Informed Searches

CHAPTER 3 CONTINUED
COSC 370
SPRING 2019
ALAN C. JAMIESON

SOME SLIDE CONTENT FROM RUSSELL & NORVIG PROVIDED SLIDES

1

- Tree Search
- Greedy Best-First Search
- A*
- Improvements to A*

2

Tree Search

```
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)
```

A strategy is defined by picking the **order of node expansion**

3

Strategy Evaluation

- Strategies can be evaluated based on:
  - Completeness – does it always find a solution if one exists?
  - Time it takes
  - Space it takes
  - Optimality
- For trees time and space can be evaluated in terms of branching factor (b), depth (d), and maximum depth of the state space (m).

4

Informed Searches

- a.k.a Heuristic Search
- Expansion based on a function f(n) – evaluation function.
- A part of this function may be a heuristic h(n):

  h(n) = estimated cost of cheapest path from n to goal

- Constant: h(n) = 0 for goal state

5

Best-First Search

- Evaluation Function for each node – estimating desirability.
- Expand most desirable unexpanded node
- Implementation – use a queue! Sort by desirability.
- Specific Cases:
  - Greedy
  - A*

6

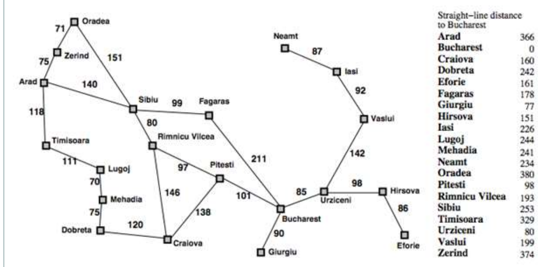
Greedy Best-first Search

- Greedy – expand the node that is closest to the goal.
- f(n) = h(n) = *estimate* of cheapest path
- Example: Romanian Vacation Problem

  h_{sld} = Straight line distance between two cities

- Take home – Greedy expands the node that appears to be the closest to the goal.

7

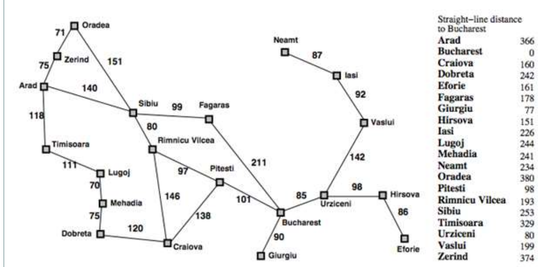


8

Analysis of Technique

- Complete – nope, can run into loops (example: take RVP with Oradea as goal)

9



10

Analysis of Technique

- Complete – nope, can run into loops (example: take RVP with Oradea as goal)
- Time –*$O(b^m)$*, though heuristics can improve this
- Space – *$O(b^m)$*, all states in memory
- Optimal – not necessarily

11

A*

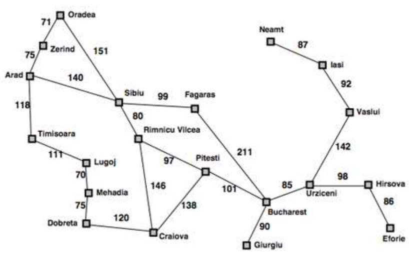
- An extension of Dijkstra's shortest path algorithm developed by Hart, Nilsson, and Raphael (SRI).
- General Idea: try not to go down the expensive paths!
- Evaluation function:

  f(n) = h(n) + g(n)

  h(n) = estimated cost from n to goal
  g(n) = cost so far to reach n

- A* uses an admissible heuristic – one that never overestimates the cost to reach the goal.

12



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

13

Exercise!

- Here's a starter 8-puzzle. How would A* solve this puzzle? What's the heuristic?

4	3	7
2		1
5	6	8

14

Two More Heuristic Problems

- You're given a maze with one entry and one exit. Possible moves are cardinal directions only. What's the heuristic?
- 9x9 sudoku solver. What's the heuristic?

15

Analysis of Technique

- Complete – yes, assuming finite number of states
- Time – potentially exponential – why so long?
  - while efficient and optimal, the number of potential states (think: paths) that have to be expanded/searched is still exponential.
- Space – $O(b^m)$, all states in memory
- Optimal – yes!

16

Some additional terminology

- Dominance – assuming that we have admissible heuristics, if h2(n) >= h1(n) for any node n, then h2(n) dominates h1(n) and is "better".
- Consistency (aka monotonicity) – a heuristic is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'.
- Relaxed problems

17

Memory-bounded Heuristic

- One issue with A* - memory usage!
- Briefly – Iterative-Deepening A* (IDA*)
  - Iterative Deepening Search (see last set of slides)
  - Cutoff used – f(n) rather than the depth
- Recursive best-first search (RBFS) – simple recursive algorithm for BFS in linear space.
- Memory-Bounded A* (MA*) & Simplified MA* (SMA*) – A* until memory is full, then expand by dropping the "worst" leaf node.

18