CS5100: Foundations of Artificial Intelligence

Informed search algorithms

Dr. Rutu Mulkar-Mehta

Lecture 2

Administrative

- Assignment P0 – How many did it?
- Project 1- How many started it?

Review from Lecture 1

- Agents and Agent types
- Agents to perform search
- Uninformed Search Techniques
  - Breadth First Search
  - Depth First Search
  - Uniform Cost Search
  - Iterative Deepening Search

Terminology

- Start and Goal
- Parent Node
- Child Node
- Fringe or Frontier
  - Next list of nodes to be expanded
- Search Strategy
  - BFS, DFS, UCS etc.

Space Time Complexity

- Time: Number of nodes generated during the search
- Space: Maximum number of nodes stored in memory

Review: BFS

- Starts with the root node and explores all neighboring nodes
  - Repeats this for every one of those
- This is realized in a FIFO queue
- It does an exhaustive search until it finds a goal.

Review: BFS

- BFS is complete
  - (i.e. it finds the goal if one exists and the branching factor is finite).
- It is optimal (Finds best solution)
  - (i.e. if it finds the node, it will be the shallowest in the search tree).

Review: BFS

- Space: $O(b^d)$
- Time: $O(b^d)$

- b = branching factor
- d = depth to which BFS is reached

Review: DFS

- Explores one path to the deepest level and then backtracks until it finds a goal state.
- This is realized in a LIFO queue (i.e. stack).

Review: DFS

- DFS is complete
  - (if the search tree is finite).
- It is not optimal
  - (it stops at the first goal state it finds, no matter if there is another goal state that is shallower than that).

Review: DFS

- Space: O(bm)
  - Much lower than BFS
- Time: $O(b^m)$
  - (Higher than BFS if there is a solution on a level smaller than the maximum depth of the tree).
  - Danger of running out of memory or running indefinitely for infinite trees.

- b = branching factor
- m = maximum depth

Review: Iterative Deepening DFS

- The search depth for DFS increased iteratively over time, gradually increasing the maximum depth to which it is applied.
- This is repeated until finding a goal.
- Combines advantages of DFS and BFS.
- It is complete.
- It is optimal (the shallowest goal state will be found first, since the level is increased by one every iteration).

Review: Iterative Deepening DFS

- Space: O(bd)
  - (better than DFS, d is depth of shallowest goal state, instead of m, maximum depth of the whole tree).
- Time: $O(b^d)$

- b = branching factor
- m = maximum depth

Review: Uniform Cost Search

- Expand lowest path cost
- Demands the use of a priority queue

Uniform Cost Search

- Space: $O(b^{1+C/e})$
- Time: $O(b^{1+C/e})$

→ Can be much greater than b^d

- C = cost of optimal solution
- e = cost of every action
- b = branching factor
- m = maximum depth

Today

- Informed Search
  - Problem Specific Knowledge is provided
  - This is beyond the definition of the problem itself
  - This information can be used to find solutions more efficiently than uninformed search

Today

- Best-first search
- Greedy best-first search
- A^* search

Best-first search

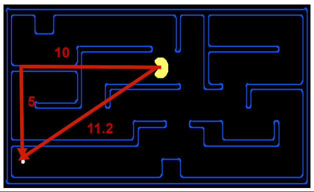
- Implementation:
  Order the nodes in fringe in decreasing order of desirability

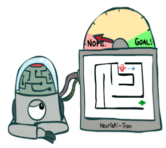
- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
  - f(n) is based on the heuristic h(n)

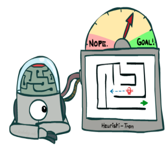
- Special cases:
  - greedy best-first search
  - A^* search

Search Heuristics

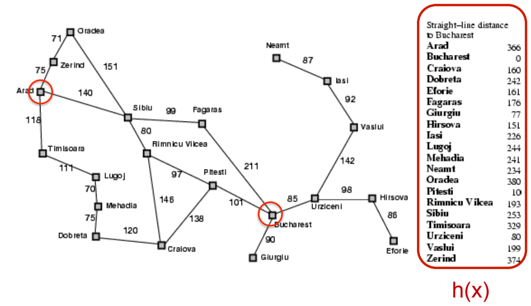
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing







Romania with step costs in km

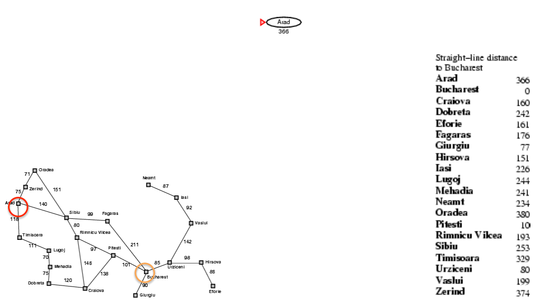


Greedy Search

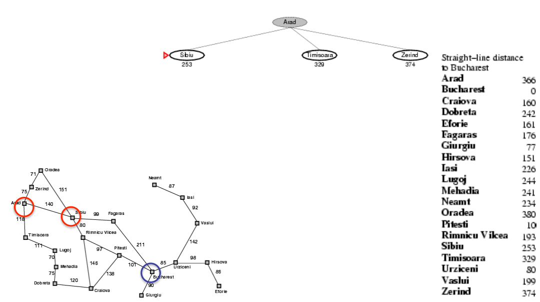
Greedy best-first search

- Evaluation function *f(n) = h(n)* (heuristic)
  - e.g. estimate of cost from *n* to *goal*
- e.g., $h_{SLD}(n)$
  - straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

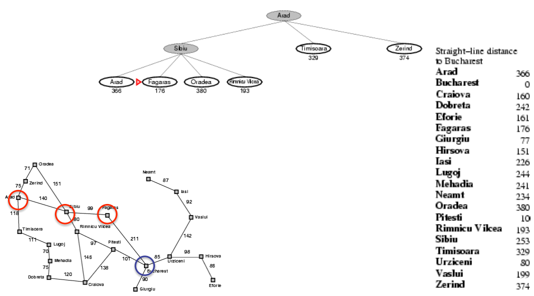
Greedy best-first search example



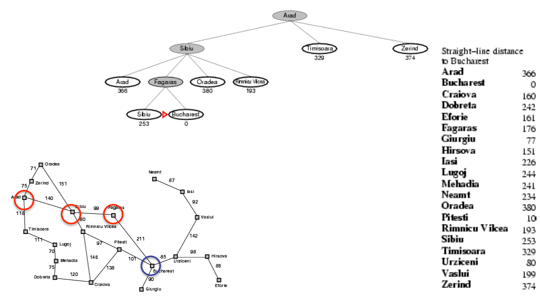
Greedy best-first search example



Greedy best-first search example



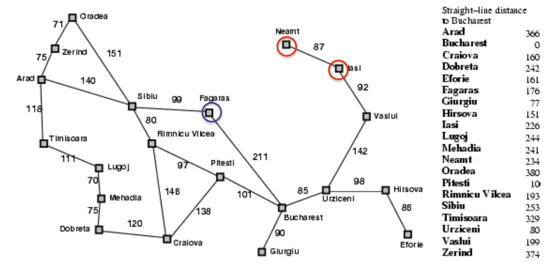
Greedy best-first search example



Properties of greedy best-first search

- Complete? No
  - can get stuck in loops,
  - e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$
  - but a good heuristic can give dramatic improvement
- Space? $O(b^m)$
  - keeps all nodes in memory
- Optimal? No

Best First: Incomplete



A* Search

A* search

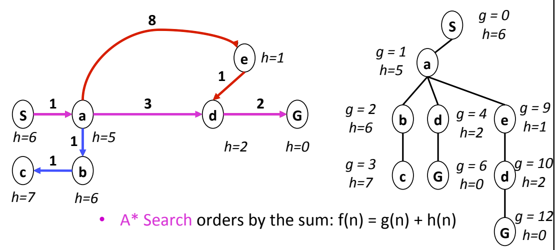
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

A* search and UCS

- A* search is identical to UCS, except it uses *g(n) + h(n)* instead of *h(n)*

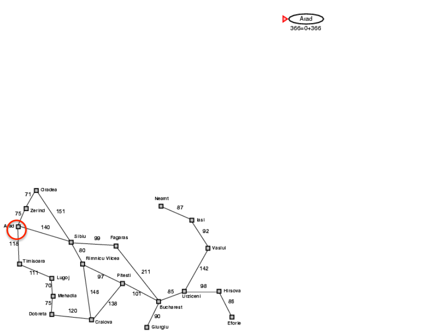
Combining UCS and Greedy

- Uniform-cost orders by path cost g(n)
- Greedy orders by goal proximity h(n)



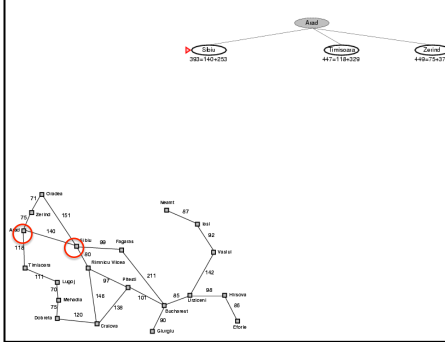
- A* Search orders by the sum: f(n) = g(n) + h(n)

A* search example



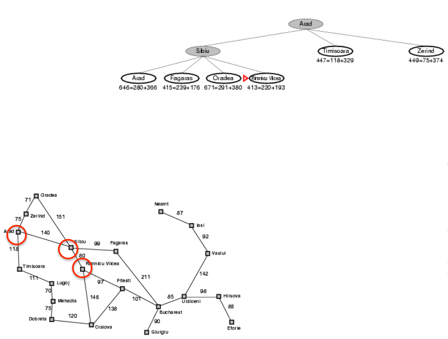
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example



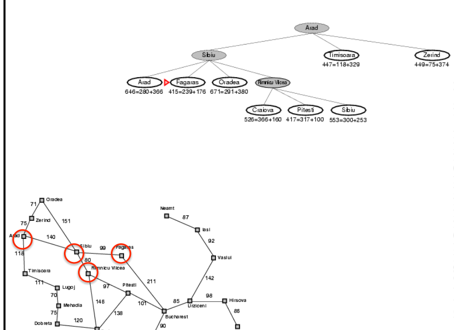
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example



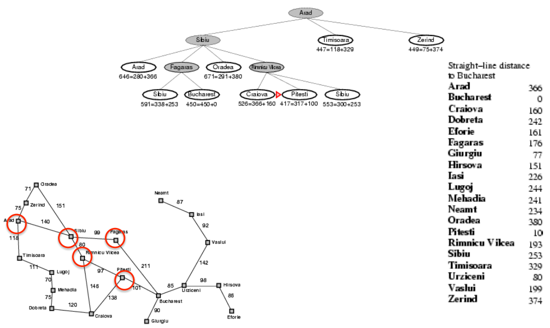
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

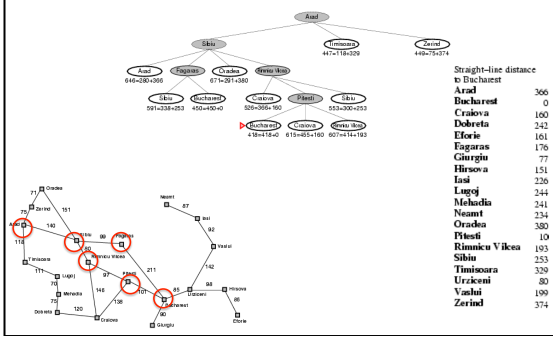


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

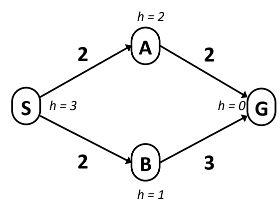


A* search example



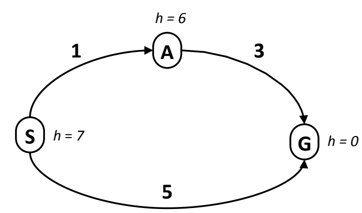
When should A* terminate?

- Should we stop when we enqueue a goal?



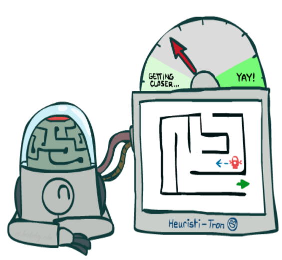
- No: only stop when we dequeue a goal

Is A* Optimal?

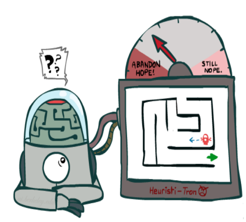


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

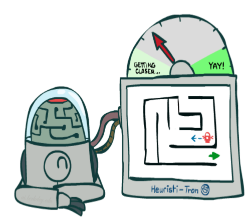
Admissible Heuristics



Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

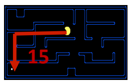
Admissible Heuristics

- A heuristic *h* is *admissible* (optimistic) if:

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

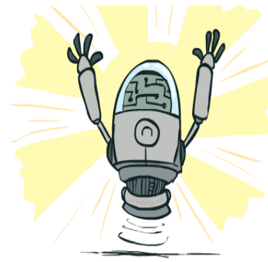
where $h^*(n)$ is the true cost to a nearest goal

- Examples:



- Coming up with admissible heuristics is most of what's involved in using A* in practice.

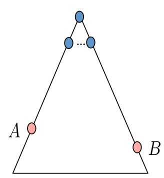
Optimality of A* Tree Search



Optimality of A* Tree Search

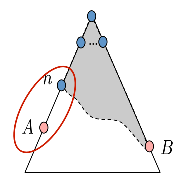
Assume:
- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

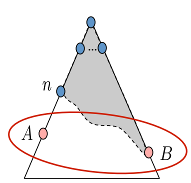
Proof:
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)



$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

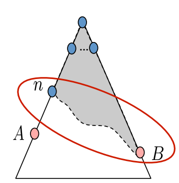
Proof:
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)



$g(A) < g(B)$	B is suboptimal
$f(A) < f(B)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

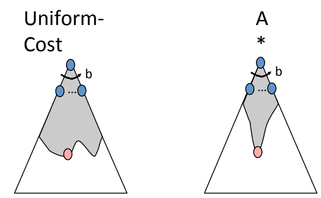
Proof:
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



$$f(n) \leq f(A) < f(B)$$

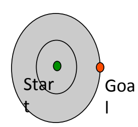
Properties of A*

Properties of A*

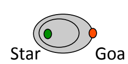


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Video of Demo Contours (Empty) -- UCS

Video of Demo Contours (Empty) -- Greedy

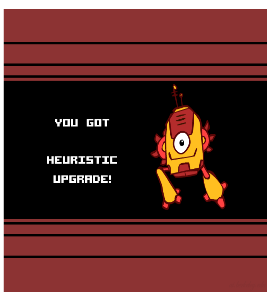
Video of Demo Contours (Empty) – A*

Video of Demo Contours (Pacman Small Maze) – A*

Comparison

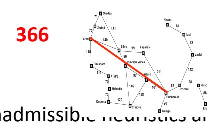
Greedy   Uniform Cost   A*

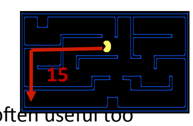
Creating Heuristics



Creating Admissible Heuristics

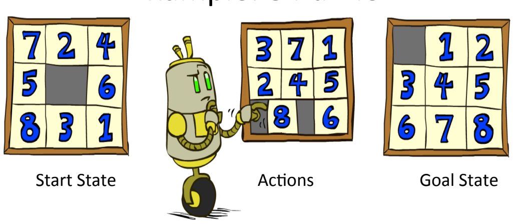
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available





- Inadmissible heuristics are often useful too

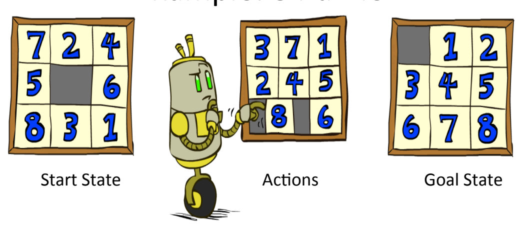
Example: 8 Puzzle



Start State   Actions   Goal State

- What are the states?
- How many successors from the start state?

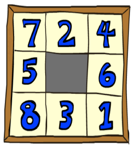
Example: 8 Puzzle



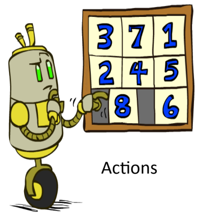
Start State   Actions   Goal State

- What is the branching factor?
- If the average solution ~22 steps, what is the max depth that you would like to hit in a search tree?

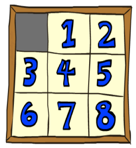
Example: 8 Puzzle



Start State



Actions



Goal State

- How many states?
  - 9!/2
  - half the states are impossible to solve

Relaxed problems

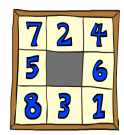
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

8 Puzzle - heuristics

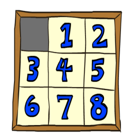
- *$h_1(n)$: The number of Misplaced Tiles*
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- *h2(n): sum of distances of the tiles from their goal position*
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

8 Puzzle I

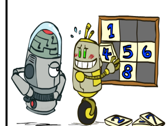
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



Goal State

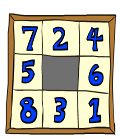


	Average nodes expended when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

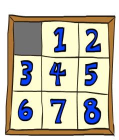
Statistics from Andrew Moore

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18



Start State



Goal State

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?
- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

A*: Summary

A*: Summary

- A* uses both path costs and heuristics
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems