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

- $f(n) = \text{estimate of cost from } n \text{ to } \textit{goal}$
- e.g., $f(n) = \text{straight-line distance from } n \text{ to Bucharest}$
- Greedy best-first search expands the node that \textit{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
GBFS is not complete

f(n) = straightline distance
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!
A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ cost so far to reach $n$
  - $h(n) =$ estimated cost from $n$ to goal
  - $f(n) =$ estimated total cost of path through $n$ to goal
- Best First search has $f(n)=h(n)$
- Uniform Cost search has $f(n)=g(n)$
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:
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance
  (i.e., no. of squares from desired location of each tile)

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

$$f(n') = g(n') + h(n') \quad \text{(by def.)}$$

$$= g(n) + c(n,a,n') + h(n') \quad \text{(g(n')=g(n)+c(n.a.n'))}$$

$$\geq g(n) + h(n) = f(n) \quad \text{(consistency)}$$

$$f(n') \geq f(n)$$

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

**Theorem:**
If $h(n)$ is consistent, A* using $\text{GRAPH-SEARCH}$ 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:
\[ f(n) < f(G_2) \]
(then A* will prefer \( n \) over \( G_2 \))

- \( f(G_2) = g(G_2) \) 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 (under-estimate)
- \( g(n) + h(n) \leq g(n) + h^*(n) \) from above
- \( f(n) \leq f(G) \) since \( g(n)+h(n)=f(n) \) & \( g(n)+h^*(n)=f(G) \)
- \( f(n) < f(G_2) \) from
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: 
  \[ h(B \rightarrow G2) = 9, \quad h(D \rightarrow G2) = 10, \quad h(A \rightarrow G1) = 2, \quad h(C \rightarrow G1) = 1 \]

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. $h=SLD$, 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 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 \text{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 \textit{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.
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.
- State space = set of "complete" configurations.
- Find configuration satisfying constraints, e.g., n-queens.
- In such cases, we can use **local search algorithms**.
- Keep a single "current" state, try to improve it.
- Very memory efficient (only remember current state).
Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.

Note that a state cannot be an incomplete configuration with $m<n$ queens.
Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima
Gradient Descent

- Assume we have some cost-function: $C(x_1, ..., x_n)$ and we want to minimize over continuous variables $X_1, X_2, ..., X_n$

1. Compute the gradient: $\frac{\partial}{\partial x_i} C(x_1, ..., x_n) \quad \forall i$

2. Take a small step downhill in the direction of the gradient:
   $$x_i \rightarrow x_i' = x_i - \lambda \frac{\partial}{\partial x_i} C(x_1, ..., x_n) \quad \forall i$$

3. Check if $C(x_1, ..., x_i', ..., x_n) < C(x_1, ..., x_i, ..., x_n)$

4. If true then accept move, if not reject.

5. Repeat.
Exercise

- Describe the gradient descent algorithm for the cost function:

\[ C(x, y) = \sqrt{(x - a)^2 + (y - b)^2} \]
Line Search

• In GD you need to choose a step-size.
  • Line search picks a direction, $v$, (say the gradient direction) and searches along that direction for the optimal step:

$$
\eta^* = \arg\min C(x_t + \eta v_t)
$$

• Repeated doubling can be used to effectively search for the optimal step:

$$
\eta \rightarrow 2\eta \rightarrow 4\eta \rightarrow 8\eta \ (\text{until cost increases})
$$

• There are many methods to pick search direction $v$. Very good method is “conjugate gradients”.
Hill-climbing search: 8-queens problem

- $h =$ number of pairs of queens that are attacking each other, either directly or indirectly ($h = 17$ for the above state)

Each number indicates $h$ if we move a queen in its corresponding column.
Hill-climbing search: 8-queens problem

- A local minimum with $h = 1$

  what can you do to get out of this local minima?
Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency.

- This is like smoothing the cost landscape.
Properties of simulated annealing search

- One can prove: If $T$ decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1 (however, this may take VERY long)

- Widely used in VLSI layout, airline scheduling, etc.
Local beam search

- Keep track of $k$ states rather than just one.
- Start with $k$ randomly generated states.
- At each iteration, all the successors of all $k$ states are generated.
- If any one is a goal state, stop; else select the $k$ best successors from the complete list and repeat.
Genetic algorithms

- A successor state is generated by combining two parent states
- Start with $k$ randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation
- Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times \frac{7}{2} = 28$)
- $P(\text{child}) = \frac{24}{24+23+20+11} = 31\%$
- $P(\text{child}) = \frac{23}{24+23+20+11} = 29\%$ etc
Given N cities and all their distances, find the shortest tour through all cities.

Try formulating this as a search problem. I.e., what are the states, step-cost, initial state, goal state, successor function.

Can you think of ways to try to solve these problems?
2. (5pts) **The 8-puzzle** Consider the 8-puzzle problem described in the book and homework.

   a. (1pt) We like to search for a solution using \( A^* \)-search. Describe the following aspects of the problem formulation: a) states, b) successor function, c) goal test, d) step cost, e) path cost.

   a) answer: a) a configuration of the 8 squares. b) all possible moves of the empty square, c) if end state has been reached, d) say 1 for each move, e) the total number of moves up till the current state.

   b. (2pts) For \( A^* \)-search the evaluation function, \( f(n) \), consists of the path cost, \( g(n) \), plus the heuristic function, \( h(n) \). We will use the Manhattan distance between the current state and the goal state to be the heuristic function. Explain when a heuristic function is admissible and prove this fact for the Manhattan distance heuristic.

   c. (2pts) Describe when a heuristic is consistent and proof this for the Manhattan distance heuristic.
1. (30pts) **Search** Consider a fantasy chess piece called “jumper”. It can move up, down, left, right, or it can stay wherever it is. Consider $k$ such jumpers on an infinite chessboard at positions $s_1, s_2, ..., s_k$. The goal is to move these jumpers as fast as possible to positions $g_1, g_2, ..., g_k$. In each move, you are allowed to move any number of jumpers simultaneously, but 2 or more jumpers can never occupy the same square.

a. (5pts) Formulate the above problem as a search problem, i.e. describe a state, the initial state, an action, the goal test and a path-cost.

a) answer: State: the position of the jumpers on the chessboard. Initial state: the given initial positions $s_1, ..., s_k$. Action: Any allowed move of $k$ jumpers to new positions. Goal test: Have the jumpers reached the positions $g_1, ..., g_k$? Path-cost: number of moves up to current position.

b. (5pts) Set $k = 1$, i.e. a single jumper on the board. What is the maximal branching factor for this problem?

b) answer: $b = 5$, the jumper can move into 4 positions or it can stay where it is.

c. (5pts) Still $k = 1$. Describe an admissible heuristic for this problem. This could be used inside an $A^*$ search algorithm.

d. (5pts) Now consider general $k$. What is now the maximal branching factor? Remember that up to $k$ jumpers can move simultaneously.

d) answer: $5^k$, because if all jumpers are sufficiently far away from each other, they can all move and each one has 5 choices, including not moving.

e. (5pts) Let $h_i \geq 0$ be an admissible heuristic for jumper $i$ if it were alone on the board (i.e. $k = 1$). Which of the following heuristics are admissible for $k$ jumpers simultaneously: I) $\max\{h_1, ..., h_k\}$, II) $\max\{h_1, ..., h_k\} - \min\{h_1, ..., h_k\}$? Explain! (Recall that jumpers are constrained and can move simultaneously).

f. (5pts) Assuming for a moment that both I) and II) are admissible, which heuristic will expand fewer nodes (or an equal number of nodes) when used in an $A^*$ algorithm?