Heuristic Search

Part B

R&N: Chap. 4, Sect. 4.1–3

Slides from Jean-Claude Latombe at Stanford University
(used with permission)

What to do with revisited states?

The heuristic $h$ is clearly admissible

If we discard this new node, then the search algorithm expands the goal node next and returns a non-optimal solution.
What to do with revisited states?

Instead, if we do not discard nodes revisiting states, the search terminates with an optimal solution.

But ...

If we do not discard nodes revisiting states, the size of the search tree can be exponential in the number of visited states.

- It is not harmful to discard a node revisiting a state if the cost of the new path to this state is ≥ cost of the previous path. (so, in particular, one can discard a node if it re-visits a state already visited by one of its ancestors)
- A* remains optimal, but states can still be revisited multiple times. (the size of the search tree can still be exponential in the number of visited states)
- Fortunately, for a large family of admissible heuristics – consistent heuristics – there is a much more efficient way to handle revisited states.
Consistent Heuristic

A heuristic $h$ is consistent (or monotone) if:

1) for each node $N$ and each child $N'$ of $N$:
   
   
   $h(N) \leq c(N,N') + h(N')$

2) for each goal node $G$:
   
   $h(G) = 0$

A consistent heuristic is also admissible.

Intuition: a consistent heuristic becomes more precise as we get deeper in the search tree.

Consistency Violation

If $h$ tells that $N$ is 100 units from the goal, then moving from $N$ along an arc costing 10 units should not lead to a node $N'$ that $h$ estimates to be 10 units away from the goal.

Admissibility and Consistency

- A consistent heuristic is also admissible.
- An admissible heuristic may not be consistent, but many admissible heuristics are consistent.
### 8-Puzzle

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

**STATE(N)** goal

- \( h_1(N) \) = number of misplaced tiles
- \( h_2(N) \) = sum of the (Manhattan) distances of every tile to its goal position

are both consistent (why?)

### Robot Navigation

Cost of one horizontal/vertical step = 1
Cost of one diagonal step = \( \sqrt{2} \)

\[
h_1(N) = \sqrt{(x_N-x_g)^2+(y_N-y_g)^2}
\]

is consistent

\[
h_2(N) = |x_N-x_g| + |y_N-y_g|
\]

is consistent if moving along diagonals is not allowed, and not consistent otherwise.

### Result #2

If \( h \) is consistent, then whenever A* expands a node, it has already found an optimal path to this node’s state.
Proof (1/2)

1) Consider a node $N$ and its child $N'$. Since $h$ is consistent: $h(N) \leq c(N,N') + h(N')$

\[
f(N) = g(N) + h(N) \leq g(N) + c(N,N') + h(N') = f(N')
\]

So, $f$ is non-decreasing along any path.

Proof (2/2)

2) If a node $K$ is selected for expansion, then any other node $N$ in the fringe verifies $f(N) \geq f(K)$

If one node $N$ lies on another path to the state of $K$, the cost of this other path is no smaller than that of the the path to $K$:

\[
f(N') = g(N') + h(N') \geq f(N) \geq f(K) = g(K) + h(K)
\]

Then because $h(N') = h(K)$, we must have $g(N') \geq g(K)$.

Implication of Result #2

The path to $N$ is the optimal path to $S$.

$N_2$ can be discarded.
Revisited States with Consistent Heuristic

- When a node is expanded, store its state into CLOSED
- When a new node N is generated:
  - If STATE(N) is in CLOSED, discard N
  - If there exists a node N' in the fringe such that STATE(N) = STATE(N'), discard the node N or N' with the largest f

Is A* with some consistent heuristic all that we need?

No!

There are very dumb consistent heuristic functions.

For example: $h(x) = 0$

- It is consistent (hence, admissible)!
- A* with $h=0$ is uniform-cost search
- Breadth-first and uniform-cost are particular cases of A*
Heuristic Accuracy

Let $h_1$ and $h_2$ be two consistent heuristics such that for all nodes $N$:

$$h_1(N) \leq h_2(N) \leq h^*(N)$$

$h_2$ is said to be more accurate (or more informed) than $h_1$.

- $h_1(N)$ = number of misplaced tiles = 6
- $h_2(N)$ = sum of distances of every tile to its goal position = 13
- $h_2$ is more accurate than $h_1$

Result #3

- Let $h_2$ be more accurate than $h_1$
- Let $A_1^*$ be $A^*$ using $h_1$
- and $A_2^*$ be $A^*$ using $h_2$
- Whenever a solution exists, all the nodes expanded by $A_2^*$, except possibly for some nodes such that $f_1(N) = f_2(N) = C^*$ (cost of optimal solution), are also expanded by $A_1^*$.

Proof

- $C^*$ = cost of the optimal solution
- Every node $N$ such that $f(N) < C^*$ is eventually expanded.
- No node $N$ such that $f(N) > C^*$ is ever expanded.
- Every node $N$ such that $h_2(N) < C^* - g(N)$ is eventually expanded. So, every node $N$ such that $h_2(N) < C^* - g(N)$ is expanded by $A_2^*$.
- Since $h_2(N) \leq h_1(N) < C^* - g(N)$, $N$ is also expanded by $A_1^*$.
- If there are several nodes $N$ such that $f_1(N) = f_2(N) = C^*$ (such nodes include the optimal goal nodes, if there exists a solution), $A_1^*$ and $A_2^*$ may or may not expand them in the same order (until one goal node is expanded).
Effective Branching Factor

- It is used as a measure the effectiveness of a heuristic.
- Let \( n \) be the total number of nodes expanded by A* for a particular problem and \( d \) the depth of the solution.
- The effective branching factor \( b^* \) is defined by \( n = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d \).

Experimental Results

- 8-puzzle with:
  - \( h_1 \) = number of misplaced tiles
  - \( h_2 \) = sum of distances of tiles to their goal positions
- Random generation of many problem instances
- Average effective branching factors (number of expanded nodes):

<table>
<thead>
<tr>
<th>( d )</th>
<th>IDS</th>
<th>A*</th>
<th>A(^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>6</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>12</td>
<td>2.78 (3,644,035)</td>
<td>1.40 (227)</td>
<td>1.24 (73)</td>
</tr>
<tr>
<td>16</td>
<td>--</td>
<td>1.45</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>--</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>24</td>
<td>--</td>
<td>1.48 (39,135)</td>
<td>1.26 (1,641)</td>
</tr>
</tbody>
</table>

How to create good heuristics?

- By solving relaxed problems at each node.
- In the 8-puzzle, the sum of the distances of each tile to its goal position (\( h_2 \)) corresponds to solving 8 simple problems:
  - \( d_i \) = length of the shortest path to move tile \( i \) to its goal position, ignoring the other tiles, e.g., \( d_5 = 2 \).
- \( h_2 = \sum_{i=1}^{8} d_i \).
- It ignores negative interactions among tiles.
For example, we could consider two more complex relaxed problems:

Can we do better?

\[ h = d_{1234} + d_{5678} \]  
[disjoint pattern heuristic]

These distances are precomputed and stored.  
[Each requires generating a graph of 3,024 nodes/states]
Can we do better?

For example, we could consider two more complex relaxed problems:

Can we do better?

Several order-of-magnitude speedups for the 15- and 24-puzzle

$d_{1234} = \text{length of the shortest path to move tiles 1, 2, 3, and 4 to their goal positions}$

$h = d_{1234} + d_{5678}$ [disjoint pattern heuristic]

These distances are pre-computed and stored.

On Completeness and Optimality

- $A^*$ with a consistent heuristic function has nice properties: completeness, optimality, no need to revisit states
- Theoretical completeness does not mean "practical" completeness if you must wait too long to get a solution (remember the time limit issue)
- So, if one can't design an accurate consistent heuristic, it may be better to settle for a non-admissible heuristic that "works well in practice", even though completeness and optimality are no longer guaranteed

Iterative Deepening $A^*$ (IDA$^*$)

- Idea: Reduce memory requirement of $A^*$ by applying cutoff on values of $f$
- Consistent heuristic function $h$
- Algorithm IDA$^*$:
  1. Initialize cutoff to $f(\text{initial-node})$
  2. Repeat:
     a. Perform depth-first search by expanding all nodes $N$ such that $f(N) \leq \text{cutoff}$
     b. Reset cutoff to smallest value $f$ of non-expanded (leaf) nodes
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles

Cutoff=4
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles

Cutoff: 4

Cutoff: 5
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles

Cutoff = 5

Diagram of the 8-Puzzle with a path and the heuristic function applied.
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles

Cutoff=5
Advantages/Drawbacks of IDA*

• Advantages:
  • Still complete and optimal
  • Requires less memory than A*
  • Avoid the overhead to sort the fringe

• Drawbacks:
  • Can’t avoid revisiting states not on the current path
  • Available memory is poorly used (⇒ memory-bounded search, see R&N p. 101-104)

Local Search

• Light-memory search method
• No search tree; only the current state is represented!
• Only applicable to problems where the path is irrelevant (e.g., 8-queen); unless the path is encoded in the state
• Many similarities with optimization techniques

Steepest Descent

1) S ← initial state
2) Repeat:
   a) S ← arg min \text{successors}(h(S))
   b) if GOAL?(S) return S
   c) if h(S) < h(S) then S ← S else return failure

Similar to:
  • Hill climbing with \(-h\)
  • Gradient descent over continuous space
Application: 8-Queen

1) Pick an initial state \( S \) at random with one queen in each column
2) Repeat \( k \) times:
   a) If \( \text{GOAL}(S) \) then return \( S \)
   b) Pick an attacked queen \( Q \) at random
   c) Move \( Q \) in its column to minimize the number of attacking queens 
      \( \rightarrow \) new \( S \) (non-conflicts heuristic)
3) Return failure

Why does it work ???

1) There are many goal states that are well-distributed over the state space
2) If no solution has been found after a few steps, it's better to start it all over again.
   Building a search tree would be much less efficient because of the high branching factor
3) Running time almost independent of the number of queens

Steepest Descent

1) \( S \leftarrow \) initial state
2) Repeat:
   a) \( S' \leftarrow \arg \min_{S'} \{ h(S') \} \)
   b) if \( \text{GOAL}(S') \) return \( S' \)
   c) if \( h(S') < h(S) \) then \( S \leftarrow S' \) else return failure

may easily get stuck in local minima
\( \rightarrow \) Random restart (as in n-queen example)
\( \rightarrow \) Monte Carlo descent
Monte Carlo Descent

1) \( S \leftarrow \text{initial state} \)
2) \( k \times \text{Repeat}: \)
   a) If \( \text{GOAL}(S) \) then return \( S \)
   b) \( S' \leftarrow \text{successor of } S \text{ picked at random} \)
   c) if \( h(S') \leq h(S) \) then \( S \leftarrow S' \)
   d) else \[ \Delta h = h(S') - h(S) \]
      with probability \(-\exp(-\Delta h/T)\), where \( T \) is called the "temperature" \( S \leftarrow S' \) [Metropolis criterion]
3) Return failure

Simulated annealing lowers \( T \) over the \( k \) iterations.
It starts with a large \( T \) and slowly decreases \( T \).

"Parallel" Local Search Techniques

They perform several local searches concurrently, but not independently:
• Beam search
• Genetic algorithms

See R&N, pages 115-119

Search problems

Blind search

Heuristic search
best-first and \( A^* \)

Construction of heuristics
Variants of \( A^* \)
Local search
When to Use Search Techniques?

1) The search space is small, and
   • No other technique is available, or
   • Developing a more efficient technique is not worth the effort

2) The search space is large, and
   • No other available technique is available, and
   • There exist “good” heuristics