COMP219: Artificial Intelligence

Lecture 10: Heuristic Search

Overview

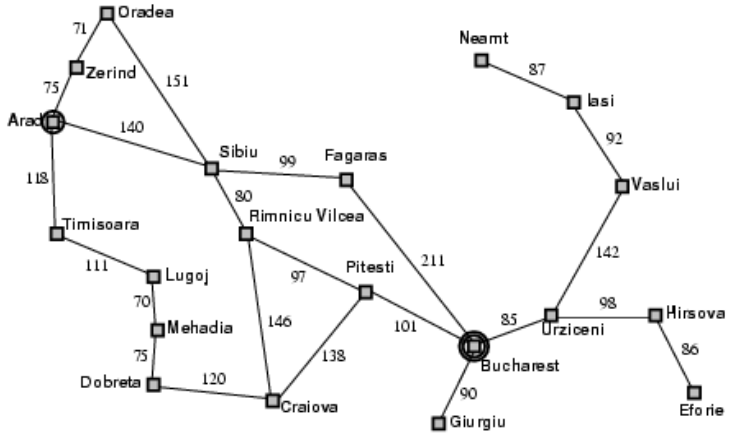
- Last time
  - Depth-limited, iterative deepening and bi-directional search
  - Avoiding repeated states
- Today
  - Show how applying knowledge of the problem can help
  - Introduce uniform cost search: dependent on the cost of each node
  - Introduce heuristics: rules of thumb
  - Introduce heuristic search
    - Greedy search
    - A* search
- Learning outcome covered today:
  Identify, contrast and apply to simple examples the major search techniques that have been developed for problem-solving in AI

Real Life Problems

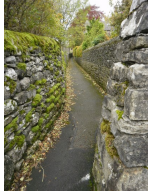
- Whatever search technique we use, we have exponential time complexity
- Tweaks to the algorithm will not reduce this to polynomial
- We need problem specific knowledge to ***guide*** the search
- Simplest form of problem specific knowledge is heuristics
- Usual implementation in search is via an ***evaluation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

Path Cost Function

Recall: we have a path cost function, which gives the cost of each path. This is comprised of the step costs of each action on the path.



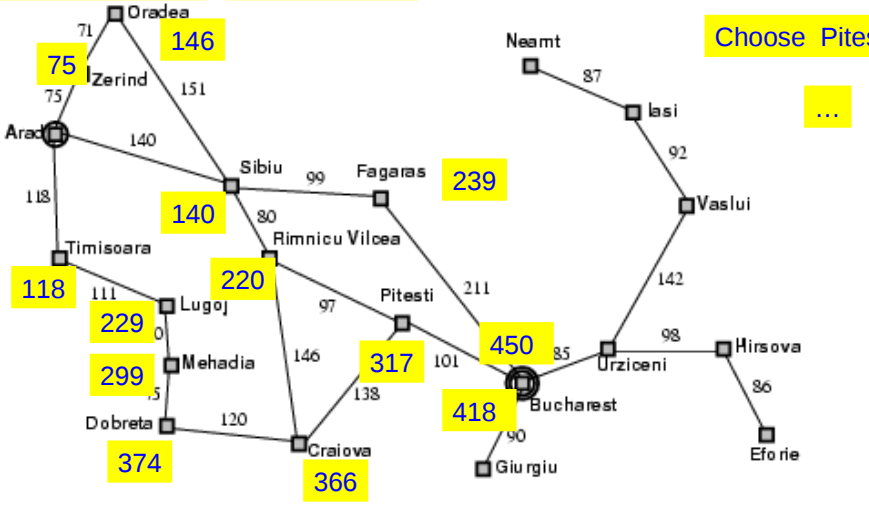
Finding The Best Paths



- 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** orders the agenda as a priority queue using the lowest path cost of a node

Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea | Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras | Choose Mehadia | Choose Pitesti | …



General Algorithm for Uniform Cost 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 Cost 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(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

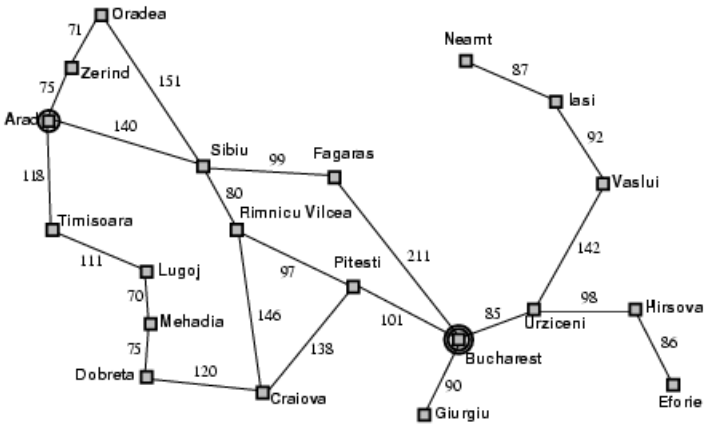
Greedy Search

- Most heuristics *estimate cost of cheapest path from node to solution*
- We have a *heuristic function*,

  $h : Nodes \rightarrow R$

  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



Straight-line distance to Bucharest

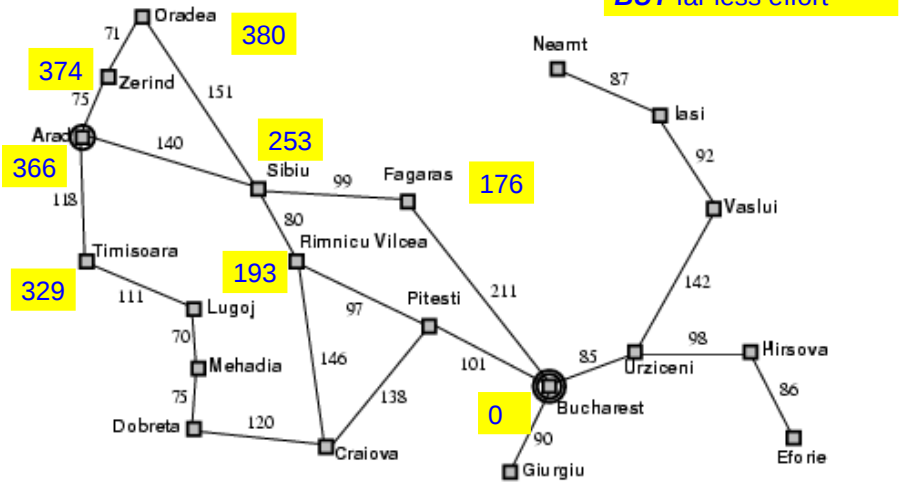
Arad	**366**
Bucharest	**0**
Craiova	**160**
Dobreta	**242**
Eforie	**161**
Fagaras	**176**
Giurgiu	**77**
Hirsova	**151**
Iasi	**226**
Lugoj	**244**
Mehadia	**241**
Neamt	**234**
Oradea	**380**
Pitesti	**98**
Rimnicu Vilcea	**193**
Sibiu	**253**
Timisoara	**329**
Urziceni	**80**
Vaslui	**199**
Zerind	**374**

Greedy Search Example

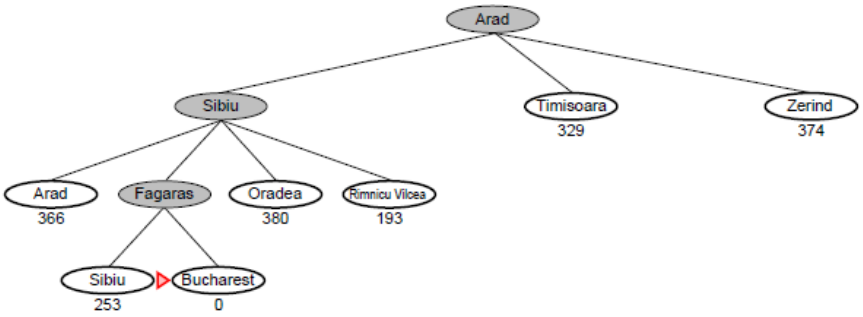
Choose Sibiu

Choose Fagaras

NOT the cheapest path
BUT far less effort



Search Tree



Exercise

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 the 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

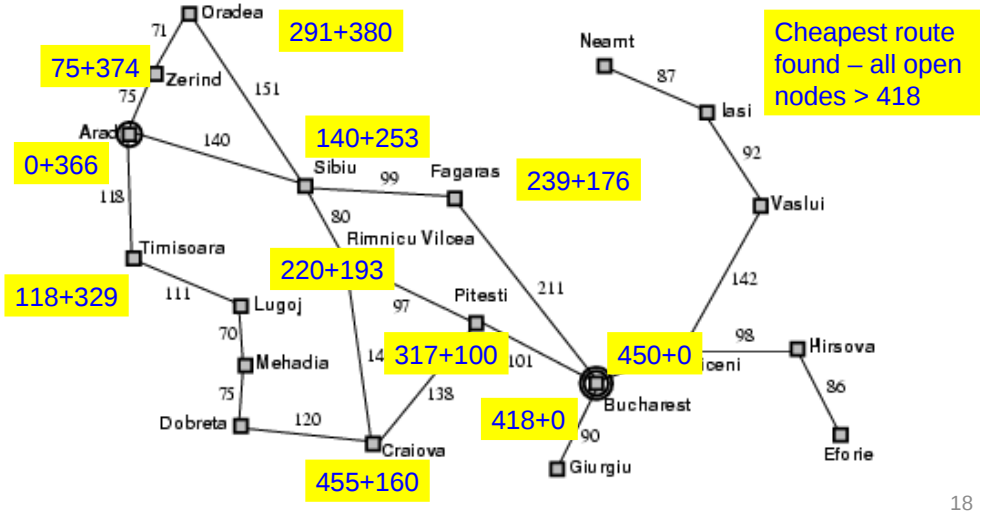
A* Search Example

Choose Sibiu

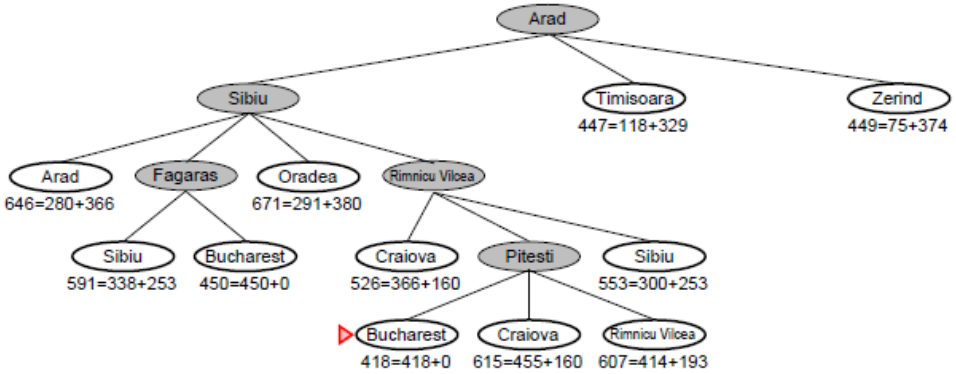
Choose Rimnicu Vilcea
413 < 415

Choose Fagaras
415 < 417

Choose Pitesti
418 < 450



Search Tree



General Algorithm for A* Search

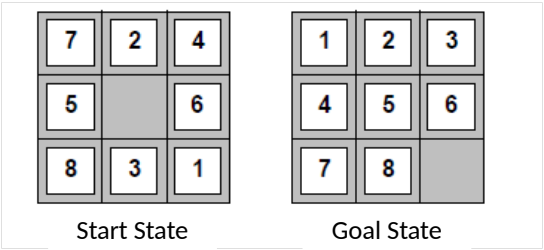
```
agenda = initial state;
while agenda not empty do
   take node from agenda such that
     f(node) = min { f(n) | n in agenda}
     where f(n) = g(n) + h(n)
   if node is goal state then
    return solution;
   new nodes = apply operations to node;
   add new nodes to the agenda;
```

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) ≥ 0`, so `h(G) = 0` for any goal `G`
- Exponential time
- Keeps all nodes in memory
- Optimal

Admissible Heuristics

- Example - 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)



Exercise

Importance of the Heuristic Choice

- Typical search costs:
  - *d = 14*
    - IDS = 3,473,941 nodes
    - $A^*(h_1)$ = 539 nodes
    - $A^*(h_2)$ = 113 nodes
  - *d = 24*
    - IDS ≈ 54,000,000,000 nodes
    - $A^*(h_1)$ = 39,135 nodes
    - $A^*(h_2)$ = 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
  - Search in complex environments (partial observation) and in game playing