Informed Search

Lecturer: Ji Liu
Thanks for Jerry Zhu's slides

[Based on slides from Andrew Moore http://www.cs.cmu.edu/~awm/tutorials ]
Main messages

- A*. Always be optimistic.
Uninformed vs. informed search

- **Uninformed search** (BFS, uniform-cost, DFS, ID etc.)
  - Knows the actual path cost $g(s)$ from start to a node $s$ in the fringe, but that’s it.

- **Informed search**
  - Also has a heuristic $h(s)$ of the cost from $s$ to goal. (‘h’= heuristic, non-negative)
  - Can be much faster than uninformed search.
Recall: Uniform-cost search

- Uniform-cost search: uninformed search when edge costs are not the same.
- Complete (will find a goal). Optimal (will find the least-cost goal).
- Always expand the node with the least $g(s)$
  - Use a priority queue:
    - Push in states with their first-half-cost $g(s)$
    - Pop out the state with the least $g(s)$ first.
- Now we have an estimate of the second-half-cost $h(s)$, how to use it?
First attempt: Best-first greedy search

- Idea 1: use \( h(s) \) instead of \( g(s) \)
- Always expand the node with the least \( h(s) \)
  - Use a priority queue:
    - Push in states with their second-half-cost \( h(s) \)
    - Pop out the state with the least \( h(s) \) first.
- Known as “best first greedy” search
- How’s this idea?
Best-first greedy search looking stupid

- It will follow the path $A \rightarrow C \rightarrow G$ (why?)
- Obviously not optimal
Second attempt: A search

- Idea 2: use $g(s) + h(s)$
- Always expand the node with the least $g(s) + h(s)$
  - Use a priority queue:
    - Push in states with their first-half-cost $g(s) + h(s)$
    - Pop out the state with the least $g(s) + h(s)$ first.
- Known as “A” search
- How’s this idea?
- Works for this example
A search still not quite right

- A search is not optimal.
Third attempt: A* search

• Same as A search, but the heuristic nonnegative function $h()$ has to satisfy $h(s) \leq h^*(s)$, where $h^*(s)$ is the true cost from node $s$ to the goal.

• Such heuristic function $h()$ is called admissible.
  • An admissible heuristic never over-estimates

It is always optimistic

• A search with admissible $h()$ is called A* search.
Admissible heuristic functions $h$

- **8-puzzle example**

<table>
<thead>
<tr>
<th>Example State</th>
<th>Goal State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
<tr>
<td>4 6 5</td>
<td>4 5 6</td>
</tr>
<tr>
<td>7 4 8</td>
<td>7 8</td>
</tr>
</tbody>
</table>

- Which of the following are admissible heuristics?
  - $h(n) =$ number of tiles in wrong position
  - $h(n) =$ 0
  - $h(n) =$ 1
  - $h(n) =$ sum of Manhattan distance between each tile and its goal location
Admissible heuristic functions $h$

- **8-puzzle example**

<table>
<thead>
<tr>
<th>Example State</th>
<th>Goal State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

- Which of the following are admissible heuristics?
  - $h(n) =$ number of tiles in wrong position **YES**
  - $h(n) =$ 0 **YES**, uninformed uniform cost search
  - $h(n) =$ 1 **NO**, goal state
  - $h(n) =$ sum of Manhattan distance between each tile and its goal location **YES**
Admissible heuristic functions $h$

- In general, which of the following are admissible heuristics? $h^*(n)$ is the true optimal cost from $n$ to goal.
  
  - $h(n) = h^*(n)$
  
  - $h(n) = \max(2, h^*(n))$
  
  - $h(n) = \min(2, h^*(n))$
  
  - $h(n) = h^*(n) - 2$
  
  - $h(n) = \sqrt{h^*(n)}$
Admissible heuristic functions $h$

- In general, which of the following are admissible heuristics? $h^*(n)$ is the true optimal cost from $n$ to goal.

  - $h(n) = h^*(n)$  YES
  - $h(n) = \max(2, h^*(n))$  NO
  - $h(n) = \min(2, h^*(n))$  YES
  - $h(n) = h^*(n) - 2$  NO, possibly negative
  - $h(n) = \sqrt{h^*(n)}$  NO if $h^*(n) < 1$
Heuristics for Admissible heuristics

• How to construct heuristic functions?

Example State

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

Goal State

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

• Often by relaxing the constraints
  • \(h(n)\)=number of tiles in wrong position
    Allow tiles to fly to their destination in one step
  • \(h(n)\)=sum of Manhattan distance between each tile and its goal location
    Allow tiles to move on top of other tiles
“my heuristic is better than yours”

- A heuristic function $h_2$ dominates $h_1$ if for all states $s$
  $$h_1(s) \leq h_2(s) \leq h^*(s)$$

- We prefer heuristic functions as close to $h^*$ as possible, but not over $h^*$.

But

- Good heuristic function might need complex computation.

- Time may be better spent, if we use a faster, simpler heuristic function and expand more nodes.
Q1: When should A* stop?

- Idea: as soon as it generates the goal state?
- $h()$ is admissible
- The goal $G$ will be generated as path $A \rightarrow B \rightarrow G$, with cost 1000.
Q1: The correct A* stop rule

- A* should terminate only when a goal is popped from the priority queue.

- If you have exceedingly good memory, you’ll remember this is the same rule for uniform cost search on cyclic graphs.

- Indeed A* with $h() \equiv 0$ is exactly uniform cost search!
Q2: A* revisiting expanded states

- **One more complication:** A* can revisit an expanded state, and discover a shorter path.

- Can you find the state in question?
Q2: A* revisiting expanded states

• One more complication: A* can revisit an expanded state, and discover a shorter path

We shall put D back into the priority queue, with the smaller \( g+h \)

• Can you find the state in question?
Q3: What if A* revisits a state in the PQ?

(Note the numbers are different)

- We’ve seen this before, with uniform cost search
- ‘promote’ D in the queue with the smaller cost
The A* algorithm

1. Put the start node S on the priority queue, called OPEN
2. If OPEN is empty, exit with failure
3. Remove from OPEN and place on CLOSED a node n for which f(n) is minimum
4. If n is a goal node, exit (trace back pointers from n to S)
5. Expand n, generating all its successors and attach to them pointers back to n. For each successor n' of n
   1. If n' is not already on OPEN or CLOSED estimate h(n'),g(n')=g(n)+c(n,n'), f(n')=g(n')+h(n'), and place it on OPEN.
   2. If n' is already on OPEN or CLOSED, then check if g(n') is lower for the new version of n'. If so, then:
      1. Redirect pointers backward from n' along path yielding lower g(n').
      2. Put n' on OPEN.
      3. If g(n') is not lower for the new version, do nothing.
A*: the dark side

- A* can use lots of memory. \( O(\text{number of states}) \)
- For large problems A* will run out of memory
- We’ll look at two alternatives:
  - IDA*
  - Beam search
IDA*: iterative deepening A*

- Memory bounded search. Assume integer costs
  - Do path checking DFS, do not expand any node with $f(n)>0$. Stop if we find a goal.
  - Do path checking DFS, do not expand any node with $f(n)>1$. Stop if we find a goal.
  - Do path checking DFS, do not expand any node with $f(n)>2$. Stop if we find a goal.
  - Do path checking DFS, do not expand any node with $f(n)>3$. Stop if we find a goal.

... repeat this, increase threshold by 1 each time until we find a goal.

- This is complete, optimal, but more costly than A* in general.
Beam search

- Very general technique, not just for A*
- The priority queue has a fixed size $k$. Only the top $k$ nodes are kept. Others are discarded.
- Neither complete nor optimal, nor can maintain an ‘expanded’ node list, but memory efficient.
- Variation: The priority queue only keeps nodes that are at most $\varepsilon$ worse than the best node in the queue. $\varepsilon$ is the beam width.
- Beam search used successfully in speech recognition.
Example

(All edges are directed, pointing downwards)
### Example

**OPEN**
- S(0+8)
- A(1+8) B(5+4) C(8+3)
- B(5+4) C(8+3) D(4+inf) E(8+inf) G(10+0)
- C(8+3) D(4+inf) E(8+inf) G(10+0) G(9+0)
- C(8+3) D(4+inf) E(8+inf) G(10+0)

**CLOSED**
- S(0+8)
- S(0+8) A(1+8)
- S(0+8) A(1+8) B(5+4)
- S(0+8) A(1+8) B(5+4) G(9+0)

Backtrack: G => B => S.
What you should know

- Know why best-first greedy search is bad.
- Thoroughly understand A*.
- Trace simple examples of A* execution.
- Understand admissible heuristics.
Appendix: Proof that A* is optimal

• Suppose A* finds a suboptimal path ending in goal $G'$, where $f(G') > f^* = \text{cost of optimal path}$

• Let’s look at the first unexpanded node $n$ on the optimal path ($n$ exists, otherwise the optimal goal would have been found)

• $f(n) > f(G')$, otherwise we would have expanded $n$

• $f(n) = g(n) + h(n)$ by definition
  
  $= g^*(n) + h(n)$ because $n$ is on the optimal path
  
  $\leq g^*(n) + h^*(n)$ because $h$ is admissible
  
  $= f^*$ because $n$ is on the optimal path

• $f^* \geq f(n) > f(G')$, contradicting the assumption at top