CS 188: Artificial Intelligence  
Fall 2007

Lecture 3: A* Search  

Dan Klein – UC Berkeley  
Many slides over the course adapted from either Stuart Russell or Andrew Moore

---

Announcements

- **Sections:**
  - New section 106: Tu 5-6pm  
  - You can go to any section, if there’s space  
  - Sections start this week

- **Homework**
  - Project 1 on the web, due 9/12  
  - New written homework format:
    - One or two questions handed out end of section (and online)  
    - Due the next week in section, graded check / no check  
    - Each assignment 1% of grade, cap of 10%, so can skip at least one week, depends on how many there are  
    - Solve in groups of any size, write up alone
Today

- A* Search
- Heuristic Design
- Local Search

Recap: Search

- Search problems:
  - States (configurations of the world)
  - Successor functions, costs, start and goal tests

- Search trees:
  - Nodes: represent paths / plans
  - Paths have costs (sum of action costs)

\[ g(n) = \sum_{x \rightarrow y \in n} \text{cost}(x \rightarrow y) \]

- Strategies differ (only) in fringe management
General Tree Search

```plaintext
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Expanding includes incrementing the path cost!

Uniform Cost

- **Strategy**: expand lowest path cost
- **The good**: UCS is complete and optimal!
- **The bad**:
  - Explores options in every "direction"
  - No information about goal location
Best First

- Strategy: expand nodes which appear closest to goal
  - Heuristic: function which maps states to distance

- A common case:
  - Best-first takes you straight to the (wrong) goal

- Worst-case: like a badly-guided DFS

Example: Heuristic Function

$h(x)$

Graph showing cities and distances.
Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or *backward cost* $g(n)$
- **Best-first** orders by goal proximity, or *forward cost* $h(n)$

A* Search orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager

---

When should A* terminate?

- Should we stop when we enqueue a goal?

- No: only stop when we dequeue a goal
Is A* Optimal?

- What went wrong?
- Actual bad goal cost > estimated good goal cost
- We need estimates to be less than actual costs!

Admissible Heuristics

- A heuristic is *admissible* (optimistic) if:

  \[ h(n) \leq h^*(n) \]

  where \( h^*(n) \) is the true cost to a nearest goal

- E.g. Euclidean distance on a map problem

- Coming up with admissible heuristics is most of what’s involved in using A* in practice.
**Optimality of A*: Blocking**

- **Proof:**
  - What could go wrong?
  - We’d have to have to pop a suboptimal goal $G$ off the fringe before $G^*$
  - This can’t happen:
    - Imagine a suboptimal goal $G$ is on the queue
    - Some node $n$ which is a subpath of $G^*$ must be on the fringe (why?)
    - $n$ will be popped before $G$

\[
\begin{align*}
    f(n) &\leq g(G^*) \\
g(G^*) &< g(G) \\
g(G) & = f(G) \\
f(n) &< f(G)
\end{align*}
\]

---

**UCS vs A* Contours**

- Uniform-cost expanded in all directions

- A* expands mainly toward the goal, but does hedge its bets to ensure optimality
Properties of A*

Uniform-Cost          A*

Admissible Heuristics

- Most of the work is in coming up with admissible heuristics
- Inadmissible heuristics are often quite effective (especially when you have no choice)
- Very common hack: use $\alpha \times h(n)$ for admissible $h$, $\alpha > 1$ to generate a faster but less optimal inadmissible $h'$ from admissible $h$
Example: 8 Puzzle

- What are the states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

Example: 8 Puzzle

- Number of tiles misplaced?
- Why is it admissible?
- h(start) = 8

- This is a relaxed-problem heuristic

<table>
<thead>
<tr>
<th>ID</th>
<th>TILES</th>
<th>Average nodes expanded when optimal path has length...</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>6,300</td>
<td>(3.6 \times 10^6)</td>
</tr>
<tr>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>
8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why admissible?
- \( h(\text{start}) = \sum d_i \) where \( d_i \) is the Manhattan distance of the tile from its goal position.

![Start and Goal States](image)

<table>
<thead>
<tr>
<th>Average nodes expanded when optimal path has length...</th>
<th>...4 steps</th>
<th>...8 steps</th>
<th>...12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
<tr>
<td>MANHATTAN</td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes?
  - What’s wrong with it?

- With A*: a trade-off between quality of estimate and work per node!
Trivial Heuristics, Dominance

- Dominance: $h_a \geq h_c$ if
  $$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible
    $$h(n) = \max(h_a(n), h_b(n))$$

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic

Course Scheduling

- From the university’s perspective:
  - Set of courses $\{c_1, c_2, \ldots, c_n\}$
  - Set of room / times $\{r_1, r_2, \ldots, r_n\}$
  - Each pairing $(c_k, r_m)$ has a cost $w_{km}$
  - What’s the best assignment of courses to rooms?

- States: list of pairings
- Actions: add a legal pairing
- Costs: cost of the new pairing

- Admissible heuristics?

- (Who can think of a cs170 answer to this problem?)
Other A* Applications

- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...

Tree Search: Extra Work?

- Failure to detect repeated states can cause exponentially more work. Why?
Graph Search

- In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)

Very simple fix: never expand a state twice

```plaintext
function Graph-Search(problem, fringe) returns a solution, or failure

closed — an empty set
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, State[node]) then return node
    if State[node] is not in closed then
        add State[node] to closed
        fringe — InsertAll(Expand(node, problem), fringe)
    end
end
```

- Can this wreck completeness? Optimality?
Optimality of A* Graph Search

- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
  - Proof idea: optimal goals have lower f value, so get expanded first

We made a stronger assumption than in the last proof… What?

Consistency

- Wait, how do we know we expand in increasing f value?
- Couldn’t we pop some node \( n \), and find its child \( n' \) to have lower f value?
- YES:

\[
\begin{align*}
&g = 10 \\
&h = 0 \quad h = 8 \\
&3 \\
&A &\rightarrow &B &\rightarrow &G
\end{align*}
\]

- What can we assume to prevent these inversions?
- Consistency: \( c(n, a, n') \geq h(n) - h(n') \)
- Real cost always exceeds reduction in heuristic
Optimality

- **Tree search:**
  - A* optimal if heuristic is admissible (and non-negative)
  - UCS is a special case (h = 0)

- **Graph search:**
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)

- In general, natural admissible heuristics tend to be consistent

Summary: A*

- A* uses both backward costs and (estimates of) forward costs

- A* is optimal with admissible heuristics

- Heuristic design is key: often use relaxed problems
Large Scale Problems

- **What states get expanded?**
  - All states with f-cost less than optimal goal cost
  - How far “in every direction” will this be?
    - Intuition: depth grows like the heuristic “gap”:
      - \( h(\text{start}) - g(\text{goal}) \)
    - Gap usually at least linear in problem size
    - Work exponential in depth

- **In huge problems, often A* isn’t enough**
  - State space just too big
  - Can’t visit all states with f less than optimal
  - Often, can’t even store the entire fringe

- **Solutions**
  - Better heuristics
  - Beam search (limited fringe size)
  - Greedy hill-climbing (fringe size = 1)

\[
g(n) = g(n) + h(n) = d + \beta(x + d) \\
f(G^*) = f(n) \Rightarrow x = \frac{1 - \beta}{1 + \beta} x
\]

Limited Memory Options

- **Hill-Climbing Search:**
  - Only “best” node kept around, no fringe!
  - Usually prioritize successor choice by h (greedy hill climbing)
  - Compare to greedy backtracking, which still has fringe

- **Beam Search (Limited Memory Search)**
  - In between: keep K nodes in fringe
  - Dump lowest priority nodes as needed
  - Can prioritize by h alone (greedy beam search), or h+g (limited memory A*)
  - Why not applied to UCS?
  - We’ll return to beam search later…

- No guarantees once you limit the fringe size!