COMP2511

Object-Oriented Design and Programming

Problem-Solving Algorithms

Wayne Wobcke

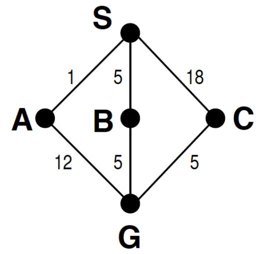
w.wobcke@unsw.edu.au

Today's Lecture

- Uniform-Cost Search
- Greedy Best-First Search
- A* Search
- Heuristics
  - Admissibility, Dominance, Consistency
- Assignment 2

Uniform-Cost Search

- Breadth First Search + Costs
  - Cost function *g(n)* of nodes: ***path cost***
  - Use PriorityQueue<Node> ordered by *g(n)*
  - Why we can't stop until the goal state is taken **off** the queue



Greedy Best-First Search

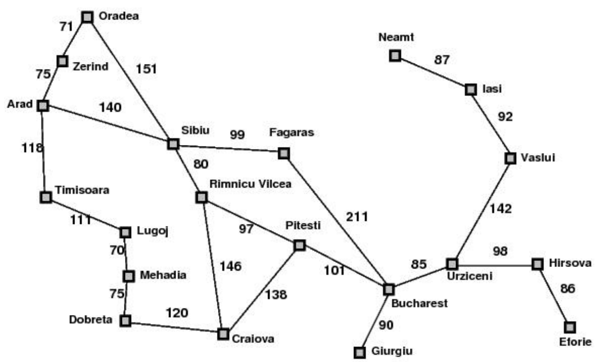
- Hill climbing towards nearest goal state
  - Heuristic function *h(n)*: estimate to goal
  - *h(n)* = 0 if *n* is a goal state
- PriorityQueue<Node> ordered by *h(n)*
- Won't find optimal solution
- Tends to follow a single path
- Problems with ridges and plateaus

Heuristics

2	8	3		1	2	3
1	6	4	→	8		4
	7	5		7	6	5

1. Number of tiles out of place
2. Manhattan distance of tiles out of place

Romania Map



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

- Does not find optimal path

A* Search

- Uniform-Cost + Greedy Best-First
  - *f(n) = g(n) + h(n)*
  - Use PriorityQueue<Node> ordered by *f(n)*
  - *f(n)* estimates cost of shortest path from initial state to a goal state that includes *n*
- Will find optimal solution
- Behaves like breadth-first

A* Search

OPEN – nodes on frontier; CLOSED – expanded nodes;

while OPEN is not empty
  remove *<n, p>* from head of queue (minimal *f(n)*)
  add *<n, p>* to CLOSED
  if *n* is a goal state return success path *p*
  for each edge *e* from *n* to *n'* (with cost *c*, so *g(n') = g(n) + c*)
    if *<n', p'>* is on CLOSED then if *f(n', p+e) < f(n', p')*
      remove *<n', p'>* from closed and add *<n', p+e>* to OPEN
    else if *<n', p'>* is on OPEN then if *f(n', p+e) < f(n', p')*
      replace *<n', p'>* by *<n', p+e>* on OPEN
    else if *n'* is not on OPEN add *<n', p+e>* to OPEN
return failure

Admissibility

- Heuristic function $h(n)$ is *admissible*
  - $h(n) \leq h^*(n)$ for every n, where $h^*(n)$ is the actual cost from n to a goal state, i.e.
  - h never overestimates distance to a goal
- Guarantees A* finds an optimal solution
- Make sure heuristics are admissible!
- Zero is admissible but not very good

Dominance

- Heuristic function h_2 *dominates* h_1
  - $h_2(n) \geq h_1(n)$ for every n
- A* expands no more nodes with h_2 vs h_1
- Higher valued heuristics are better
- Provided they are admissible!
- Can combine h_1, h_2 taking $max(h_1, h_2)$

Consistency

- Heuristic function h is *consistent*
  - $f(n)$ is always non-decreasing along a path
  - Triangle inequality: $h(n) \leq h(n') + cost(n, n')$
- First found path to n is always optimal path to any node with the same state
- No need to test (or use?) CLOSED
- ... but can be hard to find heuristics

Generating Heuristics

- Use exact solution to relaxed problem
  - Move tile anywhere ≈ tiles out of place
  - Move tile one space anywhere ≈ Manhattan
  - Ignore roads ≈ “crow flies” distance
- Still need to ensure admissible!

Assignment 2

- Design before coding – follow the object-oriented design process
- Make sure you read from a file args[0], write to System.out
- Make sure main() is in ShipmentPlanner.java
- Don't use a package – use only **the** default package
- Make sure the submitted files include java files and a pdf file
- Do implement A*, not a "heuristic" approximation algorithm
- Don't use the same problem space as for the Romania map
- States are partial schedules, edges add to a schedule
- Make sure your heuristic is admissible!
- Time limit of 10 secs to assess efficiency and heuristic
- Don't sacrifice understandability of code for efficiency

Next Week

- Software Project Management
  - Agile Methods & Scrum
- Toyota Production System