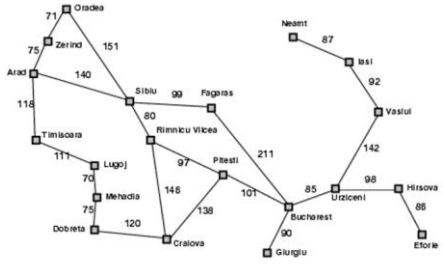
Introduction to Artificial Intelligence

Unit # 9

Acknowledgement

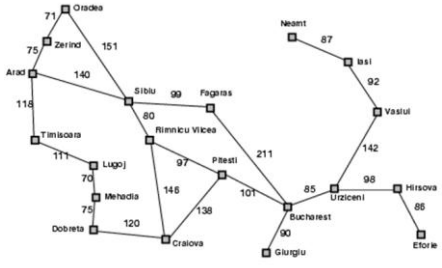
- The slides of this lecture have been taken from the lecture slides of
  - CS307 – "Introduction to Artificial Intelligence" by Dr. Sajjad Haider.
  - CS 121 – "Introduction to Artificial Intelligence" by Teg Grenager(Slides from Dan Klein, Stuart Russell, Andrew Moore), Stanford University
  - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Romania with step costs in km (Russell & Norvig)



Path Planning

- How to find the best path from 'Arad' to Bucharest' ?



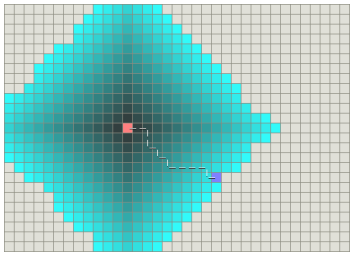
Exhaustive Search

- Exhaustive search would look for all possible paths from initial to goal node to find the optimal path.

Dijkstra's algorithm

- Dijkstra's algorithm works by visiting vertices in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined. It expands outwards from the starting point until it reaches the goal.
- Dijkstra's algorithm is guaranteed to find a shortest path from the starting point to the goal, as long as none of the edges have a negative cost.

Dijkstra's algorithm



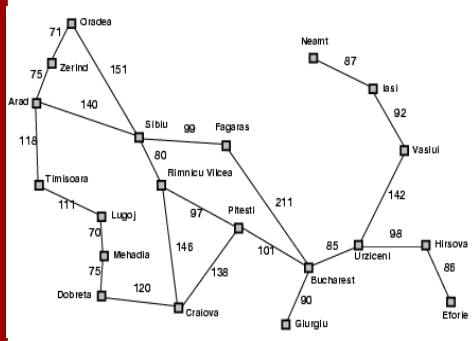
Informed vs Uninformed Search

- Search methods can be either:
  - Informed or
  - Uninformed
- A search method or heuristic is **informed** if it uses additional information (heuristics) about nodes that have not yet been explored to decide which nodes to examine next.
- If a method is not informed, it is **uninformed** or **blind**.

Basic Idea behind Informed Searches

- We assume that we have a heuristic (evaluation) function, f, to help decide which node is the best one to expand next.
- Heuristic is an "***estimate of the proximity of the goal***".
- Expand next that node, n, having the smallest value of f(n). Resolve ties arbitrarily.
- Terminate when the node to be expanded next is a goal node.

Using Heuristics



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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Best-first Search

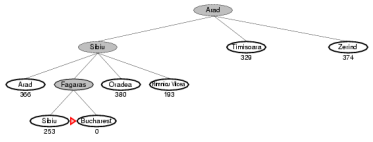
- Best-first search expands the node that appears to be closest to goal
- Evaluation function *f(n) = h(n)* (heuristic)
  = estimate of cost from *n* to *goal*

Best-first search example

Best-first search example

Best-first search example

Best-first search example

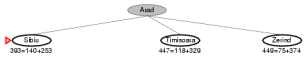


A* search

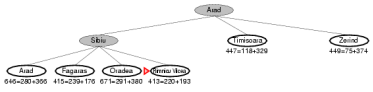
- Idea: avoid expanding paths that are already expensive
- Evaluation function *f(n) = g(n) + h(n)*
  - *g(n)* = cost so far to reach *n*
  - *h(n)* = estimated cost from *n* to goal
  - *f(n)* = estimated total cost of path through *n* to goal

A* search example

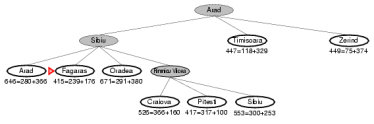
A* search example



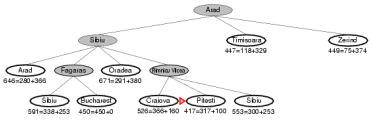
A* search example



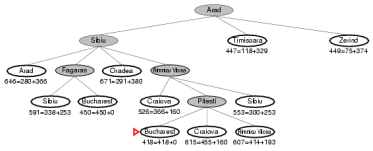
A* search example



A* search example



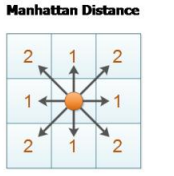
A* search example



Heuristics for grid maps

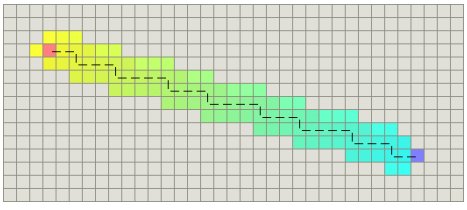
- On a grid, there are well-known heuristic functions to use:
  - On a square grid that allows **4 directions** of movement, use Manhattan distance
  - On a square grid that allows **8 directions** of movement, use Diagonal distance
  - On a square grid that allows **any direction** of movement, you might want Euclidean distance.

Manhattan Distance

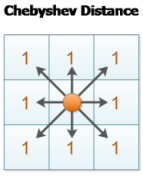


$$|x_1 - x_2| + |y_1 - y_2|$$

Manhattan Distance

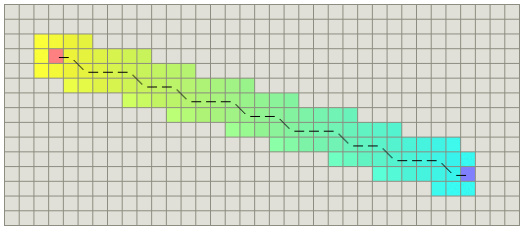


Diagonal Distance

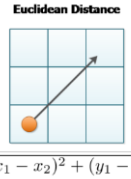


$$\max(|x_1 - x_2|, |y_1 - y_2|)$$

Diagonal Distance



Euclidean Distance



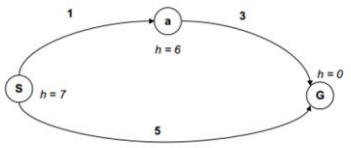
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Since Euclidean distance gives straight line distance which is the minimum distance between two points, it always gives you an admissible heuristic for A*.

Search Demo

http://www.cs.rochester.edu/u/kautz/Mazes/search_algorithm_demo.htm

Is A* Optimal ?



- A* Search orders by the sum: f(n) = g(n) + h(n)
- What went wrong?
- We need estimates to be less than actual costs!

Optimality of A*

A* search is optimal if the heuristic it uses is admissible.

- A heuristic h is *admissible* if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

Playing with A* Heuristics

- The heuristic can be used to control A*'s behavior.
  - At one extreme, if h(n) is 0, then only g(n) plays a role, and A* turns into Dijkstra's algorithm, which is guaranteed to find a shortest path.
  - If h(n) is always lower than (or equal to) the cost of moving from n to the goal, then A* is guaranteed to find a shortest path. The lower h(n) is, the more node A* expands, making it slower.
  - If h(n) is exactly equal to the cost of moving from n to the goal, then A* will only follow the best path and never expand anything else, making it very fast. It's nice to know that given perfect information, A* will behave perfectly.
  - If h(n) is sometimes greater than the cost of moving from n to the goal, then A* is not guaranteed to find a shortest path, but it can run faster.
  - At the other extreme, if h(n) is very high relative to g(n), then only h(n) plays a role, and A* turns into Best-First-Search.

Playing with A* Heuristics

- Suppose your game has two types of terrain, Flat and Mountain, and the movement costs are 1 for flat land and 3 for mountains, A* is going to search three times as far along flat land as it does along mountainous land. This is because it's *possible* that there is a path along flat terrain that goes around the mountains. You can speed up A*'s search by using 1.5 as the heuristic distance between two map spaces. A* will then compare 3 to 1.5, and it won't look as bad as comparing 3 to 1. It is not as dissatisfied with mountainous terrain, so it won't spend as much time trying to find a way around it.
- Alternatively, you can speed up A*'s search by decreasing the amount it searches for paths around mountains—just tell A* that the movement cost on mountains is 2 instead of 3. Now it will search only twice as far along the flat terrain as along mountainous terrain. Either approach gives up ideal paths to get something quicker.

A* Pseudocode

```
OPEN = priority queue containing START
CLOSED = empty set
while lowest rank in OPEN is not the GOAL:
  current = remove lowest rank item from OPEN
  add current to CLOSED
  for neighbors of current:
    cost = g(current) + movementcost(current, neighbor)
    if neighbor in OPEN and cost less than g(neighbor):
      remove neighbor from OPEN, because new path is better
    if neighbor in CLOSED and cost less than g(neighbor): **
      remove neighbor from CLOSED
    if neighbor not in OPEN and neighbor not in CLOSED:
      set g(neighbor) to cost
      add neighbor to OPEN
      set priority queue rank to g(neighbor) + h(neighbor)
      set neighbor's parent to current

reconstruct reverse path from goal to start
by following parent pointers
```