CS 482/628 AI: Intelligent Search

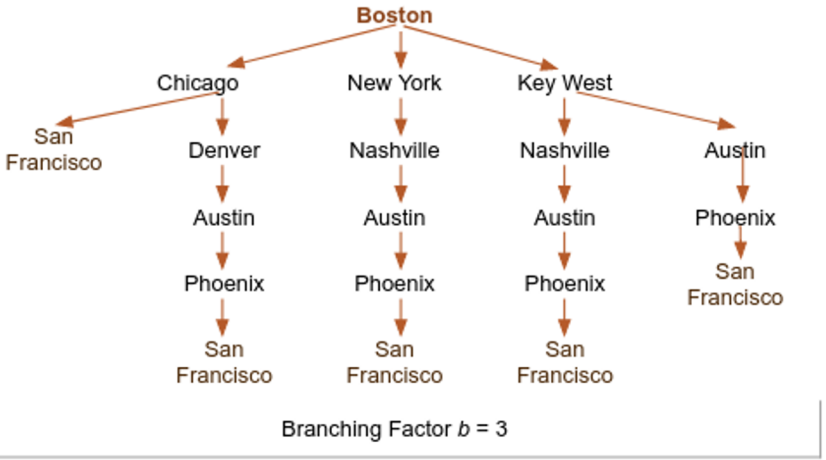
Fall 2021

previously on...

How to Search:

Generating Sequences and Data Structures

Depth
0
1
2
3
4
5



Measuring Performance

Completeness: is the strategy guaranteed to find a solution when one exists?

Time Complexity: how long does it take to find a solution?

Space Complexity: how much memory does it require to perform the search?

Optimality: Does the strategy find the best-quality solution when more than one solution exists?

Types of Uninformed Search

- Breadth-First Search: expands the shallowest nodes first
  - Complete, optimal for unit cost, exponential space complexity
- Depth-First Search: expands the deepest unexpanded node first
  - Neither complete nor optimal, but linear space complexity
- Depth Limited Search: is similar to DFS except the depth is limited
- Iterative Deepening Search: calls DFS with increasing depth limits
  - Complete, optimal for unit cost, linear space complexity
- Bidirectional Search: expands two frontiers, one around the initial state and one around the goal; stops when the two frontiers meet

More methods

Outline

- Best-first search
- A* search
- Heuristics
- Hill-climbing
- Simulated annealing

Informed (Heuristic) Search

- Uses domain-specific hints about the location of goals
- Can find solutions more efficiently than an uninformed strategy
- The hints come in the form of a heuristic function h(n)
- h(n) = estimated cost of the cheapest path from the state at node n to a goal state
    - Route-finding problem: we can estimate the distance from the current state to a goal as the straight-line distance on the map between the two points.
- A heuristic function is an inexact estimate of the evaluation function.

Review: General search

```
function GENERAL-SEARCH( problem, QUEUING-FN) returns a solution, or failure
    nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
    loop do
        if nodes is empty then return failure
        node ← REMOVE-FRONT(nodes)
        if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
        nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
    end
```

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

Best-first search

Idea: use an *evaluation function* for each node
– estimate of "desirability"

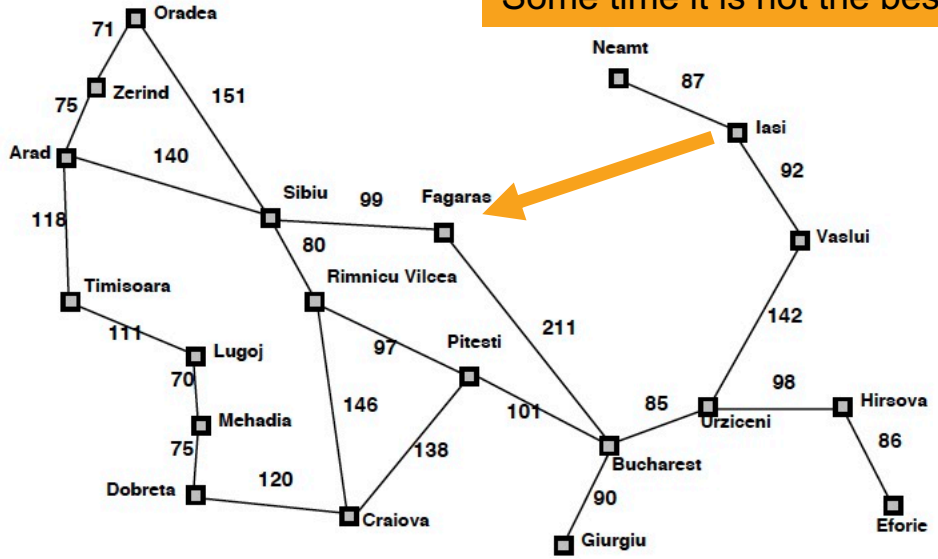
⇒ Expand most desirable unexpanded node

Implementation:
QUEUEINGFN = insert successors in decreasing order of desirability

Special cases:
greedy search
A* search

Romania with step costs in km

Some time it is not the best



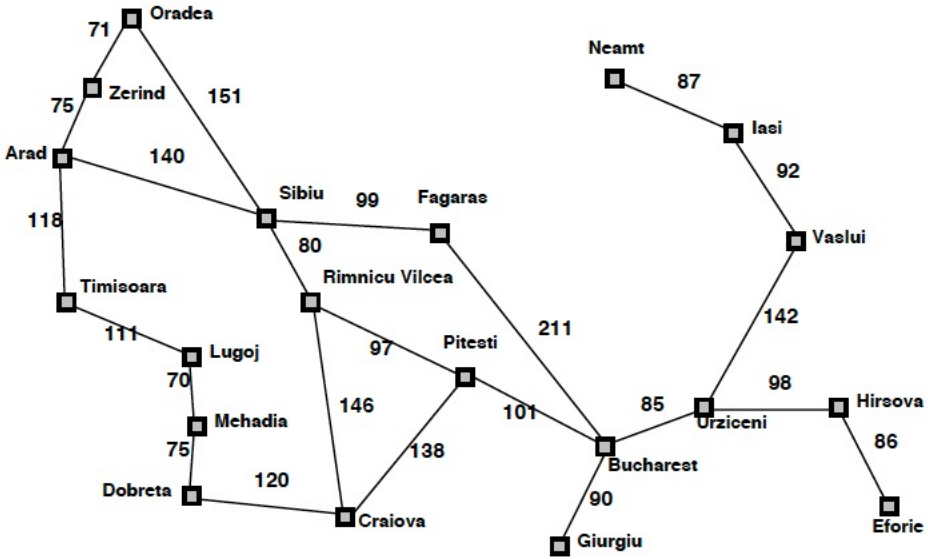
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

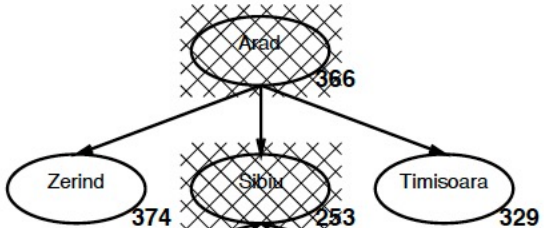
Evaluation function $h(n)$ (heuristic)
$=$ estimate of cost from n to $goal$

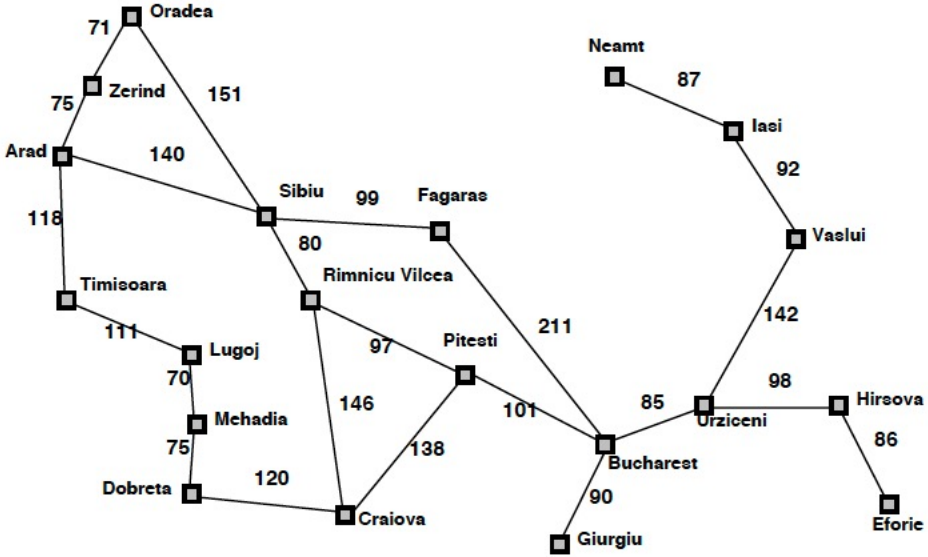
E.g., $h_{\mathrm{SLD}}(n) =$ straight-line distance from n to Bucharest

Greedy search expands the node that *appears* to be closest to goal

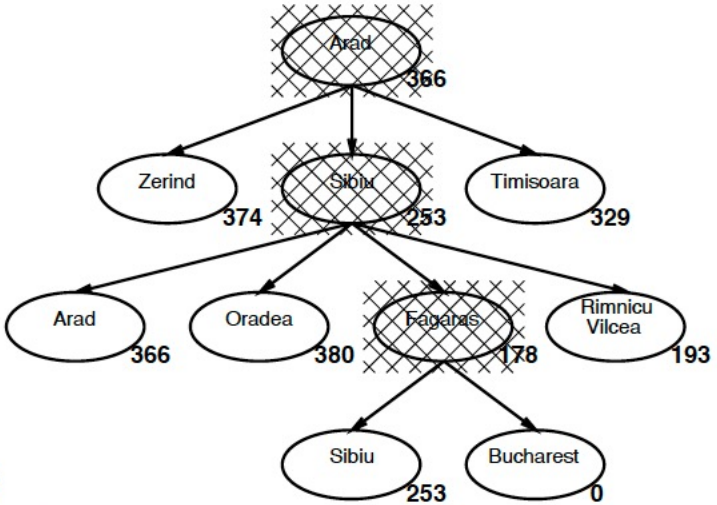


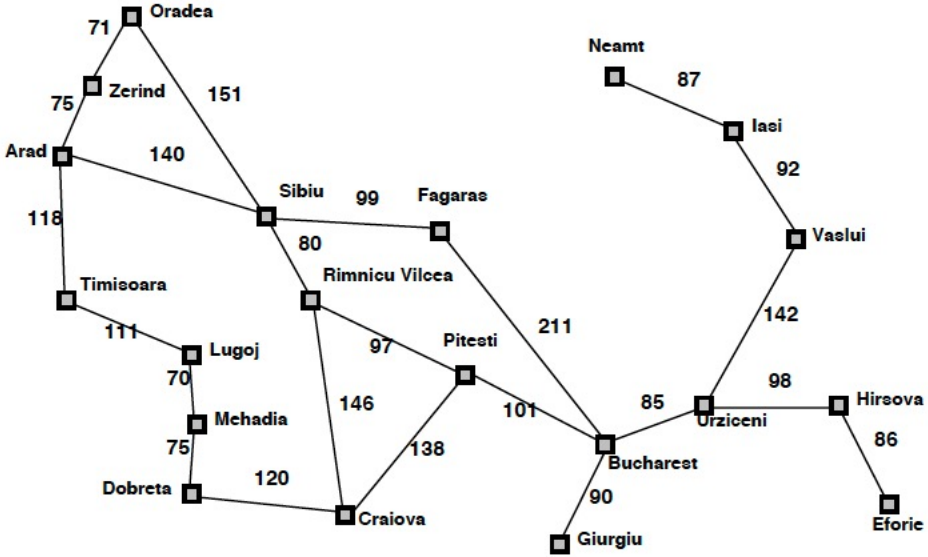
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





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





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

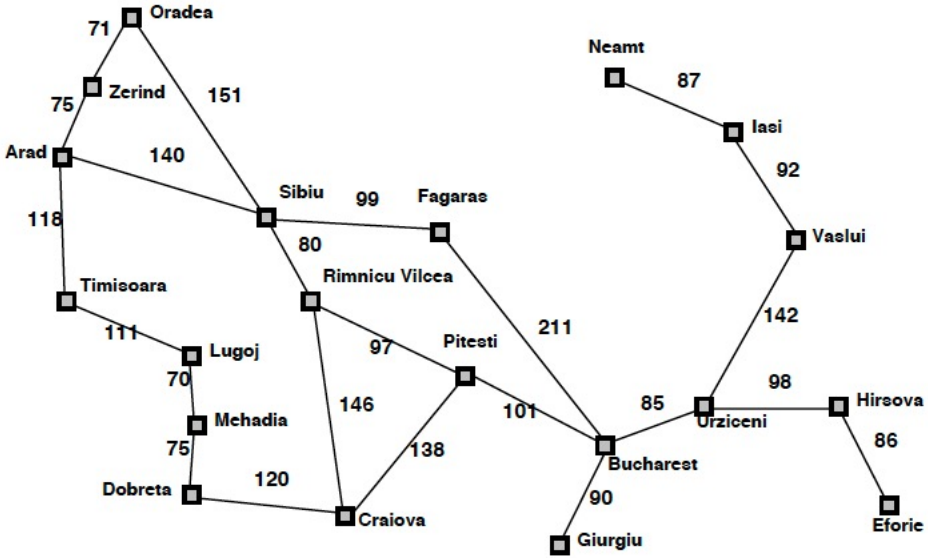
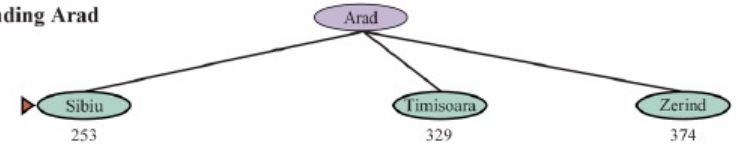


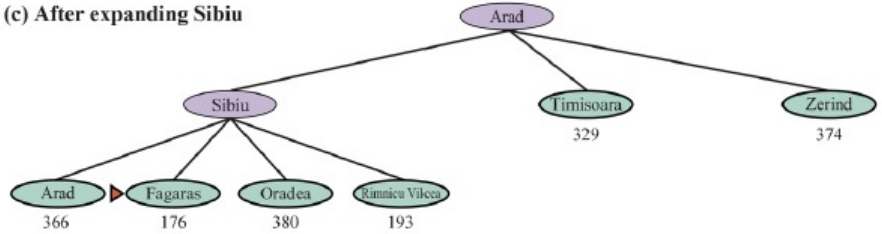
Figure 3.17

(a) The initial state

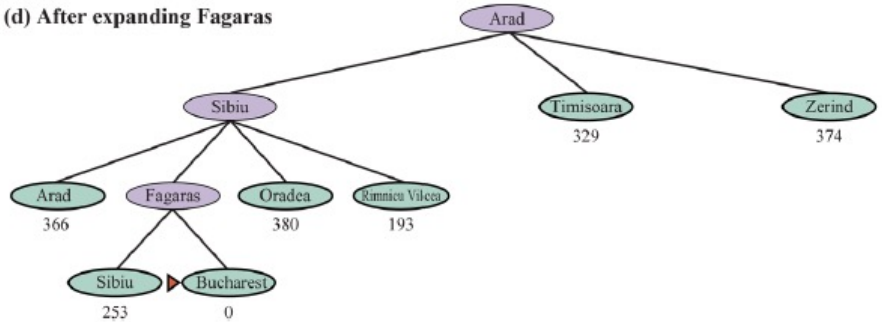
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Stages in a greedy best-first tree-like search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

Properties of greedy search

Complete??

Time??

Space??

Optimal??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

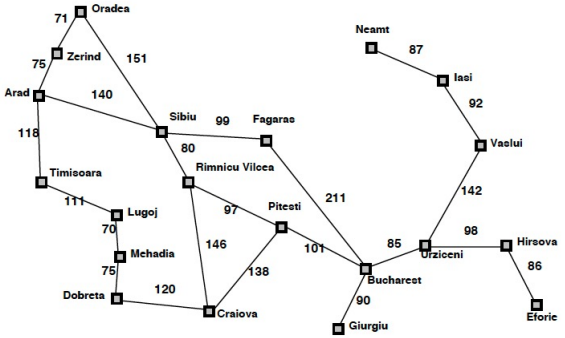
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an $admissible$ heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the $true$ cost from n.

E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

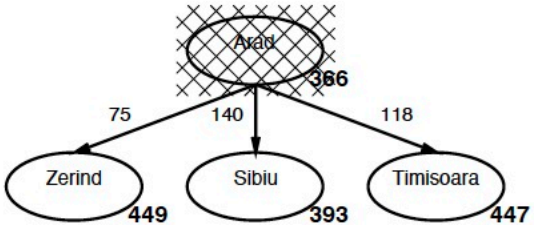
Theorem: A* search is optimal

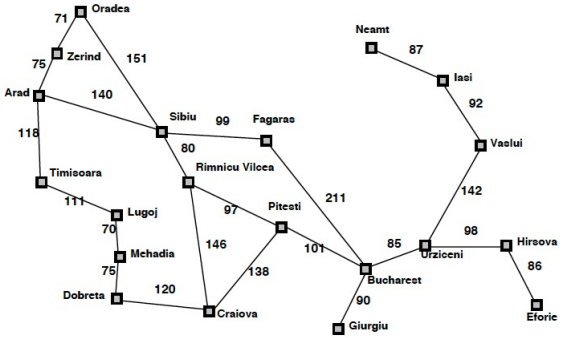
A* search example



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

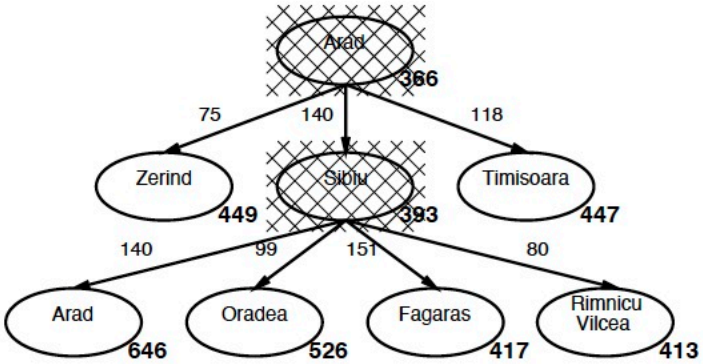
A* search example

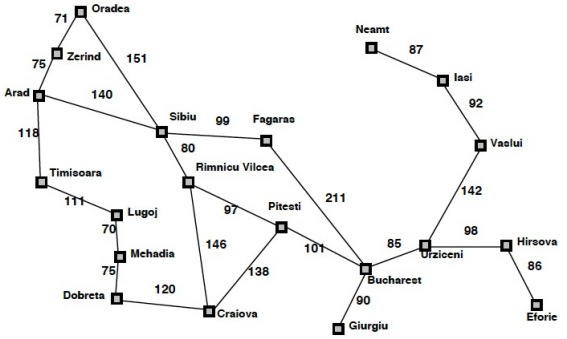




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

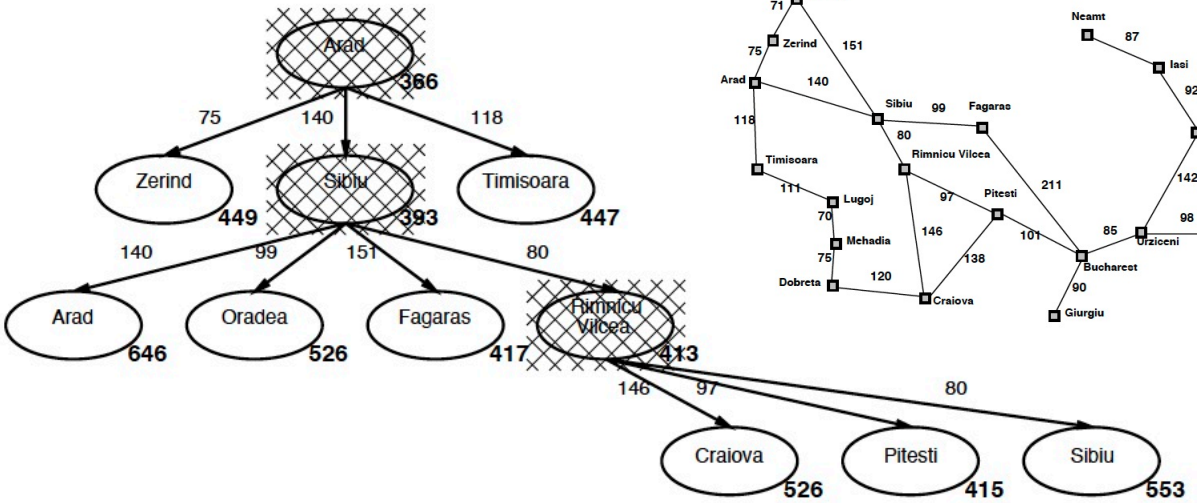
A* search example





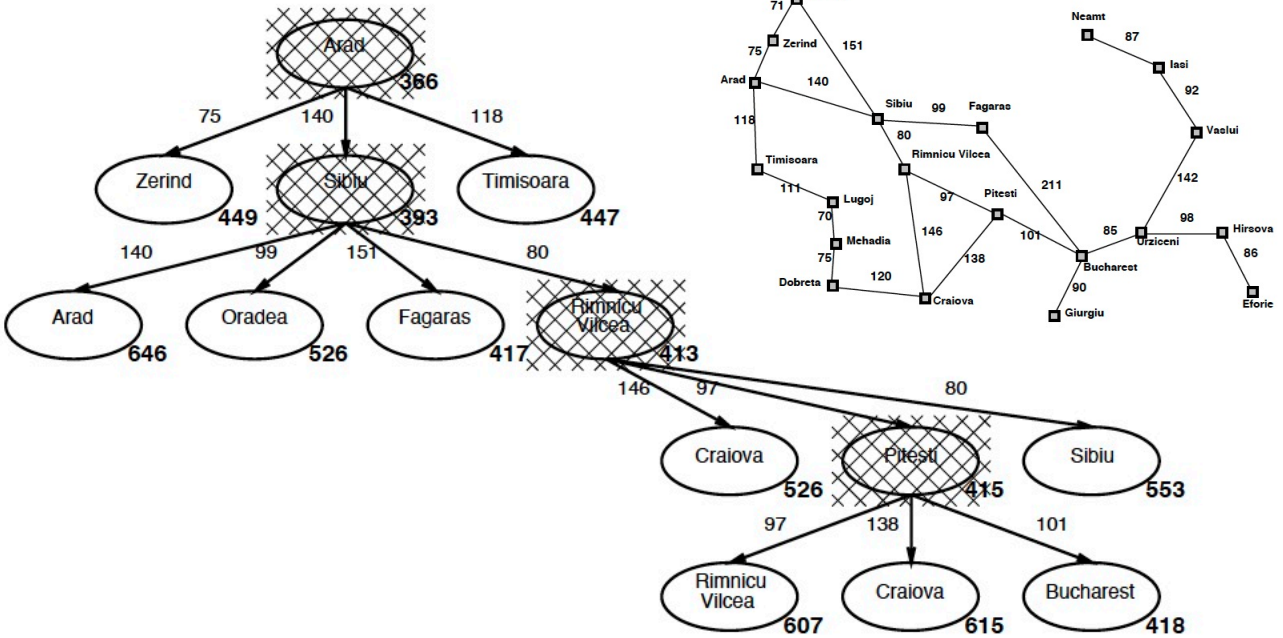
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example



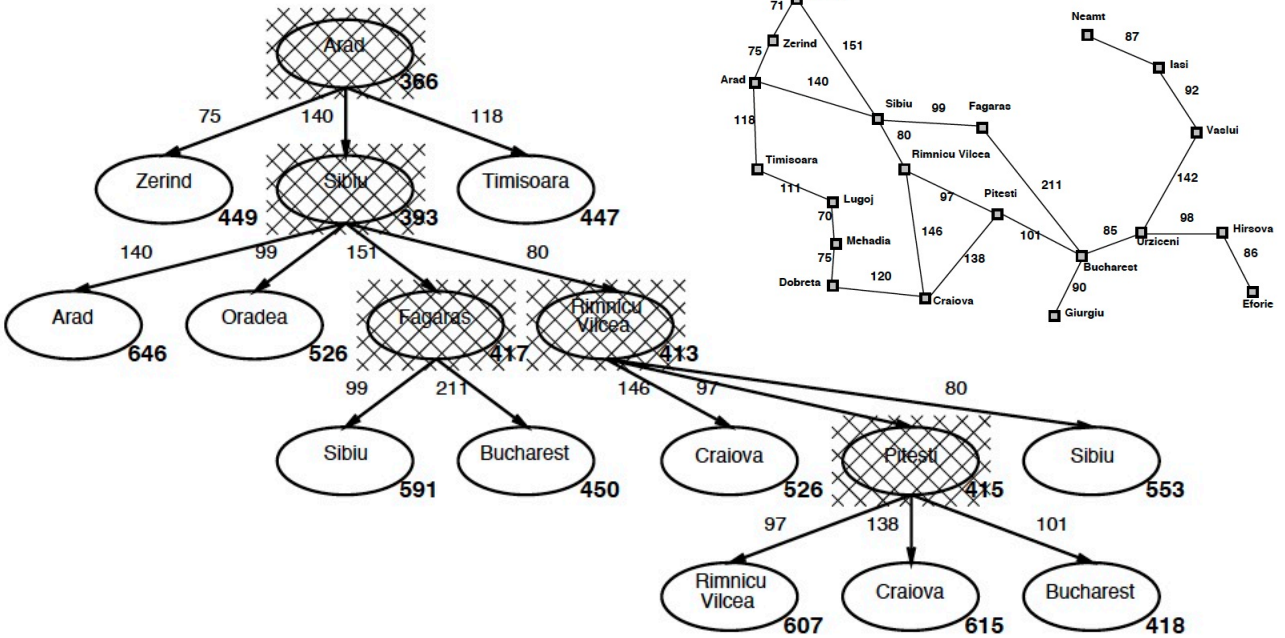
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

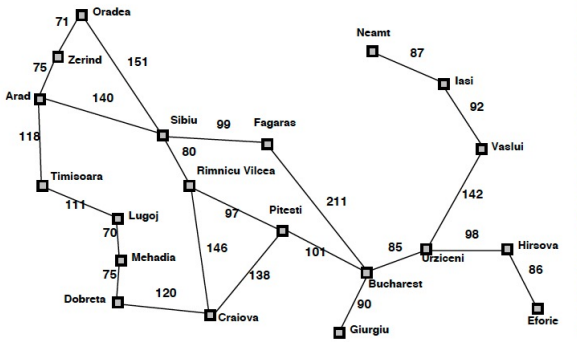


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

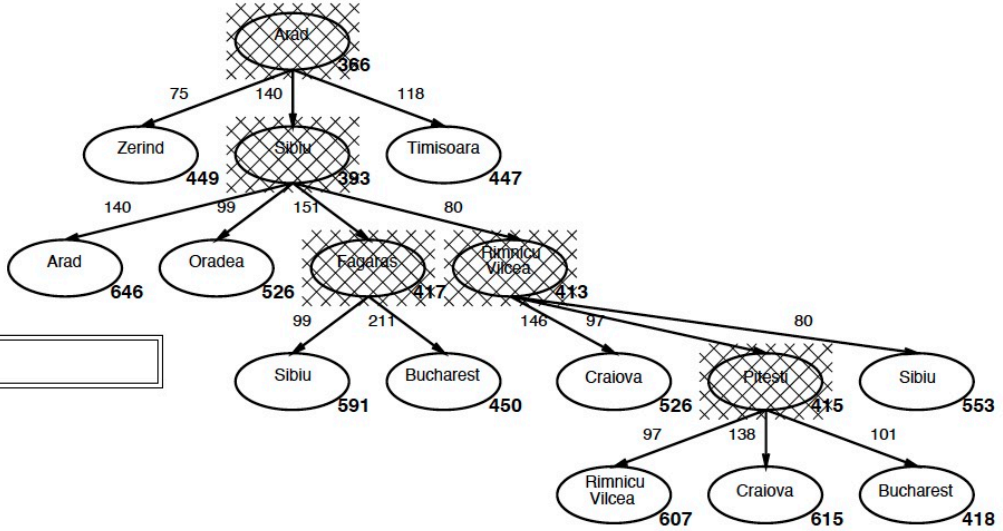


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

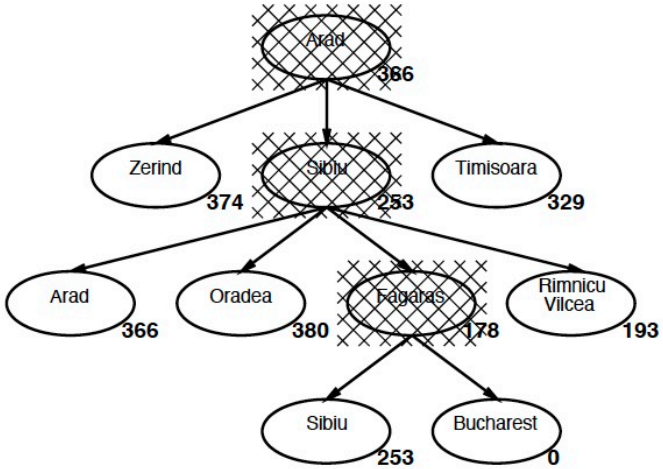


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example



Greedy search example



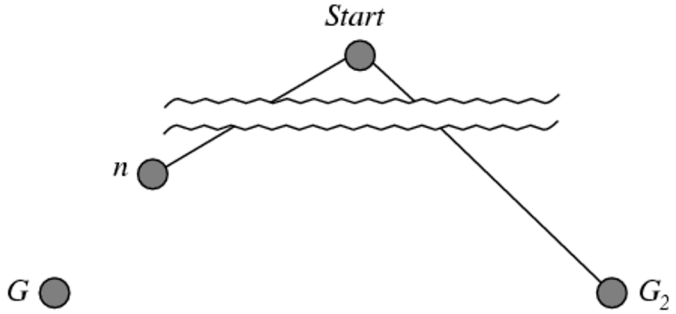
Optimality of A* (proof)

- Assume that G_2 has been generate and is in the queue
- Let n is an unexpanded node on a shortest path to an optimal goal G_1
- $f(G_2) = g(G_2)$ because $h(G_2) = 0$
- $f(G_2) = g(G_2) > g(G_1)$ since G_2 is suboptimal
- $g(G_1) = g(n) + h^*(n) \geq g(n) + h(n)$ since h is admissible
- Therefore, $g(G_1) \geq f(n)$
- $f(G_2) > g(G_1) \geq f(n)$ since h is admissible
- Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

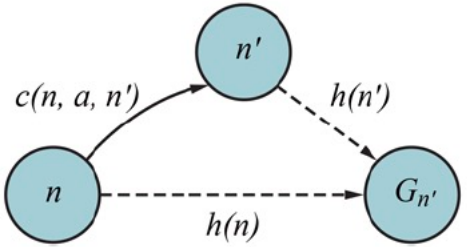
Evaluation function $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an *admissible* heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.

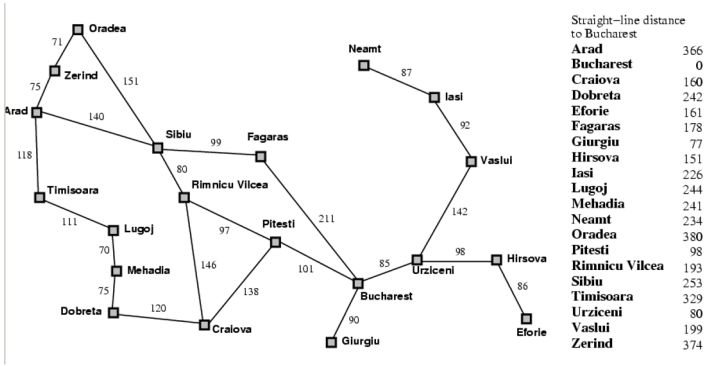


Consistency



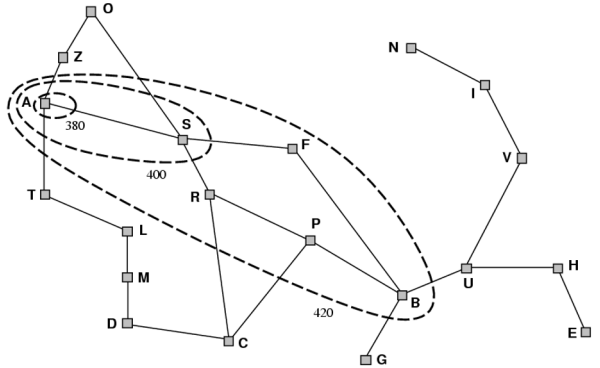
- A slightly stronger property is consistency
- h(n) is consistent if $h(n) \leq c(n,a,n') + h(n')$
- It is called “triangle inequality”
- Every consistent heuristic is admissible (but not vice versa)
- Therefore, with a consistent heuristic, A* is cost-optimal
- With a consistent heuristic, we never re-add a state
- Note that the straight-line distance above is consistent!
- Many implementations avoid inconsistent heuristics, but Felner et al. (2011) argues that the worst effects rarely happen in practice

How does A* Search Work?



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



- A useful way to visualize is to draw contours in the state space
- Inside the contour labeled 400, all nodes have $f(n) = g(n) + h(n) \leq 400$
- Because A* expands the frontier node of the lowest f-cost, A* search gradually adds “f-contours” of nodes.
- Contour I has all nodes with $f = f_i$ where $f_i < f_{i+1}$

Completeness of A*

- A* expands nodes in order of increasing *f*
- When would a solution not be found?
  - Node with an infinite branching factor
  - A path with a finite path cost but an infinite number of nodes
- A* is complete when
  - There is a finite branching factor
  - Every operator costs at least some positive ε

Complexity of A*

- Computation time is limited by the quality of the heuristic function (but is still exponential)
  - Issue #1 : Choosing the right heuristic function can have a large impact

- More serious problem is that all generated nodes need to be kept in memory
  - Issue #2 : Can we limit the memory requirements?

Properties of A*

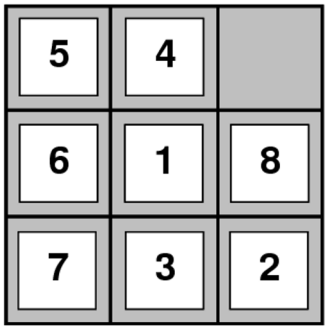
Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

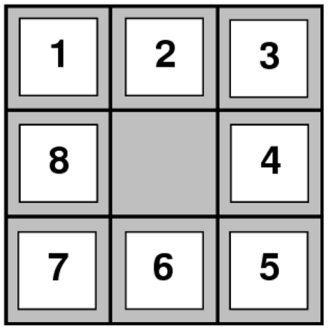
Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

Issue #1: Choosing a Heuristic Function



Start State



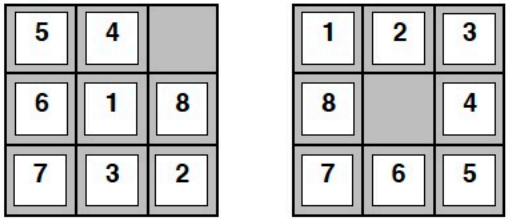
Goal State

- Must be admissible (never overestimate)
- Heuristics for the 8-Puzzle
  - *h1* = number of tiles in the wrong position
  - *h2* = sum of the distances of the tiles from their goal positions (city block distance)

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



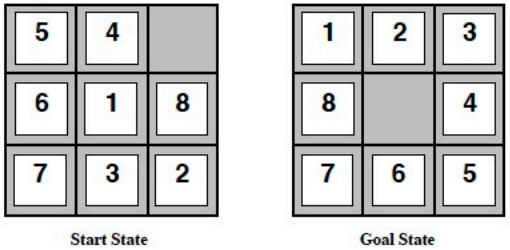
Start State Goal State

$h_1(S)$ =??
$h_2(S)$ =??

Admissible heuristics

E.g., for the 8-puzzle:

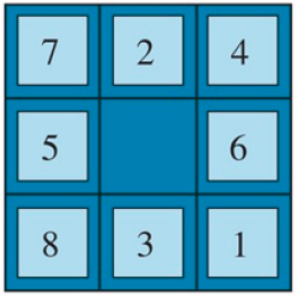
$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



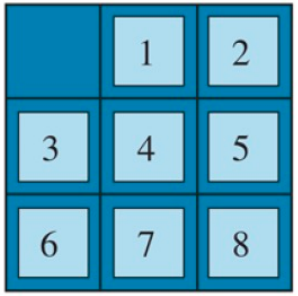
Start State Goal State

$h_1(S)$ =?? 7
$h_2(S)$ =?? 2+3+3+2+4+2+0+2 = 18

Dominance



Start State



Goal State

- How many reachable states in an 8-puzzle?
- 9!/2 = 181,400 states → easily stored in memory
- But what for 15-puzzle?
- Over 10 trillion → need a good admissible heuristic function
  - *h1* = number of tiles in the wrong position
  - *h2* = sum of the distances of the tiles from their goal positions (city block distance)
- $h_2(n) > h_1(n)$ for all n (both admissible) then h_2 dominates h_1 and is better for search

Effect of Heuristic Accuracy on Performance in the 8-puzzle

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

- Compare iterative-deepening with A* using h1 (# misplaced tiles) and h2 (city block distance)
- Effective branching factor b*
  - Number of expanded nodes = 1 + b* + (b*) 2 + … + (b*) depth
  - b* remains relatively constant across many measurements
- Always better to use a heuristic with higher values, so long as it does not overestimate

Issue #2: Limiting Memory Utilization

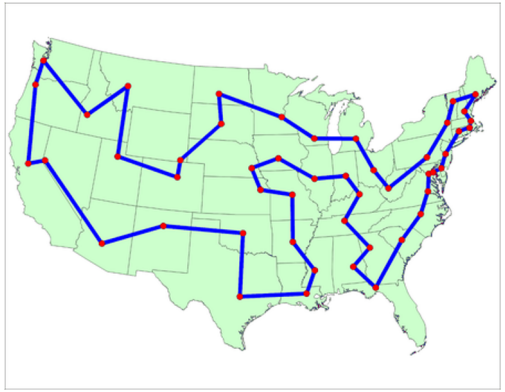
- If we can maintain a bound on the memory, we might be willing to wait for a solution

- Two techniques for Memory Bounded Search:
  - Iterative deepening A* (IDA*)
  - Recursive Best-First-Search (RBFS)

Iterative Deepening A* Search (IDA*)

- Each iteration is a depth-first search with a limit based on f rather than on depth
- Complete and optimal (with same caveats as A*)
- Requires space proportional to the longest path that it explores
- Can have competitive time complexity, since the overhead of maintaining the nodes in memory is greatly reduced

Problems with IDA*



- In the TSP, different heuristic function value for each state
- Each contour contains only one additional node
- If A* expands N nodes, the IDA* will expand

  $1+2+3+4+\ldots+N = O(N^2)$ nodes
- If N is too large for memory, N^2 is too long to wait
- Runs into problems because it recalculates every node

Recursive Best-First Search (RBFS)

- total path cost (f) =
  actual path so far (g) +
  heuristic estimate of future path to goal (h)

- uses f_limit variable to keep track of best *alternative* path available from any ancestor of the current node

Properties of Recursive Best-First Search (RBFS)

- RBFS will
  - be complete given sufficient memory to store the shallowest solution path
  - be optimal if the heuristic function is admissible (and you have enough memory to store the solution)

- Both RBFS and IDA* use not enough memory
  - Require at most linear space with the depth of the tree

Next Week

- Can you search without building a tree?
- What happens when you don't get to make the choice at each level of the search space?
- Game Playing
  - Minimax
  - Alpha-beta pruning