COSC343: Artificial Intelligence

Lecture 18: Informed search algorithms

Alistair Knott

Dept. of Computer Science, University of Otago

In today's lecture

- Best-first search
- A* search
- Heuristics

Review: Uninformed search

Here's the general tree-search algorithm, as given in Lecture 13:

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
  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] applied to STATE(node) succeeds return node
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Recall:

- The search strategy depends on *the way nodes are added to the fringe*.
  (I.e. on the definition of INSERTALL.)

Informed search

The search algorithms we've looked at so far have all been uninformed, or brute-force: they systematically explore the state space without any *knowledge* about the particular problem being solved.

- This means they're extremely inefficient.

For most interesting problems, the state space is so large that brute force methods are far too slow.

- Often, knowledge about the problem can allow us to be a bit more smart about how to add nodes to the fringe.
- The basic idea: put nodes which are 'likely to lead to a good solution' towards the front.

Evaluation functions and best-first search

To be formal, we can define an evaluation function $f(n)$ on nodes, which takes a node in the search tree and returns a number.

- By convention, a low $f(n)$ means a good candidate for expansion.

We can now define a general type of search called best-first search, where the frontier is kept ordered in increasing order of $f(n)$.

Heuristics and greedy search

What do we mean by 'a node which is *likely* to lead to a solution'?

- If we knew *for certain* that it would lead to a solution, we wouldn't need to do the search.

The principles we use for ordering the fringe are often just 'rules of thumb', or 'rules that work most of the time'.

- A heuristic function gives an *estimate of the cheapest path cost from the current state to a goal state*.
- It's typically called $h(n)$.

In greedy search, the evaluation function is defined completely by a heuristic function. I.e. $f(n) = h(n)$.

An example of a heuristic function

What counts as a good estimate of the shortest cost to the goal state? The answer depends on the domain.

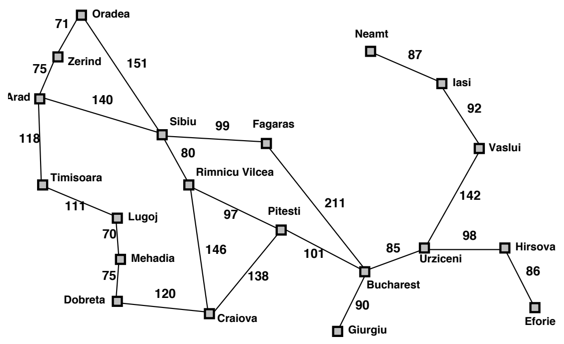
Recall the Romania state space - we're trying to get from Arad to Bucharest.

- A useful heuristic function here is the straight-line distance from a given town to the goal town (Bucharest).

Does this function *always* give the best choice for the next node to expand?

Does it *normally* give the best choice?

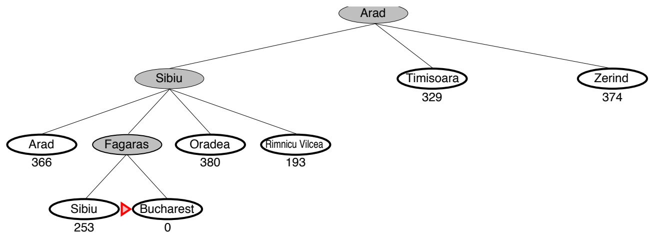
Romania, plus straight-line distances to Bucharest



Straight–line distance to Bucharest

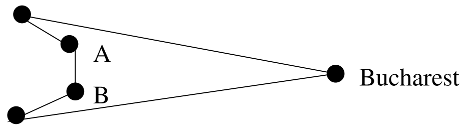
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search example



Properties of greedy search

- Complete?? No–can get stuck in loops



Complete in finite space with repeated-state checking
- Time?? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?? $O(b^m)$—keeps all nodes in memory
- Optimal?? No

A* search

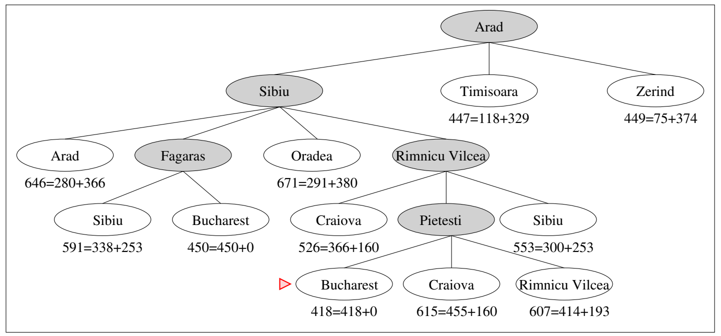
The main problem with greedy search is that it only looks *forward*, to the goal state.

- To produce an optimal search, it's also necessary to keep track of the path cost from the start node *to the current node*.
- We can define a function called $g(n)$, which returns the cost of the path from the start node to the current node.
- Note: $g(n)$ is not an estimate: we know this bit!

In A* search, the evaluation function $f(n)$ is defined as $g(n) + h(n)$.

- A*search is the starting point for all serious search algorithms.

A* search example



Admissible heuristics

One particularly interesting class of heuristics are those which are termed admissible.

- An admissible heuristic *never over-estimates* the cost of the path from the current state to the goal state.
- In other words, an admissible heuristic is always *optimistic*.

Note that straight-line distance is an admissible heuristic. (Because a straight line is the shortest distance between two points.)

Optimality of A*

If A*search uses an admissible heuristic, it is provably optimal: i.e. it is guaranteed to find the shortest path to the solution (if one exists).

Here's the proof.

- Say G_1 is the optimal goal state, and G_2 is a suboptimal goal state which is already on the fringe. And say that n is another node on the fringe which is on the shortest path to G_1.
- We must guarantee that A*will never pick G_2 from the fringe before picking n.
- We know that since G_2 is suboptimal, $g(G_2) + h(G_2)$ is greater than the cost of the actual shortest path to a solution.
- Since h is admissible, we know that $g(n) + h(n)$ is less than the actual shortest path to a solution.
- So A*will never pick G_2 before n.

Graph search with A*

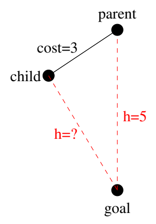
Recall that in tree search, we don't check for repeated states, while in graph search we only add a new node to the fringe if its state has not already been encountered.

- If we're interested in optimality, we must be careful when throwing away nodes in this way.
  - The two paths to the repeated state might have different costs.
  - We want to throw away the node with the higher $g(n)$.
- We can do this by explicitly comparing the $g(n)$ of each node.
- But another way is to ensure that the first time we reach a node, *we always pick the shortest route.*
  - We can prove this is the case if the heuristic used is consistent.

Consistent heuristics

It's useful to have a heuristic that evaluates nodes along a path in a way that's consistent with path costs.

Say a parent node has an h of 5, and it costs 3 to get to the child:



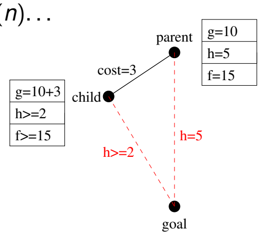
- It would be odd if the h value of the child was less than 5-3.
- It would be odd if the h value of the child was more than 5+3.

Consistent heuristics

A consistent heuristic is one which guarantees for any parent *p* and child *c* that

$$h(c) >= h(p) - \text{cost of path from p to c}$$

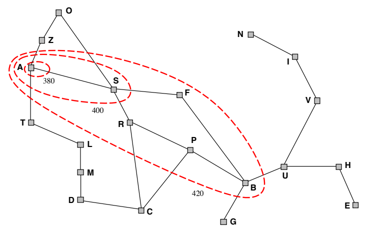
Now think what that means in relation to the *f* scores of the parent and child. $f(n) = g(n) + h(n)\ldots$



If *h* is consistent, the *f* values along any path are nondecreasing.

Consistent heuristics

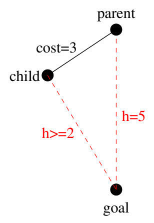
Given that A* orders the fringe in increasing order of $f(n)$, it follows that A* with a consistent heuristic proceeds by successive f-contours, just like breadth-first search proceeds by successive layers.



Because of this, we know that the first path we find to a given state will also be the shortest. (And we're safe to throw away re-encountered nodes without checking.)

The straight-line distance heuristic is consistent

This is easy to show geometrically:



$$h(child) >= h(parent) - \text{cost of path from parent to child}$$

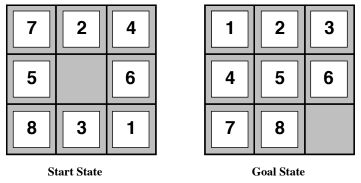
Properties of A*

- Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time?? Exponential in [relative error in $h \times$ length of soln.]
- Space?? Keeps all nodes in memory
- Optimal?? Yes—cannot expand f_{i+1} until f_i is finished
  A* expands all nodes with $f(n) < C^*$
  A* expands some nodes with $f(n) = C^*$
  A* expands no nodes with $f(n) > C^*$
  (where C^* is the path cost of the optimal solution).

Some example heuristics for the 8-puzzle

Here are a couple of admissible heuristics:

- The number of misplaced tiles.
- The total Manhattan distance of each tile from its goal location. (The Manhattan distance between two locations is the number of moves needed to get from one to the other, with no diagonal moves.)



In the above example, misplaced tiles = ?
Manhattan distance = ?

Relaxed problems

Admissible heuristics can be derived from the *exact* optimal solution cost of a *relaxed* version of the problem.

- If the rules of the 8-puzzle are relaxed so that a tile can move *to any location in a single jump*, then the number of misplaced tiles is the shortest solution.
- If the rules are relaxed so that a tile can move to *any adjacent square*, even if it's already occupied, then total Manhattan distance is the shortest solution.

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem. So it must be an admissible heuristic.

Summary and readings

- Informed search vs uninformed search
- Greedy search and A* search
- Admissible and consistent heuristics for guaranteeing optimality of A* search

For this lecture, you should read AIMA Sections 3.5–3.6.
For next lecture: AIMA Sections 5.1–5.3.