

- A*

- Weighted A*
  - $H(s) = \text{cost-so-far}(s) + W \times \text{estimated-cost}(s)$
  - Not admissible, but tends to perform much better than A*

- Hill-Climbing
  - Rationale: Heuristics tend to be better discriminators amongst local alternatives than as global (absolute) estimate
  - Random “restarts” when stuck
  - *Perfect opportunity for transformational operators*
“Enforced” Hill Climbing

• Used to Avoid “Wandering” on “Plateaus” or in Local Minima
  – Perform breadth-first search until find some descendant state whose heuristic value is less than the current state

• Shown to be Very Effective
  – Especially when search space is pruned to eliminate actions that are “unlikely” to lead to goal achievement

• Used by FF
INFORMED SEARCH ALGORITHMS

CHAPTER 4, SECTIONS 1–2
## Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes*</td>
<td>Yes*</td>
<td>No</td>
<td>Yes, if $l \geq d$</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$b^{d+1}$</td>
<td>$b^{C*/\epsilon}$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^{d+1}$</td>
<td>$b^{C*/\epsilon}$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes*</td>
</tr>
</tbody>
</table>

Chapter 3  
71
Outline

♦ Best-first search
♦ A* search
♦ Heuristics
Review: Tree search

function Tree-Search(problem, fringe) returns a solution, or failure

  fringe ← Insert(Make-Node(Initial-State[problem]), fringe)

loop do
  if fringe is empty then return failure
  node ← Remove-Front(fringe)
  if Goal-Test[problem] applied to State(node) succeeds return node
  fringe ← InsertAll(Expand(node, problem), fringe)

A strategy is defined by picking the order of node expansion
Best-first search

Idea: use an evaluation function for each node
  – estimate of “desirability”

⇒ Expand most desirable unexpanded node

Implementation:
fringe is a queue sorted in decreasing order of desirability

Special cases:
greedy search
A* search
Romania with step costs in km

Straight–line distance to Bucharest
Arad 366
Bucharest 0
Craiova 160
Dobrogea 242
Eforie 161
Fagaraș 178
Giurgiu 77
Hirsova 151
Iasi 226
Lugoj 244
Mehadia 241
Neamț 234
Oradea 380
Pitesti 98
Rimnicu Vâlcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374

Chapter 4, Sections 1–2
Greedy search

Evaluation function $h(n)$ (heuristic)

$= \text{estimate of cost from } n \text{ to the closest goal}$

E.g., $h_{SLD}(n) = \text{straight-line distance from } n \text{ to Bucharest}$

Greedy search expands the node that appears to be closest to goal
Greedy search example

Arad
366
Greedy search example

- Arad
  - Sibiu 253
  - Timisoara 329
  - Zerind 374
Greedy search example

Chapter 4, Sections 1–2
Greedy search example
Properties of greedy search

Complete??
Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with Oradea as goal,
   Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time??
Properties of greedy search

**Complete**? No—can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

**Time**? $O(b^m)$, but a good heuristic can give dramatic improvement

**Space**?
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

**Time??** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space??** $O(b^m)$—keeps all nodes in memory

**Optimal??**
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

**Time??** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space??** $O(b^m)$—keeps all nodes in memory

**Optimal??** No
A* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

$g(n) =$ cost so far to reach $n$

$h(n) =$ estimated cost to goal from $n$

$f(n) =$ estimated total cost of path through $n$ to goal

A* search uses an admissible heuristic

i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from $n$.

(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal $G$.)

E.g., $h_{SLD}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal
A* search example

Arad

366 = 0 + 366
A* search example

Sibiu
393 = 140 + 253

Timisoara
447 = 118 + 329

Zerind
449 = 75 + 374
A* search example

Chapter 4, Sections 1–2
A* search example

Arad

Sibiu

Fagaras

Oradea

Rimnicu Vilcea

Craiova

Pitesti

Sibiu

Timisoara

Zerind

447=118+329

449=75+374

646=280+366

415=239+176

671=291+380

526=366+160

417=317+100

553=300+253
A* search example

Chapter 4, Sections 1–2
A* search example

Chapter 4, Sections 1-2
Optimality of A* (standard proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[
f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0 \\
> g(G_1) \quad \text{since } G_2 \text{ is suboptimal} \\
\geq f(n) \quad \text{since } h \text{ is admissible}
\]

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.
Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing $f$ value

Gradually adds “$f$-contours” of nodes (cf. breadth-first adds layers)
Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$
Properties of A*

Complete??
Properties of A*:

Complete: Yes, unless there are infinitely many nodes with \( f \leq f(G) \)

Time: ??
Properties of A*:

**Complete?** Yes, unless there are infinitely many nodes with $f \leq f(G')$

**Time??** Exponential in $[\text{relative error in } h \times \text{length of soln.}]$

**Space??**
Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal??
Properties of A*

**Complete??** Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time??** Exponential in $[\text{relative error in } h \times \text{length of soln.}]$

**Space??** Keeps all nodes in memory

**Optimal??** Yes—cannot expand $f_{i+1}$ until $f_i$ is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$
Proof of lemma: Consistency

A heuristic is **consistent** if

\[ h(n) \leq c(n, a, n') + h(n') \]

If \( h \) is consistent, we have

\[
\begin{align*}
  f(n') & = g(n') + h(n') \\
         & = g(n) + c(n, a, n') + h(n') \\
         & \geq g(n) + h(n) \\
         & = f(n)
\end{align*}
\]

I.e., \( f(n) \) is nondecreasing along any path.
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

\[ h_1(S) = ?? \]
\[ h_2(S) = ?? \]
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

\[
\begin{array}{ccc}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{array}
\]

Start State

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & \\
\end{array}
\]

Goal State

\[ h_1(S) = 6 \]
\[ h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14 \]
Dominance

If \( h_2(n) \geq h_1(n) \) for all \( n \) (both admissible) then \( h_2 \) dominates \( h_1 \) and is better for search

Typical search costs:

\[ d = 14 \quad IDS = 3,473,941 \text{ nodes} \]
\[ A^*(h_1) = 539 \text{ nodes} \]
\[ A^*(h_2) = 113 \text{ nodes} \]

\[ d = 24 \quad IDS \approx 54,000,000,000 \text{ nodes} \]
\[ A^*(h_1) = 39,135 \text{ nodes} \]
\[ A^*(h_2) = 1,641 \text{ nodes} \]

Given any admissible heuristics \( h_a, h_b \),

\[ h(n) = \max(h_a(n), h_b(n)) \]

is also admissible and dominates \( h_a, h_b \)
Relaxed problems

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.

If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.
Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once

Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour
Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest $h$
  – incomplete and not always optimal

A* search expands lowest $g + h$
  – complete and optimal
  – also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems