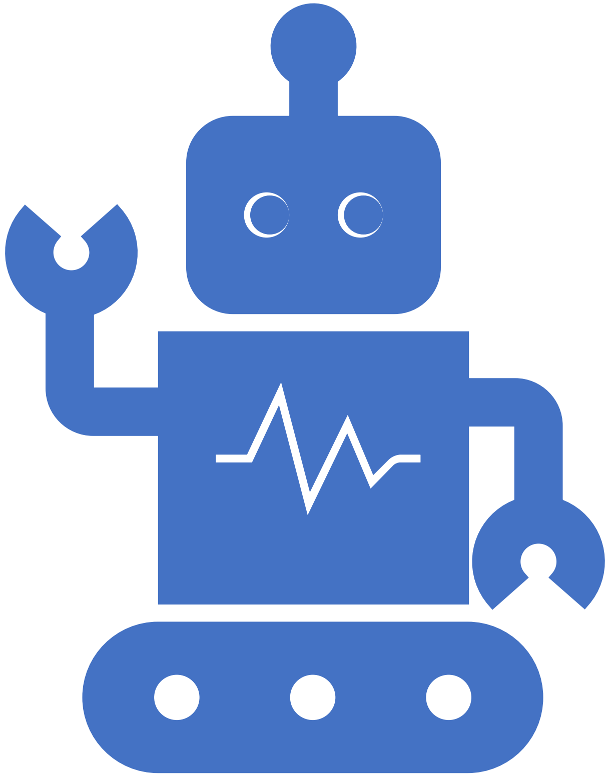
Solving Problems by Search: Informed (Heuristic) Search

Chapter 3



Problem with uniformed search

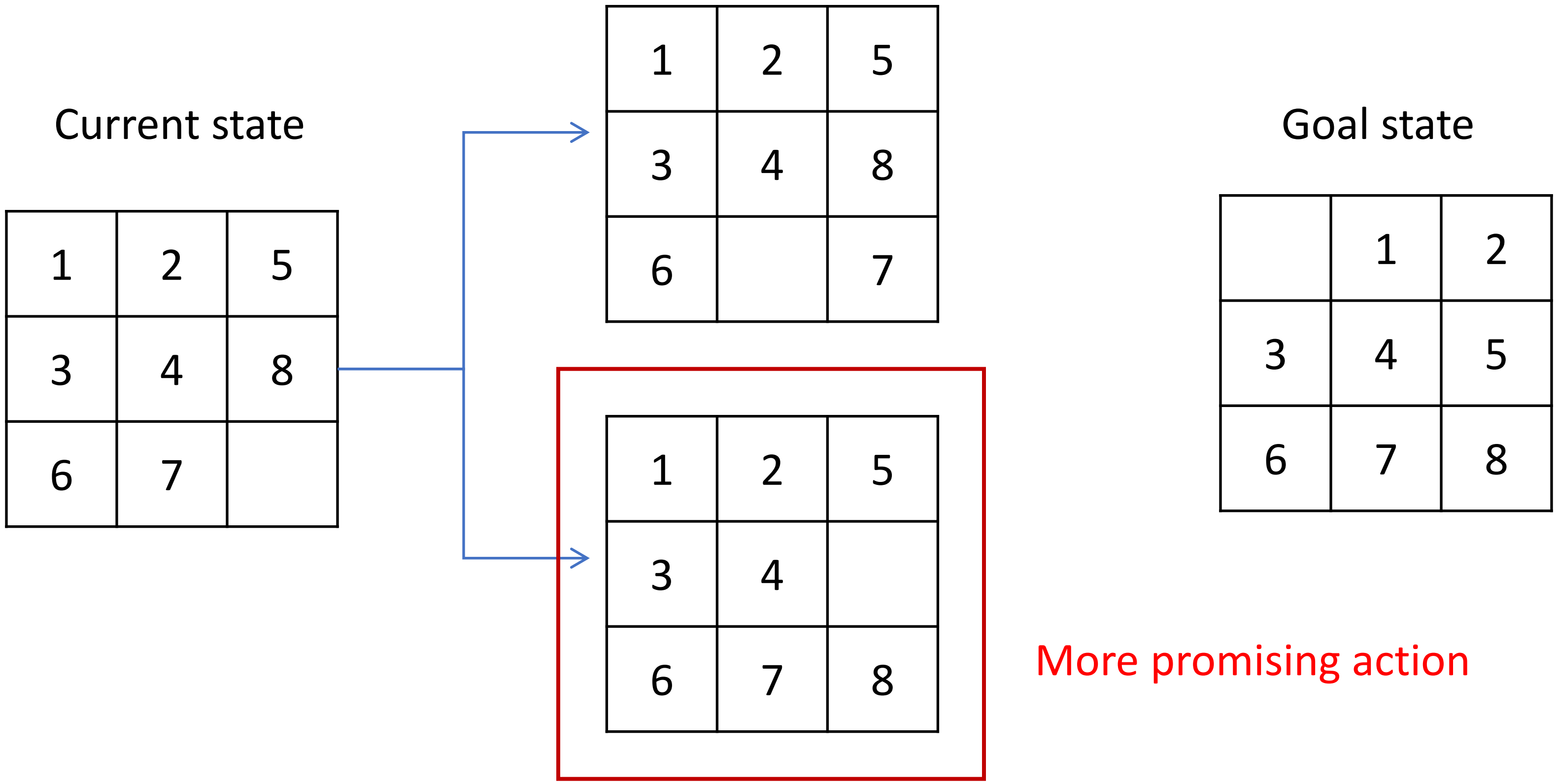
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

C* is the cost of the optimal solution

Informed (heuristic) search

- Uniformed search does not use any information specific to the problem, only its definition.
- Consider the following example:



Informed search

- Our knowledge about the 8-puzzle problem lets us choose the second state because it is more promising
- For the 8 puzzle: number of misplaced tiles
  - In the previous example, the first action leads to a state with $\text{misplaced tiles} = 5$. The second action leads to $\text{misplaced tiles} = 3$
  - Second action is more promising

Next state 1

1	2	5
3	4	8
6		7

Next state 2

1	2	5
3	4	
6	7	8

Goal state

	1	2
3	4	5
6	7	8

Informed (heuristic) search

- Idea: use a function h that tells which state is better:
  - For the 8 puzzle: number of misplaced tiles
- h is called a **heuristic function**
- Algorithms that use a heuristic function (alone or combined with other functions) are called **heuristic algorithms**

Informed (heuristic) search

We will study 3 algorithms:

1. Greedy Best First Search
2. A*
3. Memory-bounded heuristic search

1. Greedy best-first search

- Follows either the tree-search or graph-search pattern
- The list frontier is a priority queue with f as the priority. f is called an evaluation function
- By changing f, we obtain different algorithms. For example:
  - UCS is a best-first graph search with $f(n) = g(n)$, where $g(n)$ is the cost of the current node starting from the initial state
  - Greedy search is a best-first graph search with $f(n) = h(n)$
    - In the previous example of the 8-puzzle, we took: $f(n) = h(n) = misplaced\ tiles$
  - A* is a best-first graph search with $f(n) = g(n) + h(n)$
- Assumptions: $h(n) \geq 0$. If n is a goal then $h(n) = 0$.

RECALL: UCS

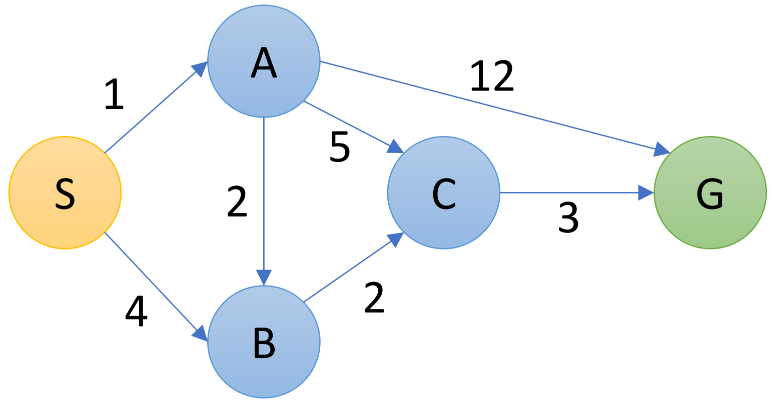
function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by $\boxed{\text{PATH-COST}}$, with *node* as the only element
explored ← an empty set
loop do
  if EMPTY?(*frontier*) **then return** failure
  node ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  add *node*.STATE to *explored*
  for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    child ← CHILD-NODE(*problem*, *node*, *action*)
    if *child*.STATE is not in *explored* or *frontier* **then**
      frontier ← INSERT(*child*, *frontier*)
    else if *child*.STATE is in *frontier* with higher PATH-COST **then**
      replace that *frontier* node with *child*

$f(n) =$		
	$g(n)$	UCS
	$h(n)$	Greedy
	$g(n) + h(n)$	A*

1. Greedy best-first search

- Take $f(n) = h(n)$
- Always tries the node that seems closer to the goal



State	h
S	7
A	6
B	2
C	1
G	0

1. Greedy best-first search example

- Example: Map of Romania
  - Heuristic: h_{sld} straight line distance
  - Use greedy best-first graph search

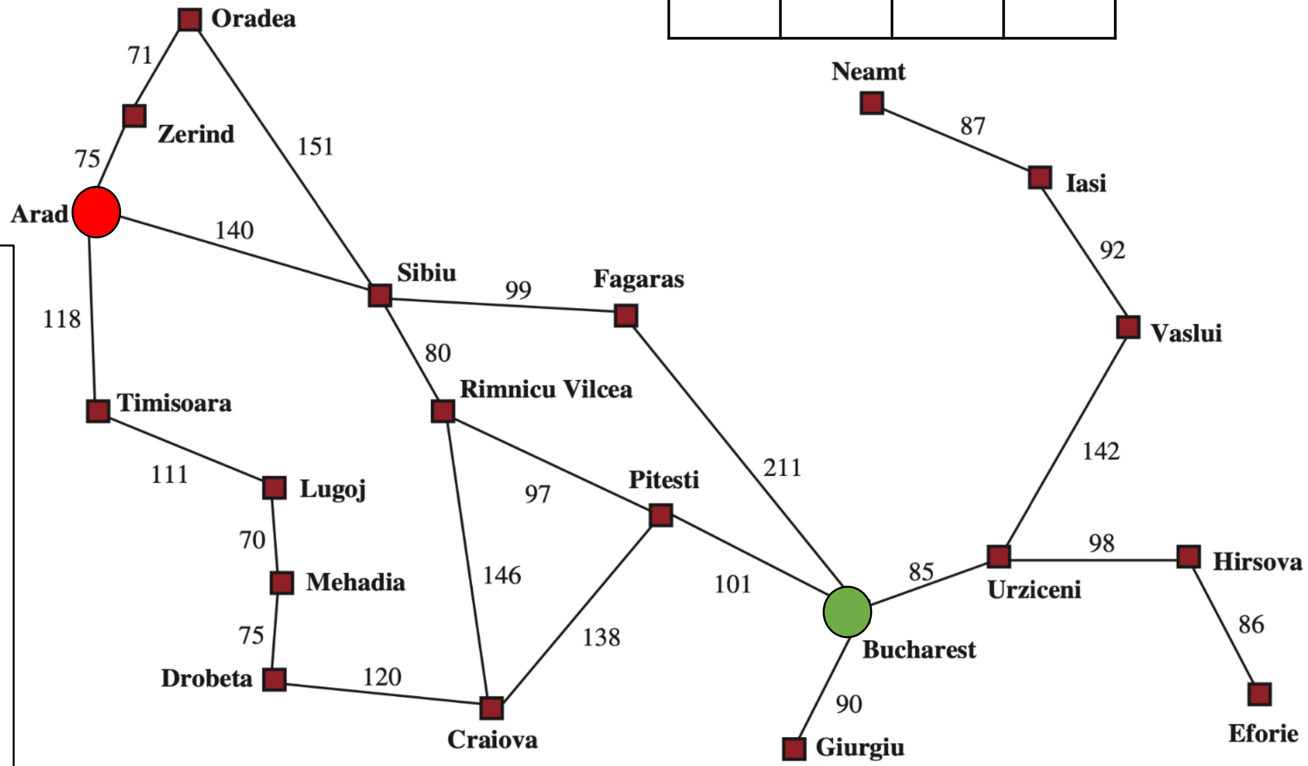
Frontier:

A			
366			

Explored:

h_{sld}

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



1. Greedy best-first search example

Frontier:

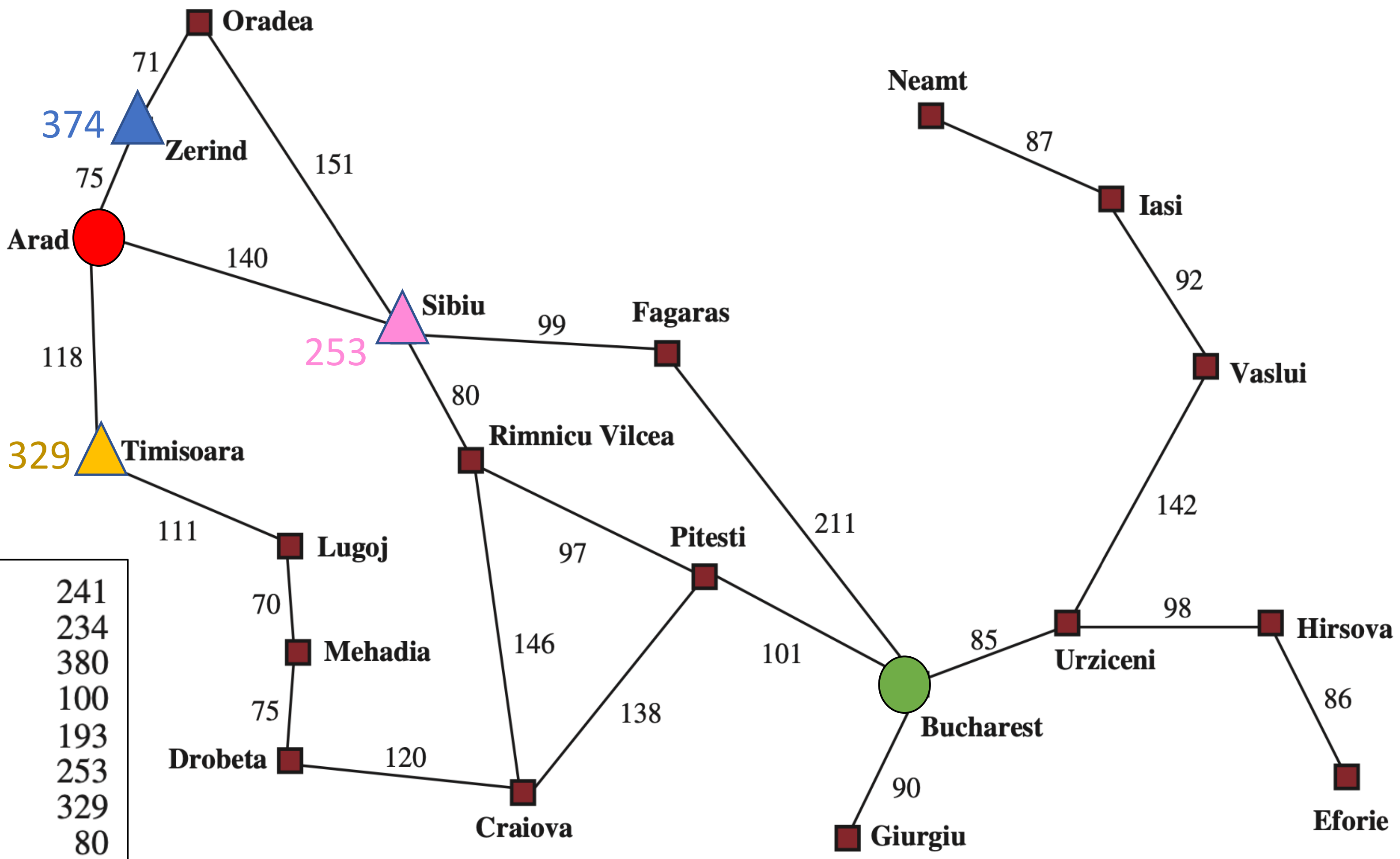
S	T	Z			
253	329	374			

Explored:

A					
366					

h_{sld}

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



1. Greedy best-first search example

Frontier:

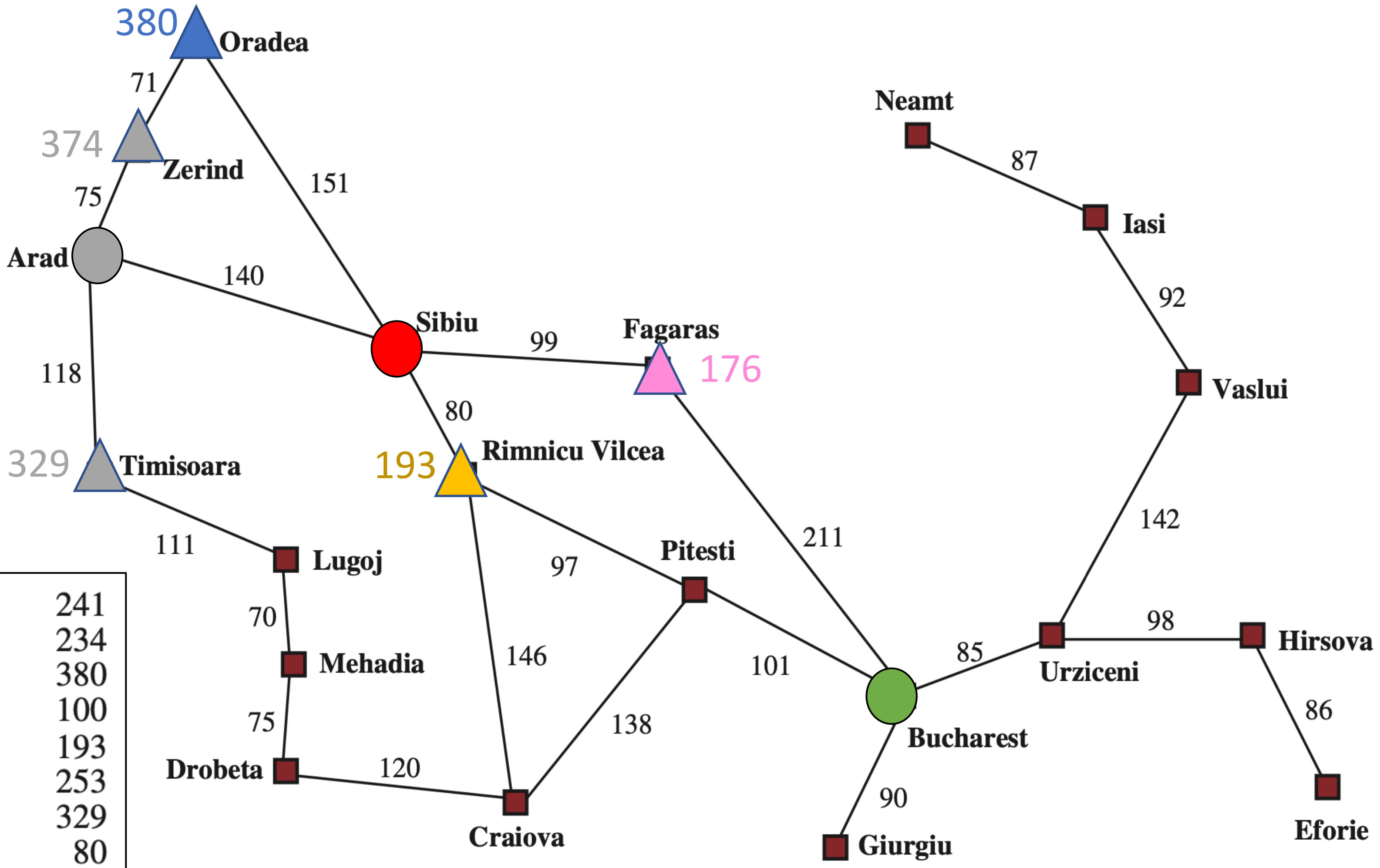
F	RV	T	Z	O	
176	193	329	374	380	

Explored:

A	S				
366	253				

h_{sld}

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



1. Greedy best-first search example

Frontier:

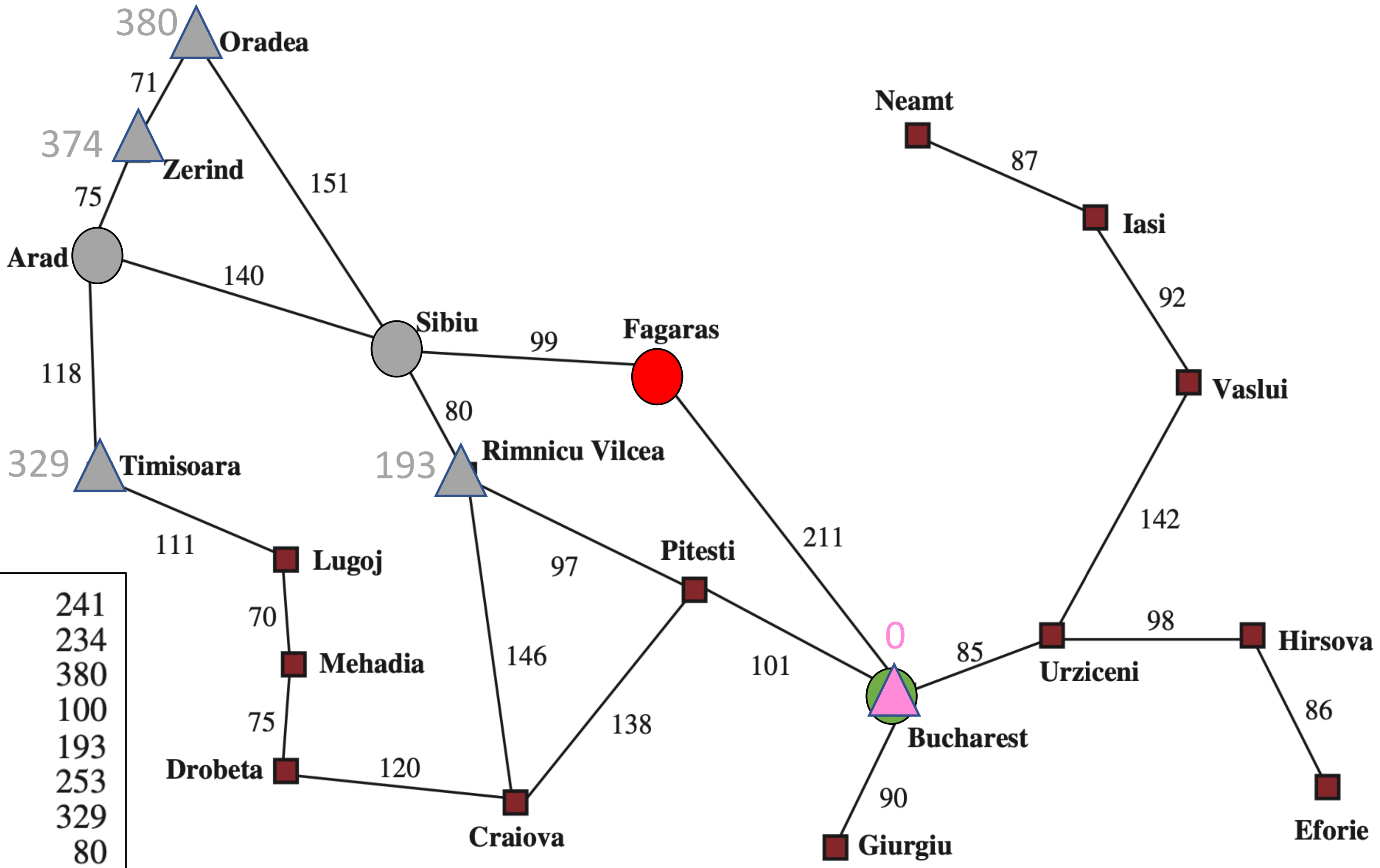
B	RV	T	Z	O	
0	193	329	374	380	

Explored:

A	S	F			
366	253	176			

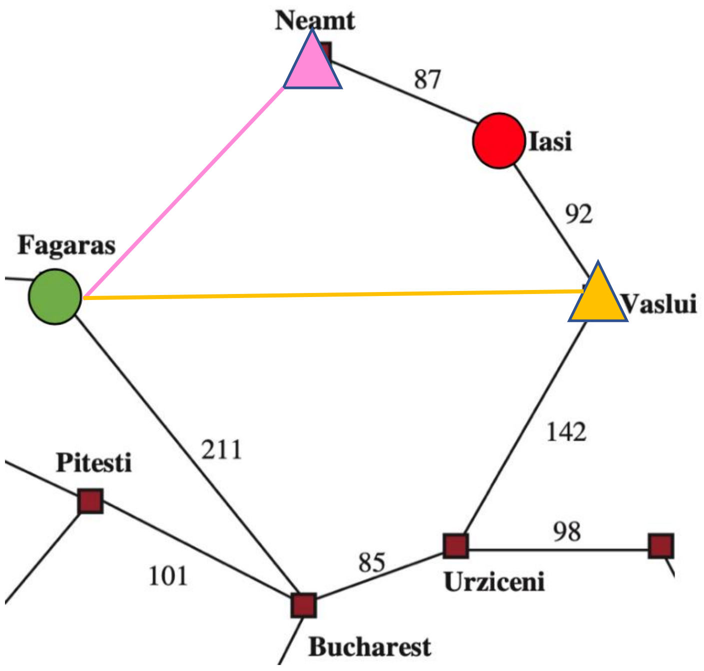
h_{sld}

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



1. Greedy best-first search Performance

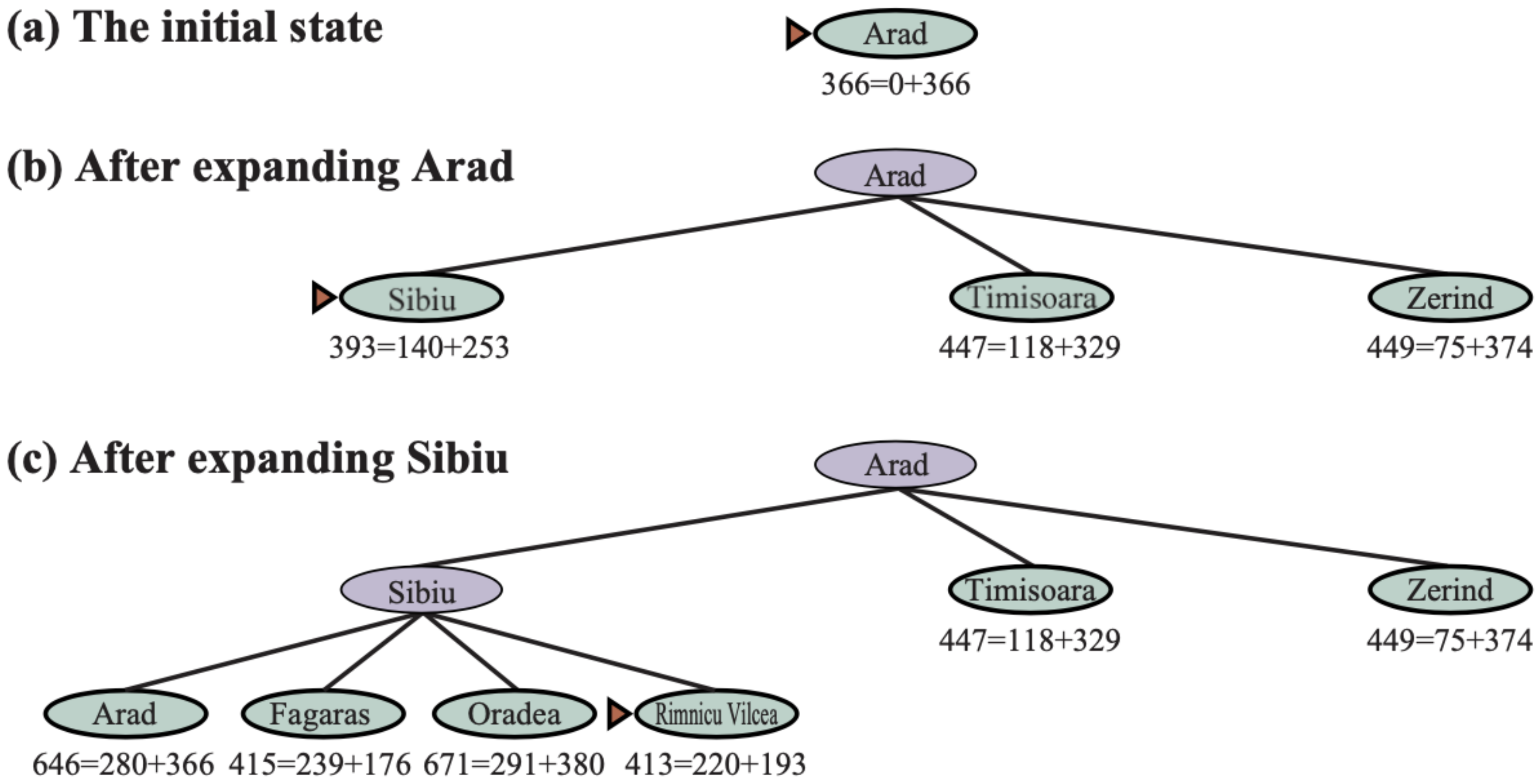
- Completeness:
  - Infinite graphs: No
  - Finite graphs:
    - Tree-search: No. It can get stuck in loops: Go from Iasi to Fagaras:
      - Tree-search: Iasi → Neamt → Iasi → Neamt → …
    - Graph-search: Yes
- Optimality: No, Rimnicu Vilcea and Pitesti path is shorter
- Time Complexity:
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space Complexity:
  - $O(b^m)$, keeps all nodes in memory



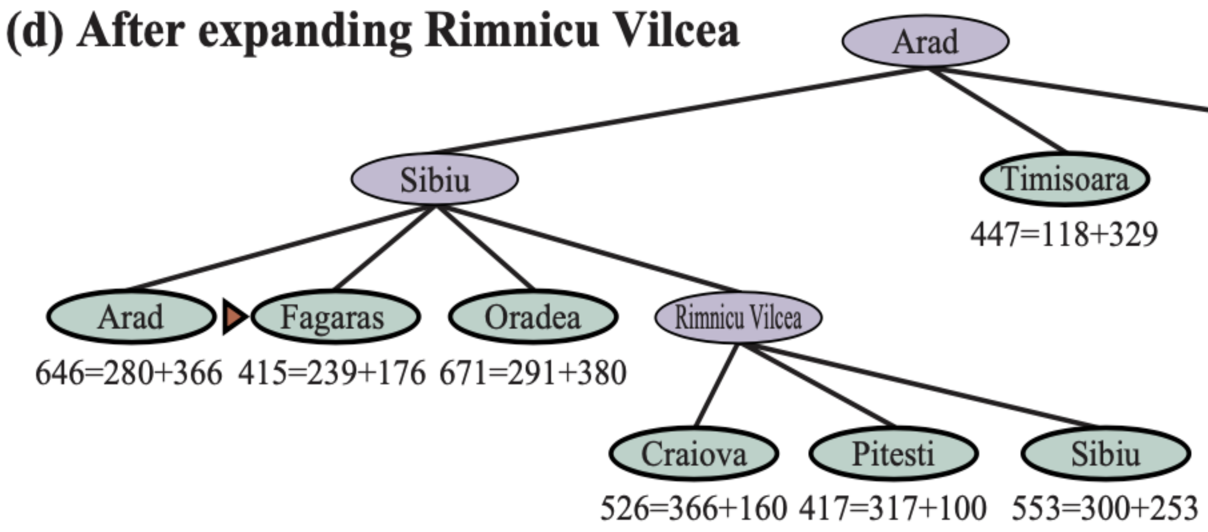
2. A*

- $f(n) = g(n) + h(n)$
  - $g(n)$ = 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
- Combine both cost from the initial state and estimate to the goal
- A* avoids expanding paths that are already expensive
- A* = UCS + Greedy best-first search

2. A* search example (tree-search)

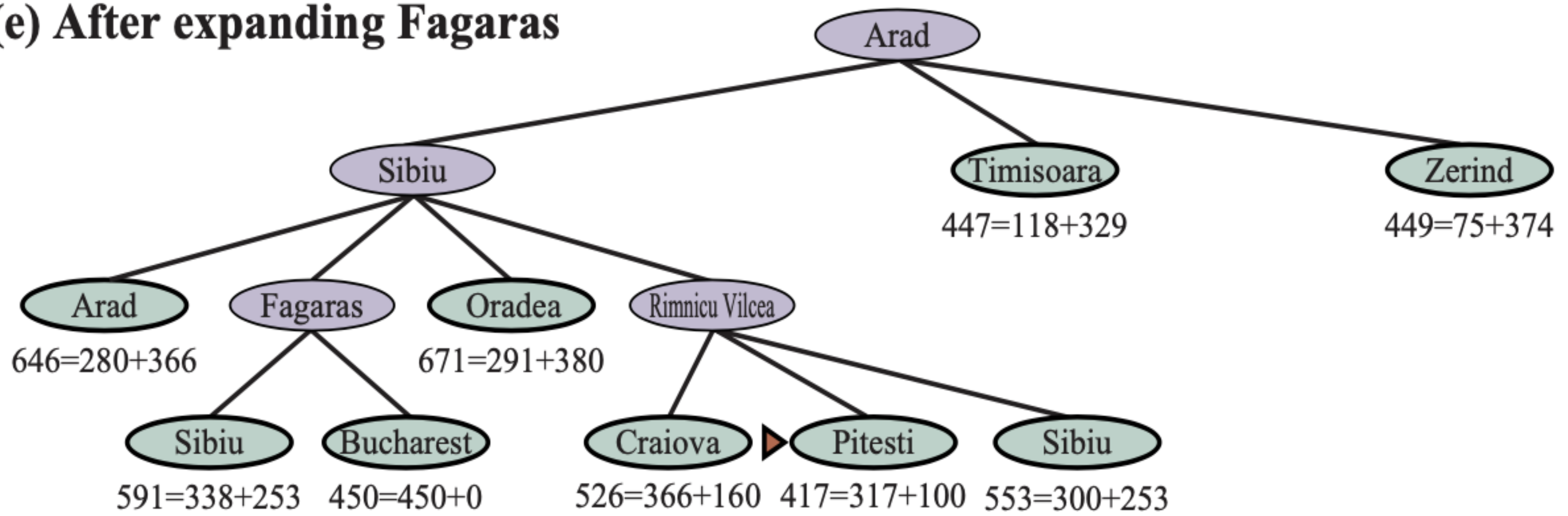


2. A* search example (tree-search)

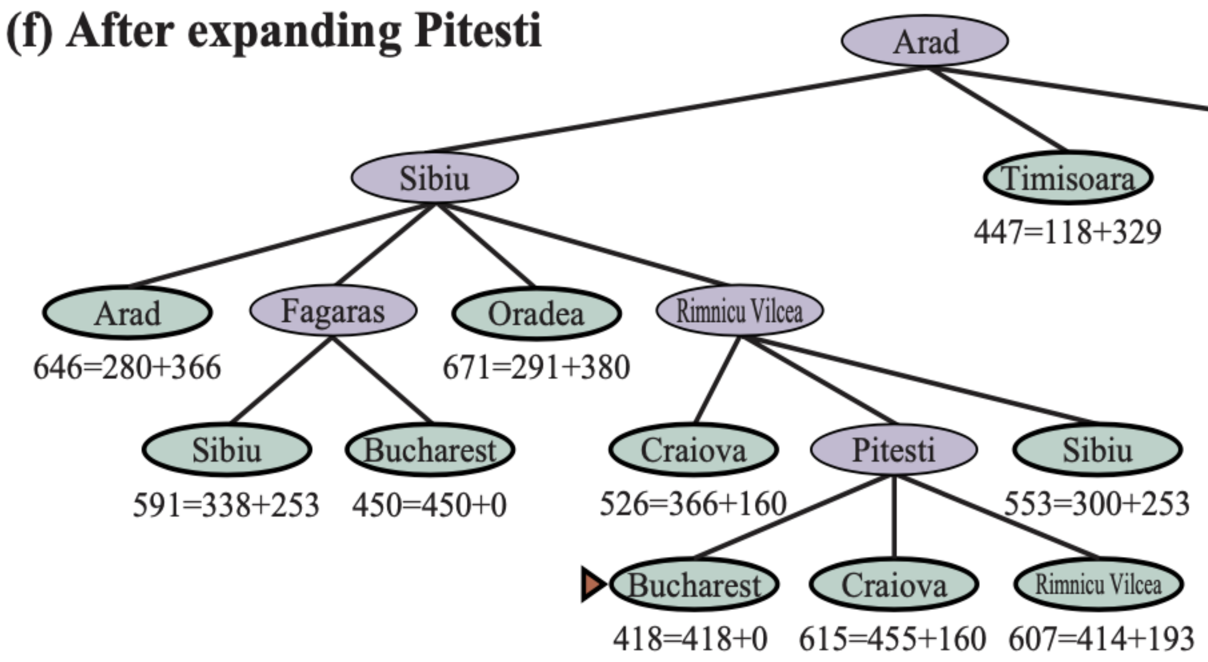


2. A* search example (tree-search)

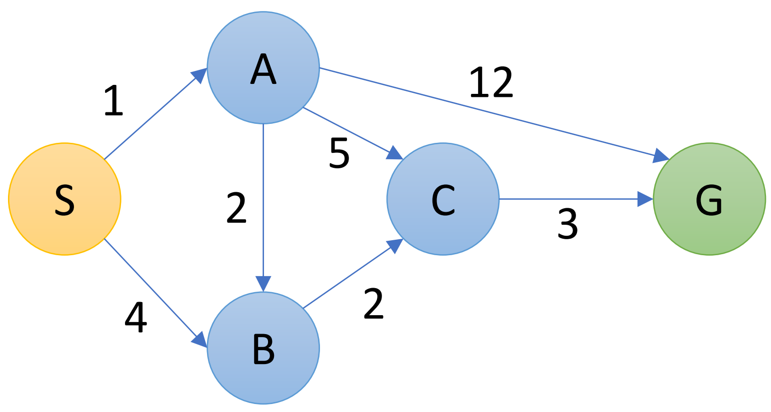
(e) After expanding Fagaras



2. A* search example (tree-search)



2. A* example $f(n) = g(n) + h(n)$



State	h
S	7
A	6
B	2
C	1
G	0

Frontier:

S			
0+7=7			

Explored:

Frontier:

(S)B	**(S)A**		
4+2=6	1+6=7		

Explored:

S			
7			

Frontier:

(S)A	**(S,B)C**		
1+6=7	6+1=7		

Explored:

S	**B**		
7	6		

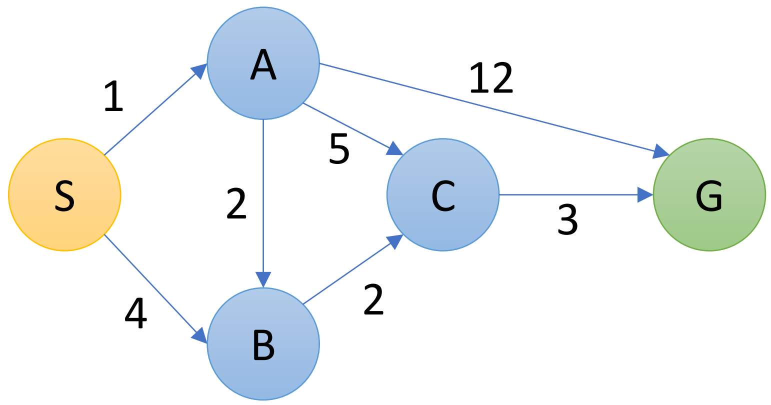
Frontier: (S,A)C = (1+5)+1

(S,B)C	**(S,A)G**		
6+1=7	13+0=13		

Explored:

S	**B**	**A**	
7	6	7	

2. A* example $f(n) = g(n) + h(n)$



State	h
S	7
A	6
B	2
C	1
G	0

Frontier: (S,A)G = 13+0

(S,B,C)G			
9+0=9			

Explored:

S	**B**	**A**	**C**
7	6	7	7

Frontier:

Explored:

S	**B**	**A**	**C**
7	6	7	7

But Wait!

S-A-B-C-G costs 1+2+2+3 = 8

What happened?

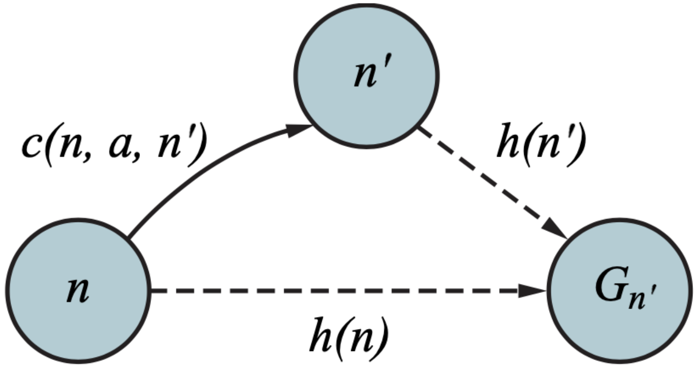
A*: Conditions for Optimality

Condition 1: $h(n)$ must be an admissible heuristic

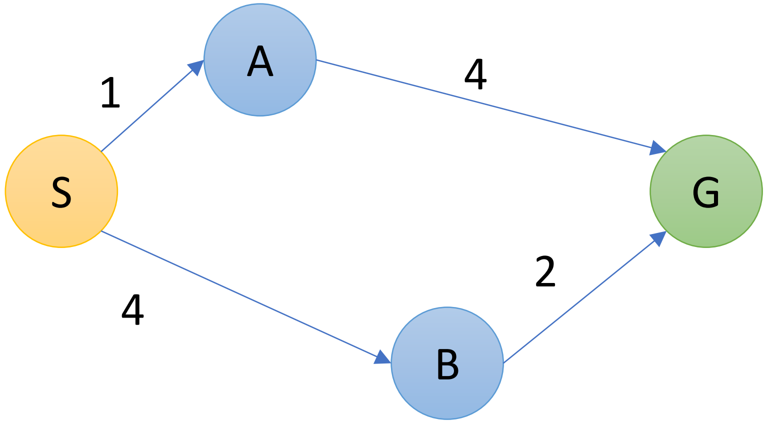
- An admissible heuristic never overestimates the cost to reach the goal: $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n to the nearest goal.
  - $h_{sld}(n)$ never overestimates the actual road distance so it is an admissible heuristic
- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is

A*: Conditions for Optimality

- **<u>Condition 2:</u>** $h(n)$ must be consistent (or monotone):
  - $h(n)$ is consistent if $h(n) \leq c(n, n') + h(n')$ for very successor n' of n
    - Is the last example consistent?
  - If $h(n)$ is consistent then $h(n)$ is admissible. The inverse is not true
  - When $h(n)$ is consistent, the values of $f(n)$ along any path are nondecreasing
  - Consistency is a form of the general **triangle inequality:** each side of a triangle cannot be longer than the sum of the other two sides



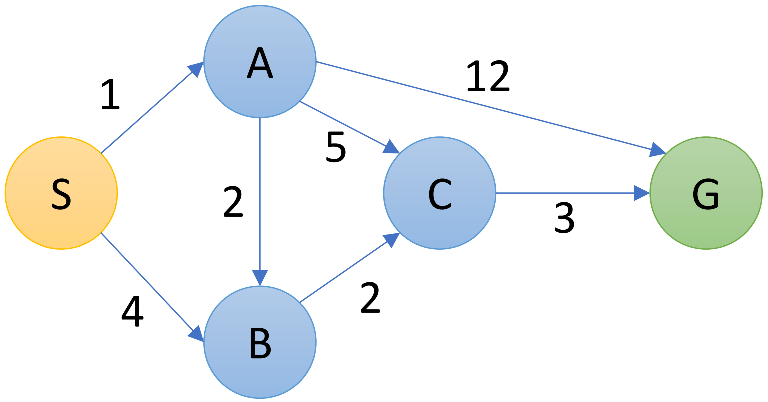
A* with a non-admissible heuristic



State	h
S	4
A	6
B	2
G	0

$$h(n) \nleq h^*(n)$$
$$6 \nleq 4$$

A* with an inconsistent heuristic



State	h
S	7
A	6
B	2
C	1
G	0

$$h(n) \not\leq c(n, n') + h(n')$$

$$h(A) \not\leq c(A, B) + h(B)$$
$$6 \not\leq 2 + 2$$

A* Properties

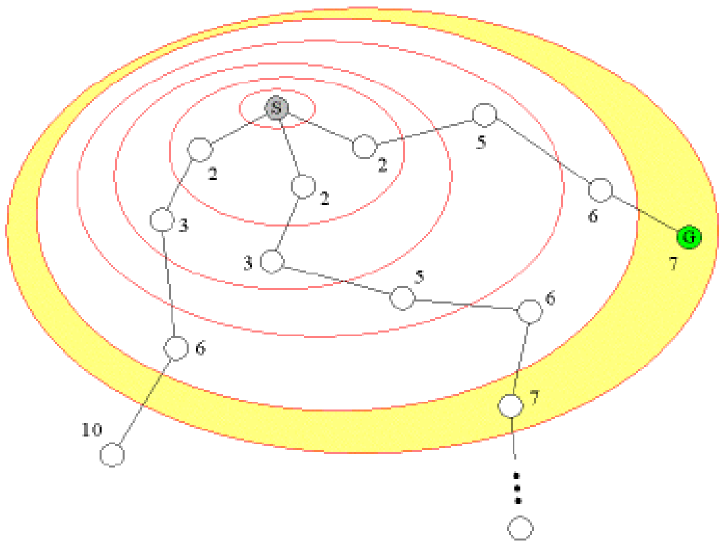
- The tree-search version of A$*$ is optimal if $h(n)$ is admissible
- The graph-search version is optimal if $h(n)$ is consistent

Properties of A*: Completeness

- If C^* is the cost of the optimal goal, then:
  - A* expands all nodes with $f(n) < C^*$
  - A* expands some nodes with $f(n) = C^*$
  - A* expands no nodes with $f(n) > C^*$
- If the number of nodes n with $f(n) \leq C^*$ is finite, then A* is complete
- This is true when all actions have $cost > \varepsilon > 0$ and the branching factor b is finite

Properties of A*: Optimality

- A* expands nodes in order of increasing f value.
- A* gradually adds "f-contours" of nodes
- For any contour, A* examines all nodes in the contour before looking at any contours further out.
- If a solution exists, the goal node in the closest contour to the start node will be found first.



A* Performance

- Completeness:
  - Yes, unless there are infinitely many nodes with $f(n) \leq C^*$
- Optimality:
  - Tree search: Yes, if h is admissible.
  - Graph search: Yes, if h is consistent.
- Time complexity: exponential
- Space complexity: exponential (all nodes in the memory). Will run out of space long before it runs out of time

3. Memory-bounded heuristic search

IDA*: Difference between IDA* and standard IDS

- Cutoff used is the f-cost $(g + h)$ rather than the depth
- At each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration

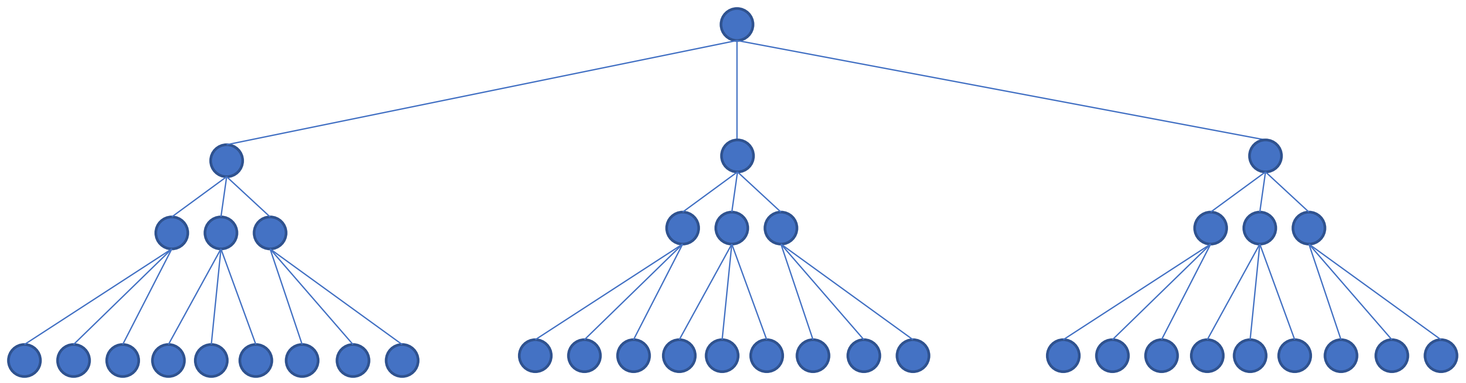
SMA* (Simplified Memory bounded A*)

- When the memory is full, SMA* drops the *worst* leaf node—the one with the highest f-value
- Back-up the value of the forgotten node to its parent

The effect of heuristics on performance

- One way to characterize the quality of a heuristic is the effective branching factor b^*
- Assume N total nodes generated. At depth d, b^* is the branching factor to contain $N + 1$ nodes

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$



Effective Branching Factor b^*

Example:

- If A$*$ finds a solution at depth 5 using 52 nodes, then the effective branching factor is 1.92
- If A$*$ finds a solution at depth 4 using 52 nodes, then the effective branching factor is 2.36

➢A **well- designed heuristic** would have a value of b^* close to 1, allowing large problems to be solved at reasonable computational cost

Comparing two heuristic functions

- Two common heuristics for the 8-puzzle:
  - h_1: The number misplaced tiles (Hamming distance)
  - h_2: The sum of the distances of tiles from their goal positions (total Manhattan distance)

$h_1 = 2$

3	1	
2	4	5
6	7	8

$h_2 = 4$

3	1	
2	4	5
6	7	8

Goal state

	1	2
3	4	5
6	7	8

Comparing two heuristic functions

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Dominance

Better at estimating

- If $h_2(n) \geq h_1(n)$ for all n, and both are admissible: then h_2 dominates h_1 and h_2 is better for search.

- Given any admissible heuristics $h_1, \ldots, h_m$, where none dominate the others:

$$h_{best}(n) = \max(h_1(n), \ldots, h_m(n))$$

$h_{best}(n)$ is also admissible and dominates $h_1, \ldots, h_m$

Relaxed problems

- **Relaxed problem**: problem with fewer restrictions on the actions
  - There are added edges in the graph representing paths that are now allowed
- Admissible heuristics can be derived from the exact solution cost of a **relaxed version** of the problem
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
  - If the rules of the 8-puzzle are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.
- Key point: the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
  - incomplete and not optimal
- A* search expands lowest $g + h$
  - complete and optimal if h is consistent (admissible for tree search)
- Admissible heuristics can be derived from exact solution of relaxed problems