Informed search algorithms

Chapter 4
Outline

- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Memory Bounded A* Search
Best-first search

- **Idea:** use an *evaluation function* $f(n)$ for each node
  - $f(n)$ provides an estimate for the total cost.
  - Expand the node $n$ with smallest $f(n)$.

- **Implementation:**
  Order the nodes in fringe increasing order of cost.

- **Special cases:**
  - greedy best-first search
  - $A^*$ search
Romania with straight-line dist.
Greedy best-first search

- $f(n) =$ estimate of cost from $n$ to $goal$
- e.g., $f(n) =$ straight-line distance from $n$ to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal.
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Properties of greedy best-first search

- **Complete?** No – can get stuck in loops.
- **Time?** $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space?** $O(b^m)$ - keeps all nodes in memory
- **Optimal?** No
  
e.g. Arad → Sibiu → Rimnicu
  Virea → Pitesti → Bucharest is shorter!
Properties of greedy best-first search

\[ f(n) = \text{straightline distance} \]
A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) = \text{cost so far to reach } n$
  - $h(n) = \text{estimated cost from } n \text{ to goal}$
  - $f(n) = \text{estimated total cost of path through } n \text{ to goal}$
- Best First search has $f(n) = h(n)$
- Uniform Cost search has $f(n) = g(n)$
Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node $n$, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from $n$.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- **Theorem**: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal.
Admissible heuristics

E.g., for the 8-puzzle:
- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = 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)

- $h_1(S) = ? \quad 8$
- $h_2(S) = ? \quad 3 + 1 + 2 + 2 + 2 + 2 + 3 + 3 + 2 = 18$
Dominance

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible)
- then $h_2$ dominates $h_1$
- $h_2$ is better for search: it is guaranteed to expand less or equal nr of nodes.

- Typical search costs (average number of nodes expanded):
  - $d=12$
    - IDS = 3,644,035 nodes
    - $A^*(h_1) = 227$ nodes
    - $A^*(h_2) = 73$ nodes
  - $d=24$
    - IDS = too many nodes
    - $A^*(h_1) = 39,135$ nodes
    - $A^*(h_2) = 1,641$ nodes
Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original 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.
Consistent heuristics

- A heuristic is **consistent** if for every node $n$, every successor $n'$ of $n$ generated by any action $a$, \[ h(n) \leq c(n,a,n') + h(n') \]

- If $h$ is consistent, we have
  \[
  f(n') = g(n') + h(n') \quad \text{(by def.)} \\
  = g(n) + c(n,a,n') + h(n') \quad (g(n')=g(n)+c(n.a.n')) \\
  \geq g(n) + h(n) = f(n) \quad \text{(consistency)} \\
  \]
  \[
  f(n') \geq f(n) 
  \]

- i.e., $f(n)$ is non-decreasing along any path.

- **Theorem:**
  If $h(n)$ is consistent, A* using `GRAPH-SEARCH` is optimal.

  "It's the triangle inequality!"

  "keeps all checked nodes in memory to avoid repeated states"
A* search example
A* search example
A* search example
A* search example
A* search example
A* search example
Properties of A*

- **Complete?** Yes (unless there are infinitely many nodes with $f \leq f(G)$, i.e. step-cost $> \varepsilon$)
- **Time/Space?** Exponential $b^d$
  except if: $|h(n) - h^*(n)| \leq O(\log h^*(n))$
- **Optimal?** Yes
- **Optimally Efficient:** Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)
Optimality of A* (proof)

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

We want to prove:

$f(n) < f(G2)$

(then A* will prefer $n$ over $G2$)

- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $f(G) = g(G)$ since $h(G) = 0$
- $g(G_2) > g(G)$ since $G_2$ is suboptimal
- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ since $h$ is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$ from above
- $f(n) \leq f(G)$ since $g(n)+h(n)=f(n)$ & $g(n)+h^*(n)=f(G)$
- $f(n) < f(G2)$ from
Exercise

1) Consider the search tree to the right. There are 2 goal states, G1 and G2. The numbers on the edges represent step-costs. You also know the following heuristic estimates: 
\[ h(B \rightarrow G2) = 9, \ h(D \rightarrow G2) = 10, \ h(A \rightarrow G1) = 2, \ h(C \rightarrow G1) = 1 \]

a) In what order will A* search visit the nodes? Explain your answer by indicating the value of the evaluation function for those nodes that the algorithm considers.
The graph above shows the step-costs for different paths going from the start (S) to the goal (G). On the right you find the straight-line distances.

1. Draw the search tree for this problem. *Avoid repeated states.*

2. Give the order in which the tree is searched (e.g. S-C-B...-G) for A* search. Use the straight-line dist. as a heuristic function, i.e. $h=\text{SLD}$, and indicate for each node visited what the value for the evaluation function, $f$, is.
Memory Bounded Heuristic Search: Recursive BFS

- How can we solve the memory problem for A* search?
- Idea: Try something like depth first search, but let’s not forget everything about the branches we have partially explored.
- *We remember the best f-value we have found so far in the branch we are deleting.*
RBFS:

RBFS changes its mind very often in practice.

This is because the \( f = g + h \) become more accurate (less optimistic) as we approach the goal. Hence, higher level nodes have smaller \( f \)-values and will be explored first.

Problem: We should keep in memory whatever we can.
Simple Memory Bounded A*

- This is like A*, but when memory is full we delete the worst node (largest f-value).
- Like RBFS, we remember the best descendent in the branch we delete.
- If there is a tie (equal f-values) we delete the oldest nodes first.
- simple-MBA* finds the optimal "reachable" solution given the memory constraint.
- Time can still be exponential.

A Solution is not reachable if a single path from root to goal does not fit into memory.
function SMA*(problem) returns a solution sequence

inputs: problem, a problem

static: Queue, a queue of nodes ordered by $f$-cost

Queue $\leftarrow$ MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})

loop do
  if Queue is empty then return failure
  $n \leftarrow$ deepest least-$f$-cost node in Queue
  if GOAL-TEST($n$) then return success
  $s \leftarrow$ NEXT-SUCCESSOR($n$)
  if $s$ is not a goal and is at maximum depth then
    $f(s) \leftarrow \infty$
  else
    $f(s) \leftarrow \text{MAX}(f(n), g(s) + h(s))$
  if all of $n$’s successors have been generated then
    update $n$’s $f$-cost and those of its ancestors if necessary
  if SUCCESSORS($n$) all in memory then remove $n$ from Queue
  if memory is full then
    delete shallowest, highest-$f$-cost node in Queue
    remove it from its parent’s successor list
    insert its parent on Queue if necessary
  insert $s$ in Queue
end
Simple Memory-bounded A* (SMA*)

(Example with 3-node memory)

Progress of SMA*. Each node is labeled with its current f-cost. Values in parentheses show the value of the best forgotten descendant.

Algorithm can tell you when best solution found within memory constraint is optimal or not.
Conclusions

- The Memory Bounded A* Search is the best of the search algorithms we have seen so far. It uses all its memory to avoid double work and uses smart heuristics to first descend into promising branches of the search-tree.