Search Gone Wrong?

CIS 471/571:
Artificial Intelligence

Fall 2012
Heuristic Search

Daniel Lowd
Based on slides from Luke Zettlemoyer, Dan Klein, Stuart Russell and Andrew Moore

Today

- A* Search
- Heuristic Design
- Graph Search

Recap: Search

- Search problem:
  - States (configurations of the world)
  - Successor function: a function from states to lists of (state, action, cost) triples; drawn as a graph
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)

Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location

Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one
Search Heuristics

- Any estimate of how close a state is to a goal
- Designed for a particular search problem

Examples: Manhattan distance, Euclidean distance

Best First / Greedy Search

Expand closest node first: Fringe is a priority queue

- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)

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 $h$ is admissible (optimistic) if:
  $$ h(n) \leq h^*(n) $$
  where $h^*(n)$ is the true cost to a nearest goal

- Examples:
  - Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A*: Blocking

Notation:
- $g(n) =$ cost to node $n$
- $h(n) =$ estimated cost from $n$ to the nearest goal (heuristic)
- $f(n) = g(n) + h(n) =$ estimated total cost via $n$
- $G^*$: a lowest cost goal node
- $G$: another goal node

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:
  - For all nodes $n$ on the best path to $G^*$
    - $f(n) < f(G)$
  - So, $G^*$ will be popped before $G$
  - $f(n) = g(n) + h(n)$
  - $g(n) + h(n) \leq g(G^*)$
  - $g(G^*) < g(G)$
  - $g(G) = f(G)$
  - $f(n) < f(G)$

Properties of A*

- Uniform-Cost
- A*

UCS vs A* Contours

- Uniform-cost expanded in all directions
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality
Which Algorithm?

- Uniform cost search (UCS):

Which Algorithm?

- A*, Manhattan Heuristic:

Which Algorithm?

- Best First / Greedy, Manhattan Heuristic:

Creating Heuristics

8-puzzle:

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

8 Puzzle I

- Heuristic: Number of tiles misplaced
- \( h(\text{start}) = 8 \)
- Is it admissible?

Average nodes expanded when optimal path has length...

\[
\begin{array}{c|c|c|c}
\text{TILES} & \text{UCS} & \text{A*} & \text{8-Puzzle II} \\
\hline
4 \text{ steps} & 112 & 6,300 & 13 \\
8 \text{ steps} & 3.6 \times 10^6 & 6,727 & 39 \\
12 \text{ steps} & & 227 & 227 \\
\end{array}
\]

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
- \( h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \)
- Admissible?
8 Puzzle III

- How about using the actual cost as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What’s wrong with it?
- With A*: a trade-off between quality of estimate and work per node!

Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available
- Inadmissible heuristics are often useful too (why?)

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

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 some nodes (which, and why?)
Graph Search

- Very simple fix: never expand a state type twice

```
Function GraphSearch(problem, fringe) returns a solution, or false
fringe = an empty set
 fringe = Insert(Make-Node(Initial-State(problem)), fringe)
while fringe is not empty:
    if fringe is empty then return false
    node = Remove-Front(fringe)
    if Problem-Solved(node) then return node
    foreach successor in Expand(node, problem):
        if state[node] not in closed
            fringe = Insert(Add-Node(successor, fringe), fringe)
```

- Can this wreck completeness? Why or why not?
- How about optimality? Why or why not?

A* Graph Search Gone Wrong

State space graph

```
S                        C
A------------------------B
     \                     \        
     1                   1
     \                   \       
    1                     2
```

Search tree

```
S (0+1)  A (1+2)  B (1+1)  C (2+1)  G (5+0)
     \         \       \     \       \      
     1         1       2       3       0     0
```

Optimality of A* Graph Search

Proof:
- Main idea: Argue that nodes are popped with non-decreasing f-scores
  - for all n, n' with n' popped after n:
    - f(n') ≥ f(n)
  - is this enough for optimality?
- Sketch:
  - assume: f(n') ≥ f(n), for all edges (n, a, n') and all actions a
  - is this true?
  - proof by induction: (1) always pop the lowest f-score from the fringe, (2) all new nodes have larger (or equal) scores, (3) add them to the fringe, (4) repeat!

Consistency

- Wait, how do we know parents have better f-values than their successors?

```
B (h=0, g=8)
      \   \ 
      3   A (g=10)
      \   \ 
      G (h=10)
```

Consistency for all edges (n, a, n'):
- h(n) ≤ c(n, a, n') + h(n')
- Proof that f(n') ≥ f(n),
  - f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') ≥ g(n) + h(n) = f(n)

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)
- Consistency implies admissibility
- 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