

COMP210: Artificial Intelligence

Lecture 8. Heuristic search

Trevor Bench-Capon
Room 215, Ashton Building

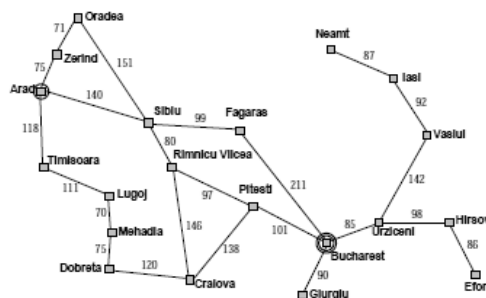
<http://www.csc.liv.ac.uk/~tbc/COMP210>

- **Last Time**
 - **Breadth-first search**
 - complete but expensive.
 - **Depth-first search**
 - cheap but incomplete
 - **Limited depth search**
 - **Iterative deepening search**
- **Today**
 - Show how applying knowledge of the problem can help
 - Introduce uniform cost search: dependent on the cost of each node
 - Introduce **heuristics**: rules of thumb
 - Introduce **heuristic search**
 - greedy search
 - A* search

Real Life Problems

- Whatever search technique we use, we have exponential time complexity.
- Tweaks to the algorithm will not reduce this to polynomial.
- We need problem specific knowledge to **guide** the search.
- Simplest form of problem specific knowledge is heuristic.
- Usual implementation in search is via an **evaluation function** which indicates desirability of expanding a node.

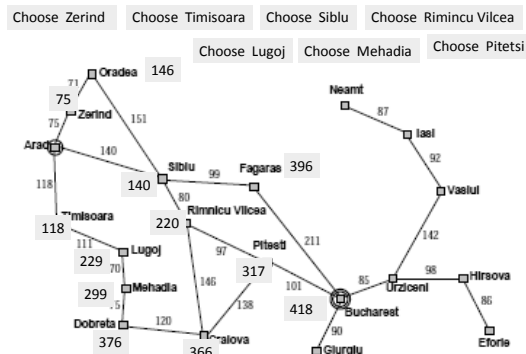
Path Cost Function gives the cost of each path



Finding The Best Paths

- Why not expand the *cheapest path first*?
- Intuition: cheapest is likely to be best!
- Performance is like breadth-first search but we use the minimum cost path rather than the shallowest to expand.
- **Uniform cost search** behaves the same as breadth-first search since the cost of every step is the same.

Cheapest first



General Algorithm for Uniform Search

```

agenda = initial state;
while agenda not empty do
  take node from agenda such that
    g(node) = min { g(n) | n in
      agenda}
  if node is goal state then return
    solution;
  new nodes = apply operations to
  node;
  add new nodes to the agenda;
  
```

Properties of Uniform Search

- Uniform cost search guaranteed to find cheapest solution assuming path costs grow monotonically, i.e. the cost of a path never decreases as we move along it.
- In other words, adding another step to the solution makes it more costly, i.e. $g(\text{successor}(n)) > g(n)$.
- If path costs don't grow monotonically, then exhaustive search is required.
- Still requires many nodes to be examined

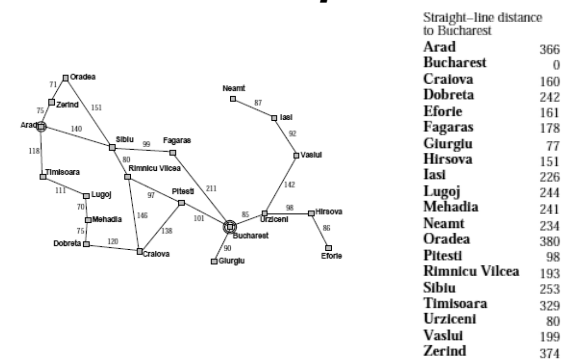
Informed Strategies

- Use problem-specific knowledge
- More efficient than blind search
- The most **promising** path first!
- Rather than trying all possible search paths, you try to focus on paths that seem to be getting you nearer your target/goal state.

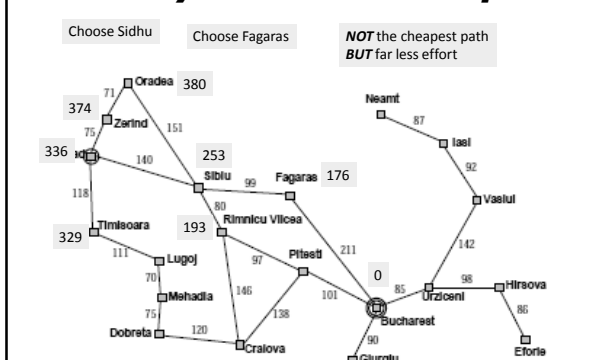
Greedy Search

- Most heuristics *estimate cost of cheapest path from node to solution.*
- We have a *heuristic function*, $h : \text{Nodes} \rightarrow \mathbb{R}$ which estimates the distance from the node to the goal.
- h can be any function but should have $h(n) = 0$ if n is a goal.
- Example: In route finding, heuristic might be straight line distance from node to destination.
- Greedy search expands the node that *appears to be* closest to goal

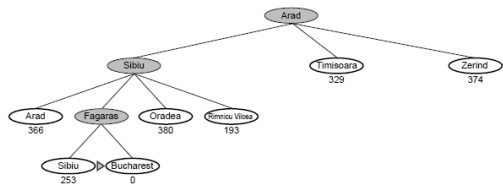
Romania Example



Greedy Search Example



Search Tree



General algorithm for greedy search

```

agenda = initial state;
while agenda not empty do
  take node from agenda such that
  h(node) = min { h(n) | n in
  agenda}
if node is goal state then
  return solution;
new nodes = apply operations to
node;
add new nodes to the agenda;
    
```

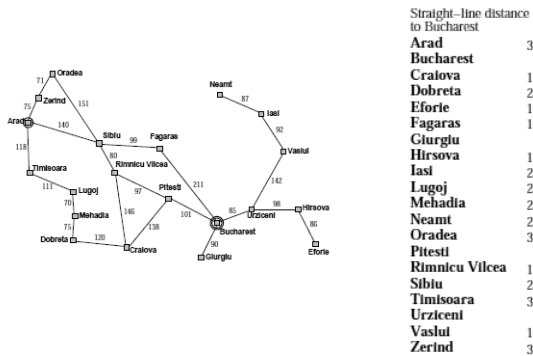
Properties of Greedy Search

- Greedy search finds solutions quickly.
- Doesn't always find best.
- May not find a solution if there is one (incomplete).
- Susceptible to false starts.
- Only looking at *current node*. *Ignores past!*
- *Short sighted*.

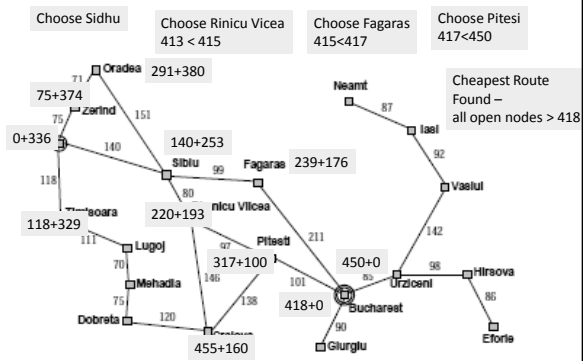
A* Search

- A* is a very efficient search strategy.
- Basic idea is to combine uniform cost search and greedy search.
- We look at the cost so far **and** the estimated cost to goal.
- Gives heuristic f:
 - $f(n) = g(n) + h(n)$
- where
 - $g(n)$ is path cost of n;
 - $h(n)$ is expected cost of cheapest solution from n.
- Aims to minimise overall cost.

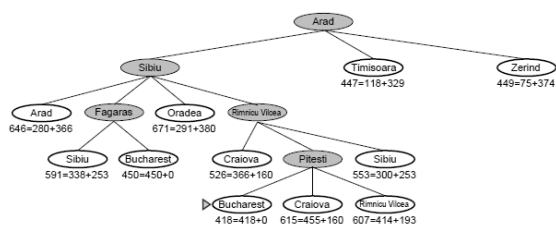
Romania Example



A* Search Example



Search Tree



General algorithm for A* search

```

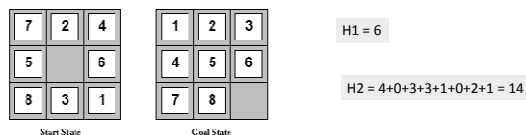
agenda = initial state;
while agenda not empty do
  take node from agenda such that
  f(node) = min { f(n) | n in agenda}
  where f(n) = g(n) + h(n)
  if node is goal state then
    return solution;
  new nodes = apply operations to node;
  add new nodes to the agenda;
  
```

Properties of A* search

- Complete provided
 - only finitely many nodes with $f < f(G)$
 - an admissible heuristic is used
 - Never overestimates the distance
 - i.e., $h(n) < true(n)$,
 - where $true(n)$ is the true cost from n .
 - Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G .
- Exponential time
- Keeps all nodes in memory
- Optimal

Admissible heuristics

- E.g., for the 8-puzzle:
 - $h1(n)$ = number of misplaced tiles
 - $h2(n)$ = total *Manhattan distance*
 - (i.e., no. of squares from desired location of each tile)



Importance of the Heuristic Choice

- Typical search costs:
 - $d = 14$
 - IDS = 3,473,941 nodes
 - $A(h1) = 539$ nodes
 - $A(h2) = 113$ nodes
 - $d = 24$
 - IDS = 54,000,000,000 nodes
 - $A(h1) = 39,135$ nodes
 - $A(h2) = 1,641$ nodes

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
 - incomplete and not always optimal
- A* search expands lowest $g + h$
 - complete and optimal also optimally efficient
- Next Week
 - Applying search to games