University of Strathclyde Glasgow

CS310 – AI Foundations

Andrew Abel

January 2024

Week 4: A* Search

Welcome!

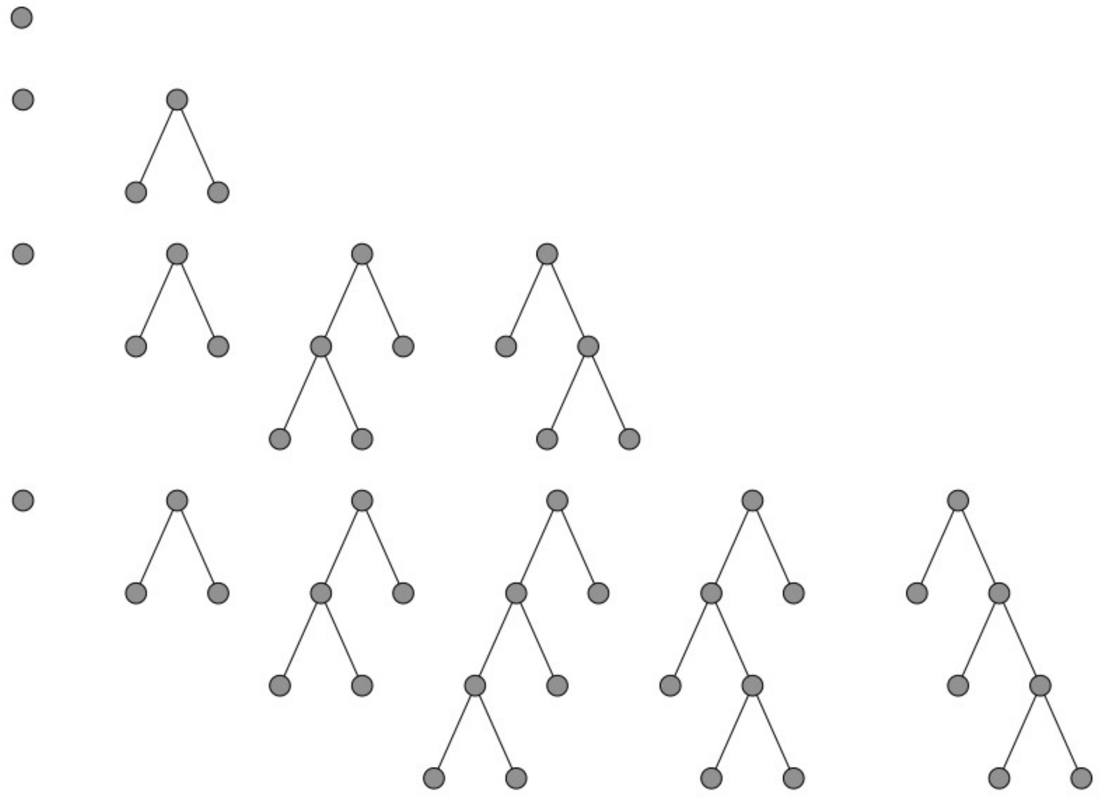
- Welcome to CS310 week 4!
- This week we will be introducing more costs and heuristics
  - Best first Search
  - A Algorithm
  - A* Algorithm
- Adversarial Search
  - Game Trees

Last Week

- Iterative Deepening Search
- Visited list
- Introduction to Heuristics
- Hill Climbing

Iterative Deepening

- Algorithm: do (repeated) depth-limited search, but with increasing depth.
- Expands nodes multiple times, but time complexity is of the same order of magnitude



Breadth-First Search with visited list

```
let Agenda = [$S_0$]
let Visited  $= []
while Agenda ≠ [] do
    let Current = remove-first (Agenda)
    let Agenda = rest(Agenda)
    if Goal (Current) then return ("Found it!")
    let Next = NextStates (Current)
    let Agenda = Agenda + Next – Visited)
    Visited.append(Current)
```

Making Search cost-aware: Uniform-cost search

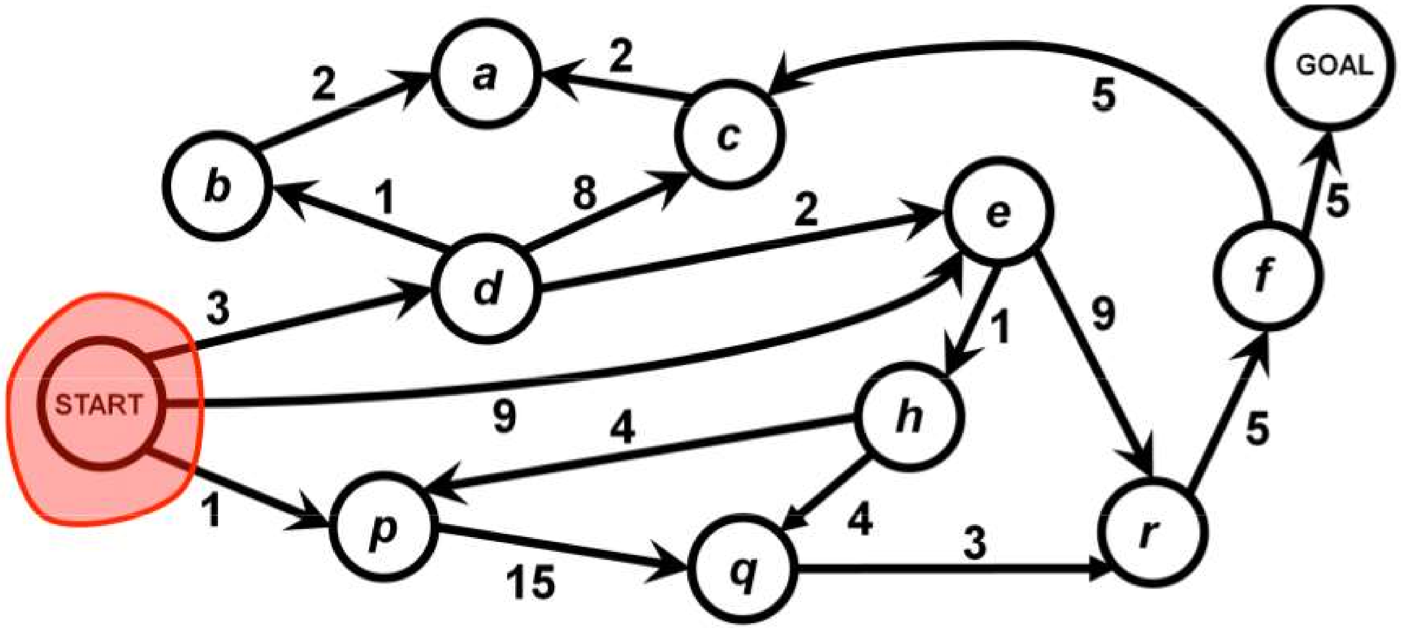
- Goal: find the ``cheapest'' path from the initial state to a goal state.
- Idea: same algorithm as BFS but we use a priority queue as agenda
  - Previously, we added states to the agenda arbitrarily
    - Simply gone Left to Right in our examples
    - Because it doesn’t matter!
  - Now we consider costs when deciding order
- Nodes are added to agenda in increasing order of path so far
  - Agenda is now ordered in terms of priority
  - Or we find cheapest solution
- Guaranteed to find an optimal solution!

Uniform-cost search

```
● let Agenda = [($S_0$,0)]
● let Visited  $= []
● while Agenda ≠ = [ ]:
    ○ let (Current,i) = lowest-cost (Agenda)
    ○ let Agenda = remainder(Agenda)
    ○ if Goal (Current): return ("Found it!")
    ○ if Current not in Visited:
        ■ let Next = NextStatesWithCosts(Current)
        ■ let Agenda = Agenda + Next
        ■ Visited.append(Current)
```

Example of algorithm variant

- PQ = Priority Queue
- Want to reach goal state



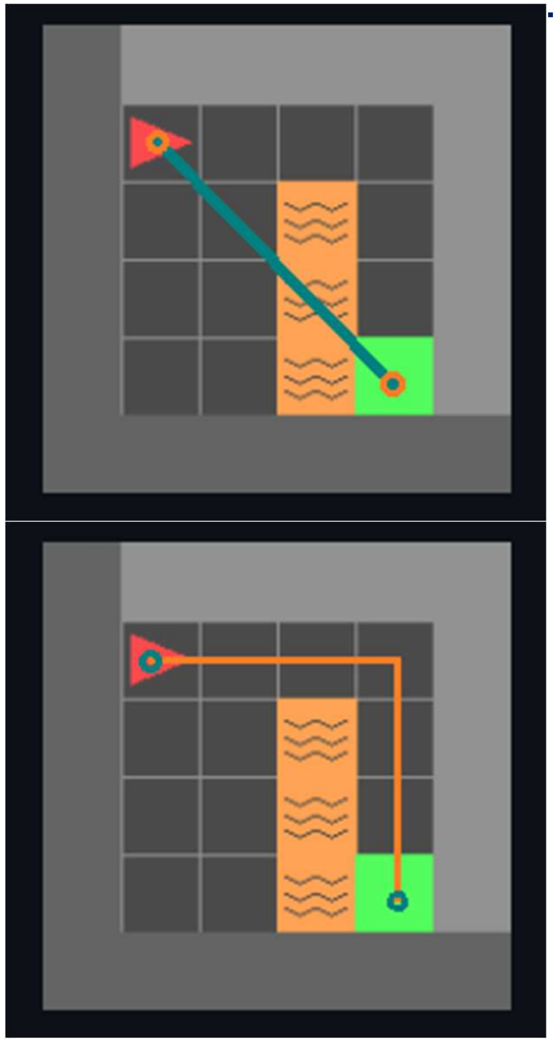
PQ = {(START,0)}

Heuristic Search & Hill Climbing

- DFS and BFS are both searching blind " they search all possibilities in an order dictated by NextStates(S_i)
  - Its still uninformed search
- When people search, we look in the most promising places first
  - Solving maze with logical rules
  - Rule of thumb (looking for lost items)
- e.g. in the block world at state {[a],[b],[c]}, the most promising move is to {[a],[b,c]}
  - Obvious to (nearly) any human, right?
- The most promising states are often those which are closest to the goal state
- But how can we know how close we are to the goal state?

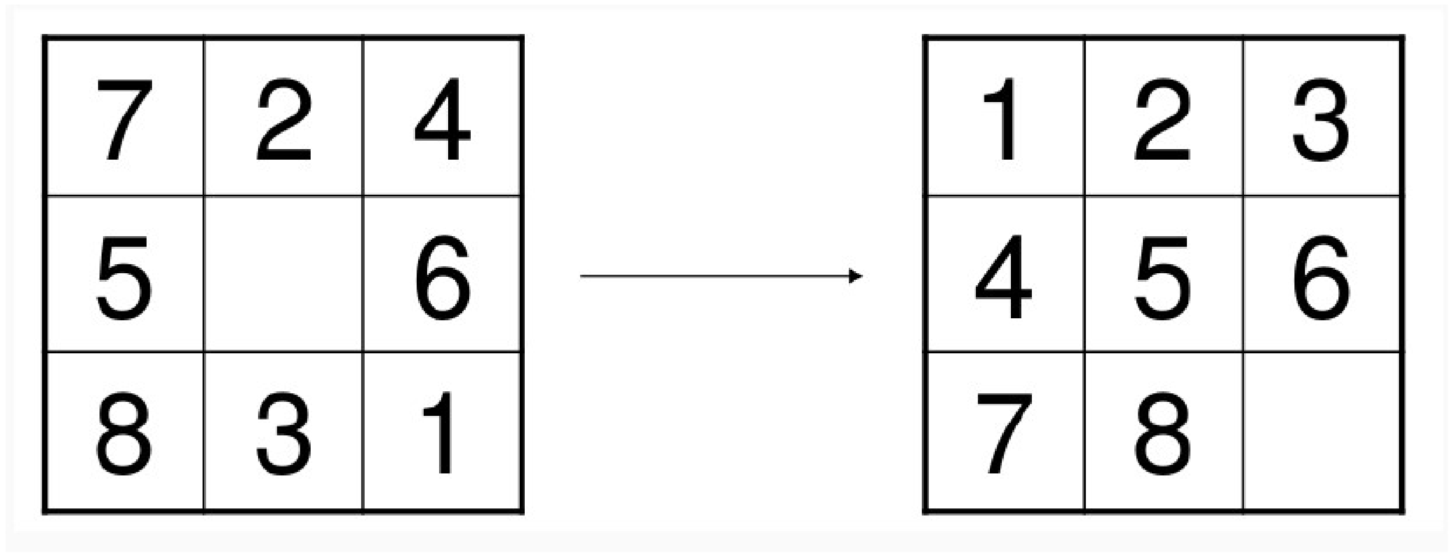
Heuristic Functions

- We had 2 possible metrics
- Euclidean Distance and Manhattan Distance
- Slightly different values
- Both ignore the magma, they assume that there are no constraints on actor movement.
- They relax the problem
- This means that we consider a much simpler and potentially easier to calculate case



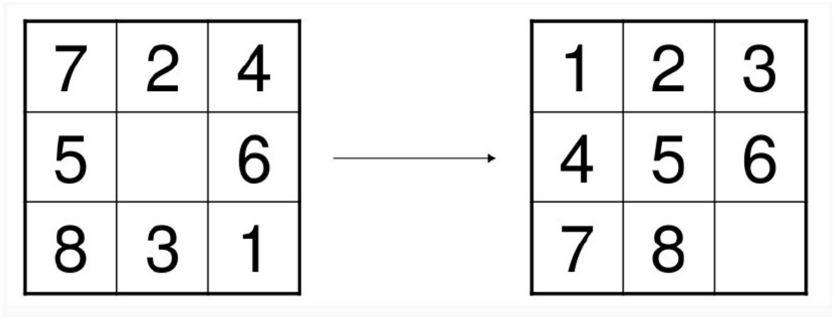
Heuristic Functions

- Consider the sliding blocks puzzle
- We can move 1 block at a time into the blank space
- Can we design a good heuristic function?



Consider a Relaxed Function

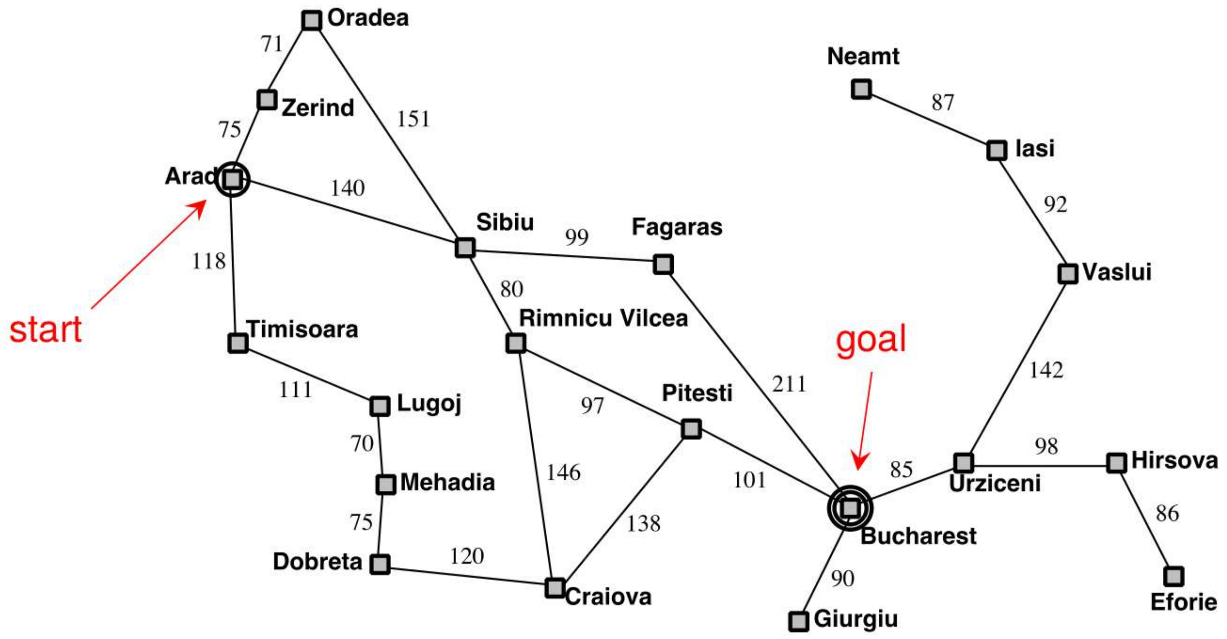
- When we relax the problem, we remove some of the restrictions
- Possible examples
  - $h_1(s)$ = number of misplaced tiles on board
  - $h_2(s)$ = sum of all Manhattan distances of putting tiles in correct position
  - $h_{2(s)} = \max(h_1(s), h_2(s))$
    - We can combine admissible heuristics
- We can ignore the rule about having to slide into a blank space
- We relax the problem



Heuristic Search

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.



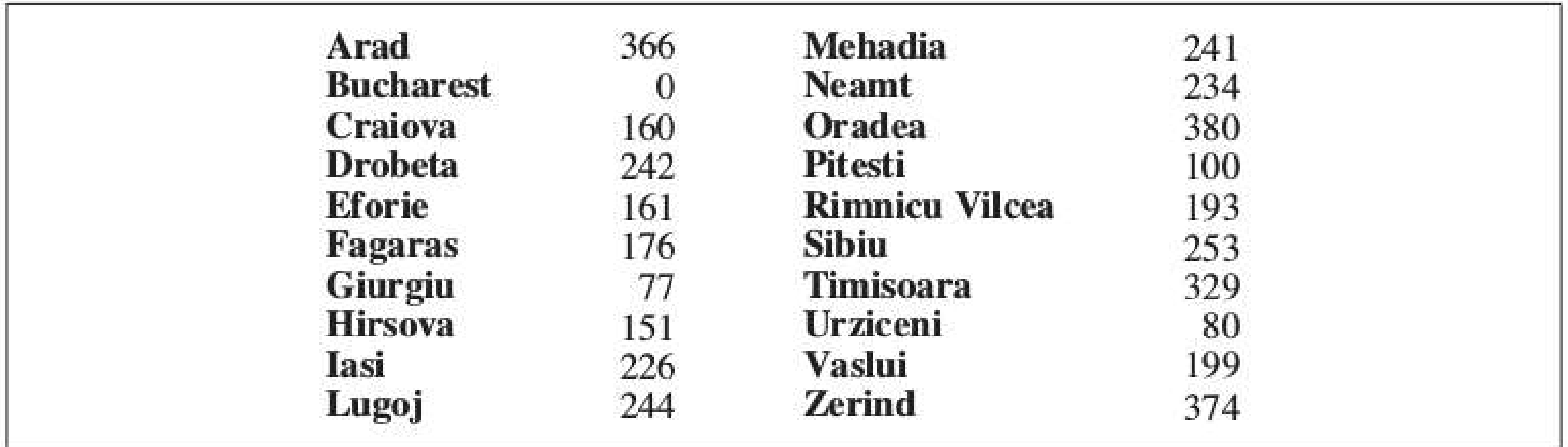
Hill Climbing

- The easiest way to use a heuristic estimate to search is to require that every single move we make takes us closer to the goal
- The form of search doesn't even require an agenda, since at each decision point, we take the action that looks best to us and repeat until we are done
- Very low memory footprint, as no agenda or visited list needed
- Problems: dead ends, plateaus, solution quality (i.e. the number of steps can be very poor)
- Used to good effect in planning software

Hill Climbing

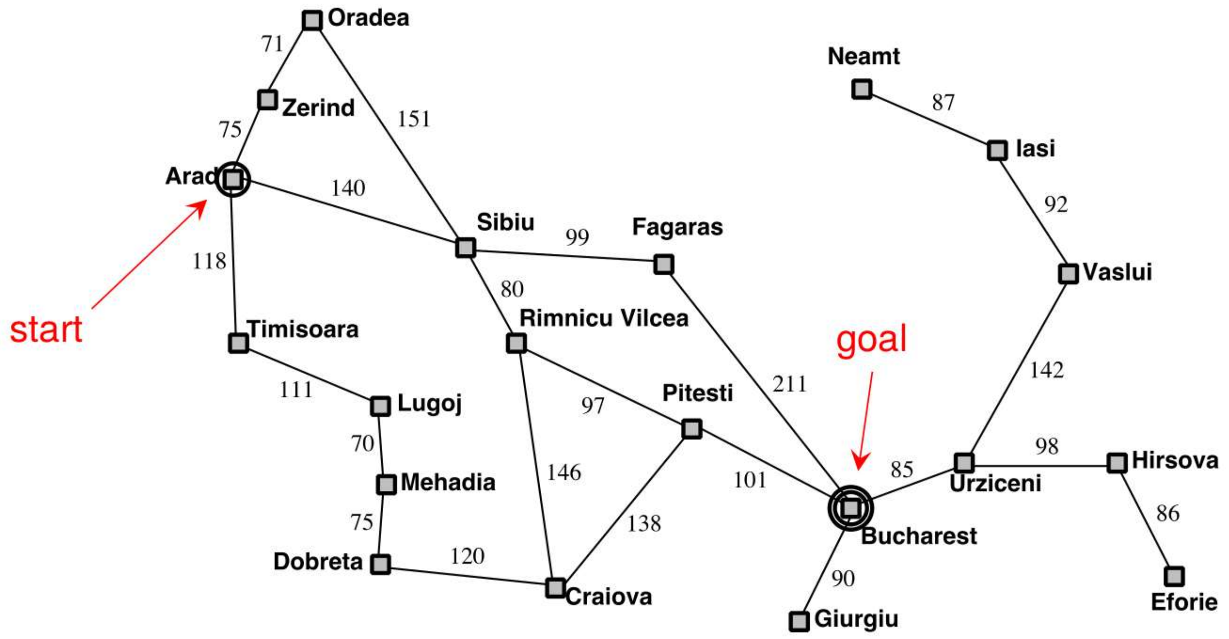
```
HCS (Graph, state):
        current ← state
        while true do
                if goal(current) then
                        return current
                E = next_states(current) in Graph
                current ← element s’ of E with min h(s’)
```

- Problem: Navigate from Iasi to Oradea
- Option, Neamt or Vaslui
- Heuristic says Neamt is closer!
- But then, dead end, so go back. Use lowest cost
- That is also Neamt! (distance 87)



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.



Enforced Hill Climbing

- Problems: dead ends, plateaus, solution quality (i.e. the number of steps can be very poor)
- Hill climbing can get stuck
- Enforced hill climbing uses Breadth first search
  - goal condition that h(s’) < h(s)
- Doesn’t look for the minimum, but used breadth first search to identify a better successor
- Can identify a better route than just hill climb (may avoid plateaus)

EnforcedHill Climbing

```
HCS (Graph, state):
        current ← state
        heuristic_val ←h(state)
        while true do
                if goal(current) then
                        return current
                current ← BreadthFirstSearch(Graph, state)
```

- Does not look for the minimum h, looks for the first h better than the heuristic_val
- Can depend how you order the search

Best First Search

Greedy Best First Search

- Enforced hill climbing is great when it works, but for some problems its better to keep track of the nodes we havent yet expanded, using the agenda
    - Keep our agenda and visited list
- We can then use the heuristic function to determine which node to expand next
- As new states are discovered, we add them to the agenda and record the value of the heuristic function
- When we pick the next node to explore, we choose the one which has the lowest value for the heuristic function (i.e. the one that looks nearest to the goal)
- We do this by picking the best heuristic value from the agenda
    - We do not consider actual values, purely the heuristic
- Very similar to Uniform Cost Search, but using no cost information

```
● let Agenda = [($S_0$,0)]
● let Visited  $= []
● while Agenda ≠ = [ ]:
    ○ let (Current,i) = lowest-heuristic (Agenda)
    ○ let Agenda = remainder(Agenda)
    ○ if Goal (Current): return (''Found it!")
    ○ if Current not in Visited:
        ■ let Next = NextStatesWithCosts(Current)
        ■ let Agenda = Agenda + Next
        ■ Visited.append(Current)
```

Analysing Greedy Best-First Search

- The good:
  - Complete in finite search spaces – it will find a solution
  - Best case complexity of O(bm)
  - Easy to implement
  - With a good heuristic, very efficient
- The bad:
  - Very dependent on heuristic
  - Not complete in infinite space
  - Can be slow
- The ugly:
  - **Its greedy**
  - Its not always the best thing to try to get to the goal as quickly as possible

Best First Search and Algorithm A

- Best-first search can speed up the search by a very large factor, but can it isn't guaranteed to return the shortest solution
- When deciding to expand a node, we need to take account of how long the path is so far, and add that on to the heuristic value:
- $f(S_i, G) = g(S_0, Si) + h(Si, G)$
- Here, g represents the actual cost value (i.e. the distance covered from the initial state to the current location, and h is the heuristic value for the remaining distance
- This will give a search which has elements of both breadth-first search (uniform cost search) and best-first search
- This type of search is called “Algorithm A”

Algorithm A*

- If h(Si ,G) never over-estimates the distance from S_i to the goal G, it is called an admissible heuristic
- If h(Si ;G) is admissible, then Algorithm A will always return the shortest path (like breadth-first search) but will omit much of the work if the heuristic function is informative
- The use of an admissible heuristic turns Algorithm A into Algorithm A^*
- Uses: problem solving, route finding, path planning in robotics, computer games, etc.
- Complete as long as heuristic is safe.
- Optimal as long as the heuristic is admissible.

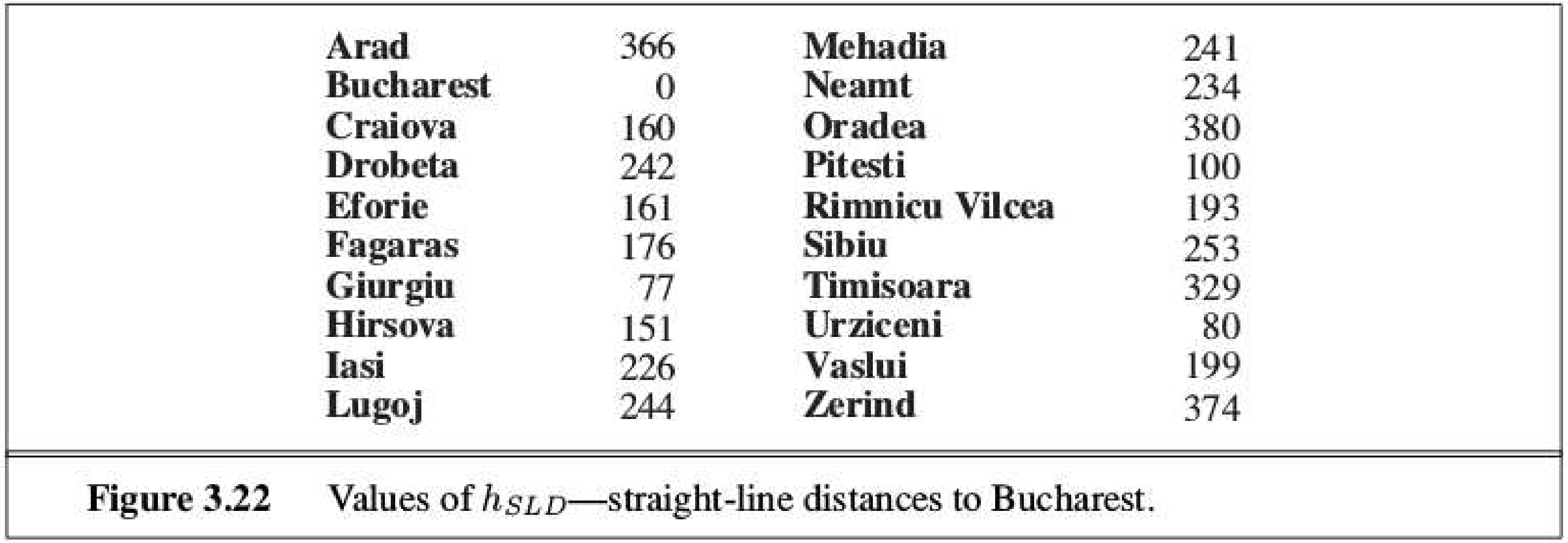
Why is A* Optimal?

- Suppose a suboptimal goal node, S_k, appears in the agenda -we have not selected it yet, so we do not yet know that it is a goal node
- Also on the agenda, there must be a node, S_i which is on the optimal path from S_0 to the goal state
- Since the heuristic function, h(Si , G), is admissible, this means:
  - $g(S_0; S_k) > g(S_0; S_i) + h(S_i; G)$
  - (as the right hand side will be smaller or equal to the actual costs of reaching the goal G)
- Therefore S_k will never be selected over Si for expansion

A* And visited List

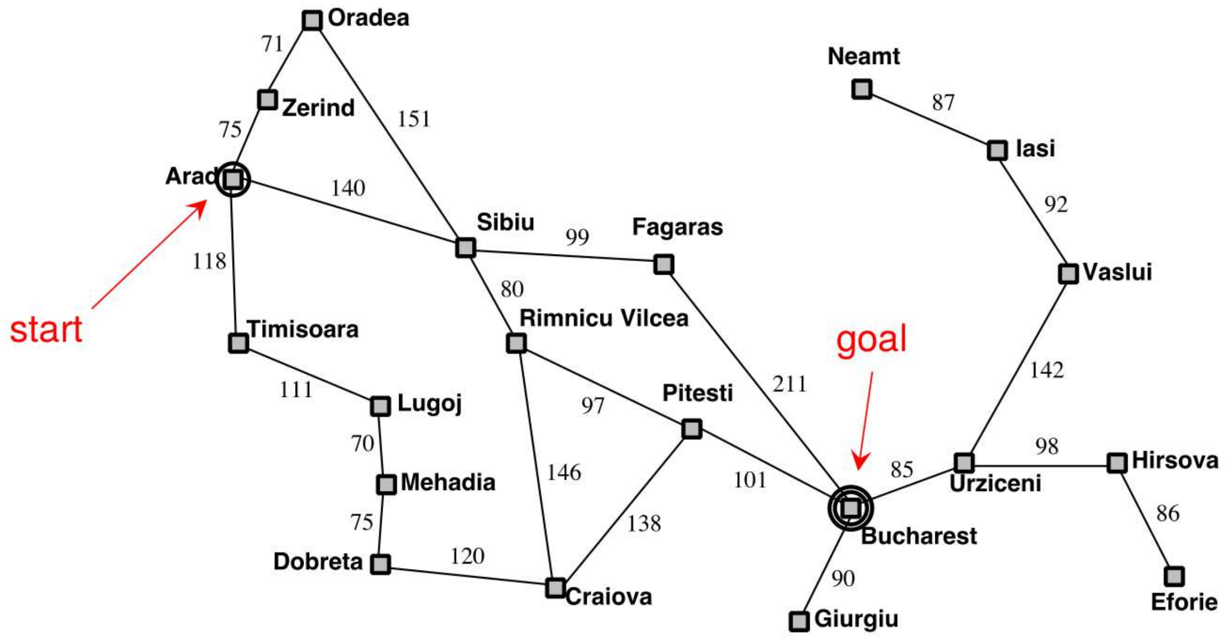
- To optimise the algorithm one can maintain a visited list (similar to UCS)
- in this case you need to record the expanded state together with the value at which it was expanded
- if we meet a node that has been expanded already at a higher cost value, we expand again!

- A* Can use the best of both worlds



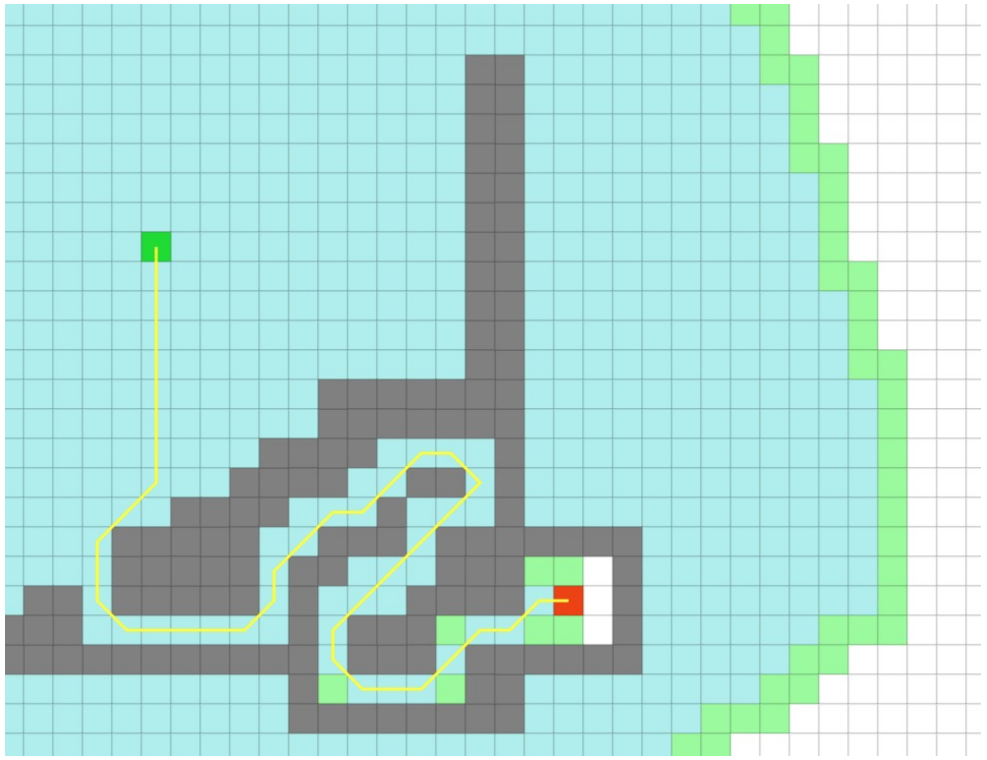
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.



Search Algorithm Examples

- http://qiao.github.io/PathFinding.js/visual/
- You can test different Heuristics and Algorithms



Summary

- Try these algorithms out!
  - Not just for maps, but for states too
- Greedy Best First Search
- A* Algorithm