UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Artificial Intelligence

Informed Search

LESSON 5

prof. Antonino Staiano

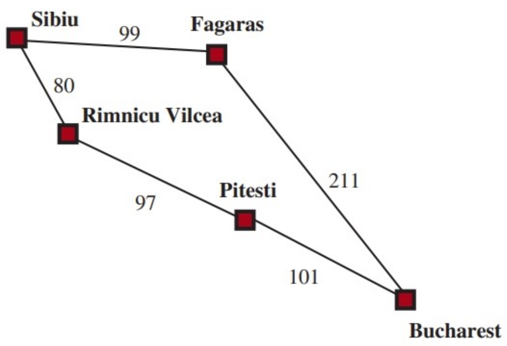
M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

Informed Search

- Uninformed search is based on systematically exploring the search space without using any information (if any) about which nodes are more "promising" than others for the solution
- When some knowledge is available, it can be exploited to improve the effectiveness and efficiency of tree search
  - Uses domain-specific hints about the location of the goals
- Idea
  - Use the available problem-specific knowledge to identify the best node to expand at each step of the general tree-search algorithm
  - This general approach is named best-first search

Uninformed Best-first search

- Recall that the uniformed search algorithms we have seen are a kind of best-first search algorithms
  - A node n with a minimum value of some evaluation function f(n) is selected for expansion
- However, no information on the closeness of the goal node is exploited, rather
  - Uniform-cost search, for instance, expands the least-cost unexpanded node



Informed Best-First Search

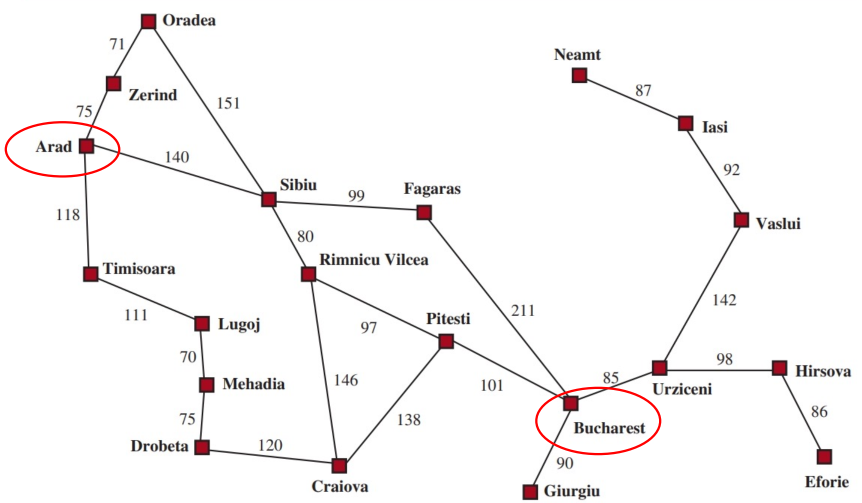
- It is based on quantitatively evaluating how promising a given node n is toward a solution
  - It uses a suitable node evaluation function f(n)
- Different definitions of f(n) correspond to different specific best-first strategies, i.e.,
  - Greedy search
  - A* search and its many variants
- Given a suitable f(n) the search algorithm can be implemented based on the general tree-search algorithm
- Best-first search can be implemented by sorting nodes in the frontier for increasing values of f(n)
  - This means that the node n with the lowest f(n) will be selected for expansion at each iteration

Best-First Search

- To define *f*(n), the cost of the actions that will lead from any given node n to a goal state might be used
- The **exact** cost is usually unknown, thus an **estimate** can be easily computed
  - The estimated cost, a function of the nodes, is denoted with *h*(n)
    - Note that, by definition, *h*(n) = 0 if n contains a goal state (this is the only case in which the cost is **exactly** known)
- For historical reasons *h*(n) is named the heuristic function, and search strategies are named heuristic search
- Heuristic search is one of the earlier achievements of AI (50's) and is still widely used in real-world problems and investigated by researchers in AI

Heuristic Function: Example

- Let's consider the shortest route findings on maps
  - From Arad to Bucharest, using the information on the map



- The actions' cost is evaluated as the route length
- A heuristic function is defined as estimating the distance between any given city and the destination

Heuristic Function: Example

- An easy-to-compute estimate is a straight-line distance (using the geographical coordinates of each city)
- If the destination is Bucharest, the heuristic function h(n) can be defined as the straight-line distance from the node n city to Bucharest
- The values of h(n) are

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

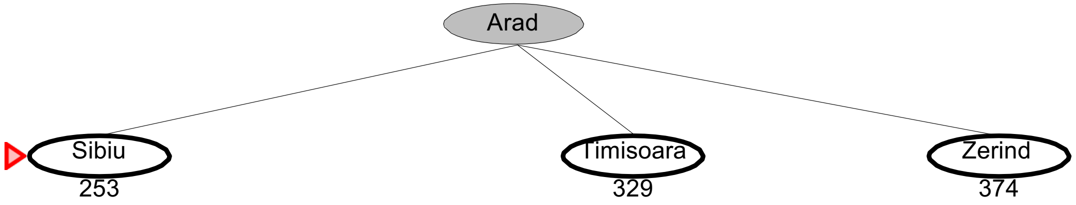
Greedy Best-First Search

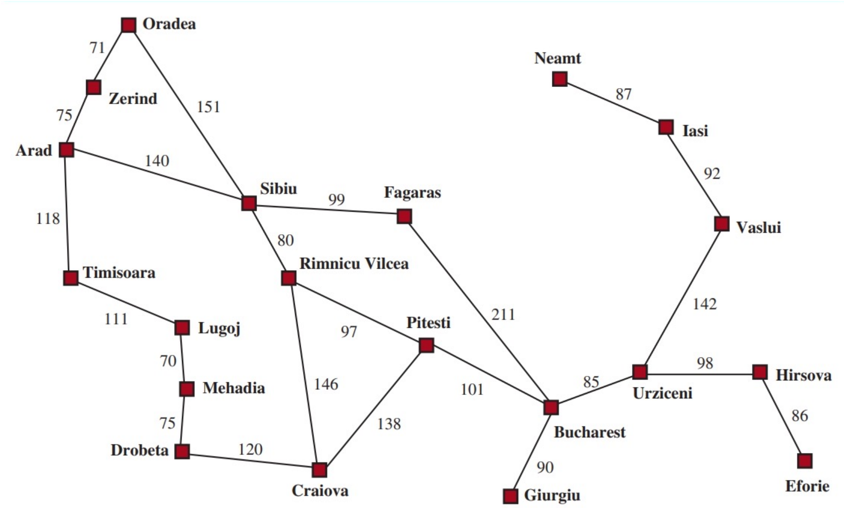
- It is the simplest best-first search strategy
  - Expanding the node closest to the solution, that is, the node with the lowest h(n)
    - $f(n) = h(n)$
- It's a greedy strategy since it favors partial solutions that seem the closest to the actual solution
  - However, that's not an optimal choice

Greedy Best-First Search: Example

- Consider again the problem of finding the shortest route from Arad to Bucharest, using the straight-line distance as a heuristic
- In the following, a greedy strategy is used to build the search tree
  - It is shown the value of f(n)
  - An arrow denotes the node chosen for the expansion

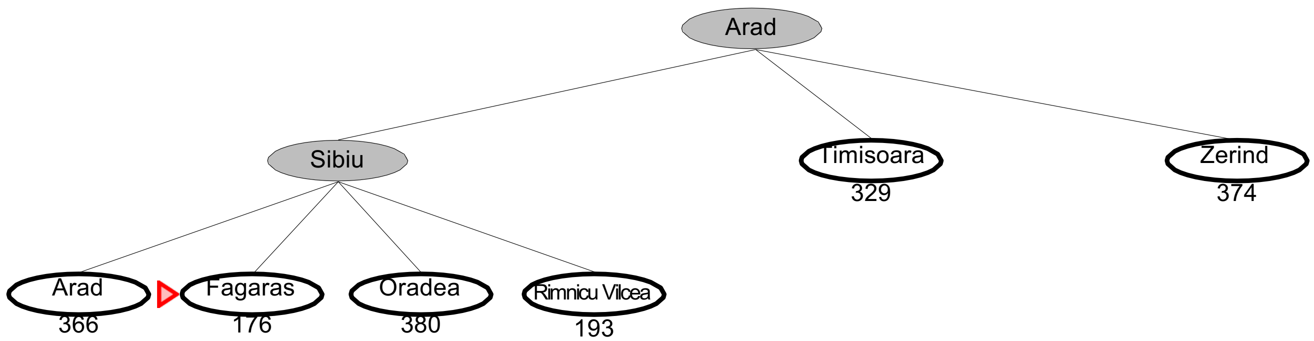
Greedy search example

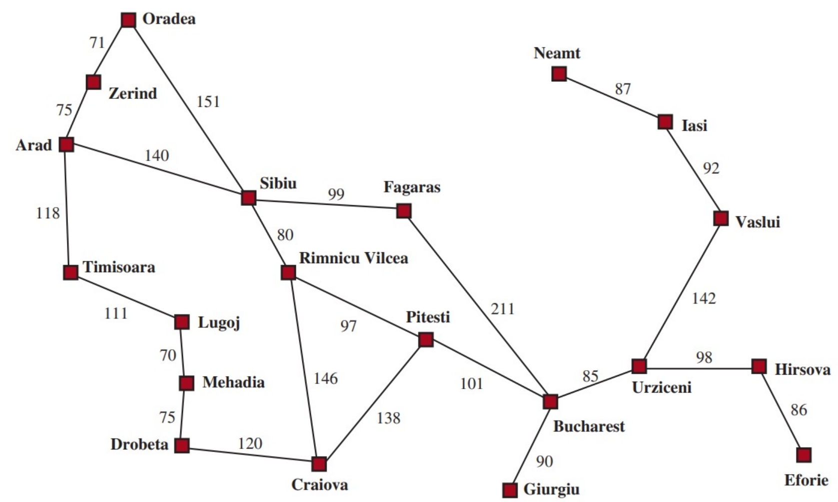




Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

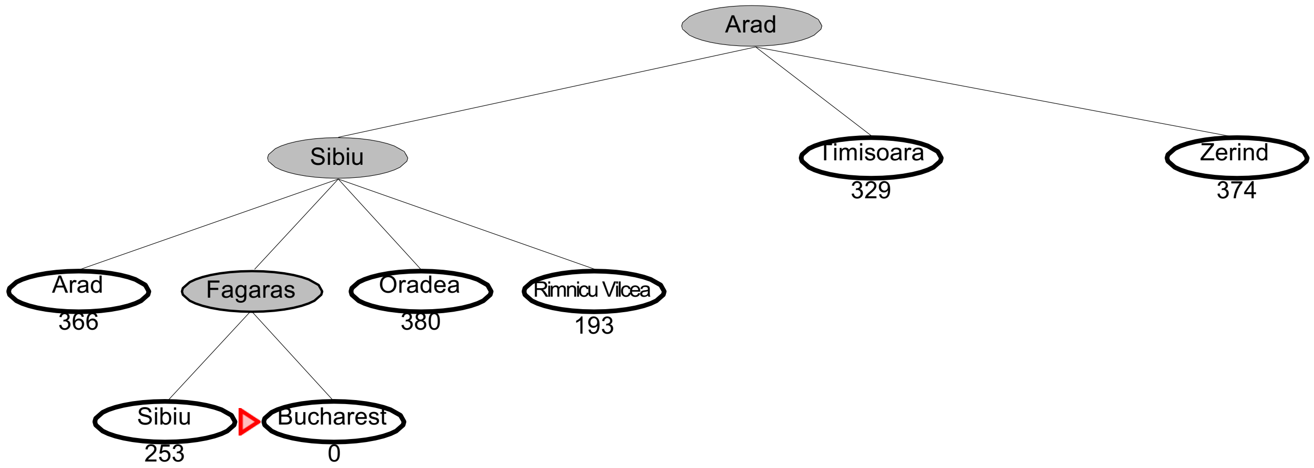
Greedy search example

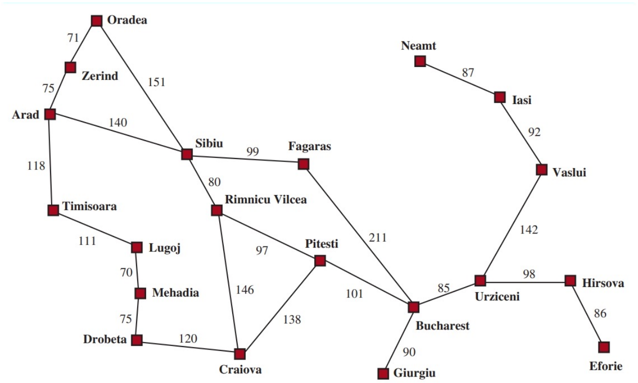




Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy search example

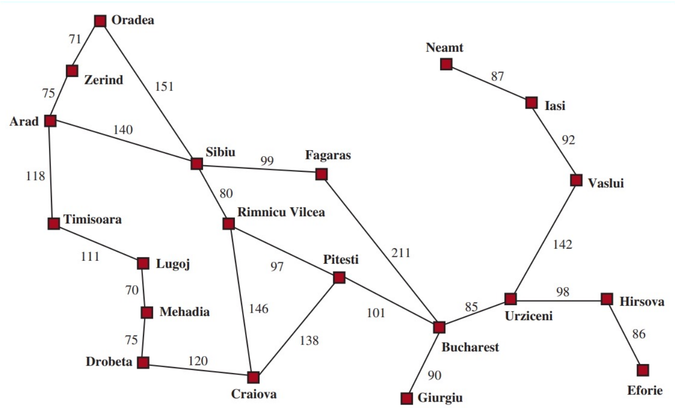




Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Properties of greedy best-first search

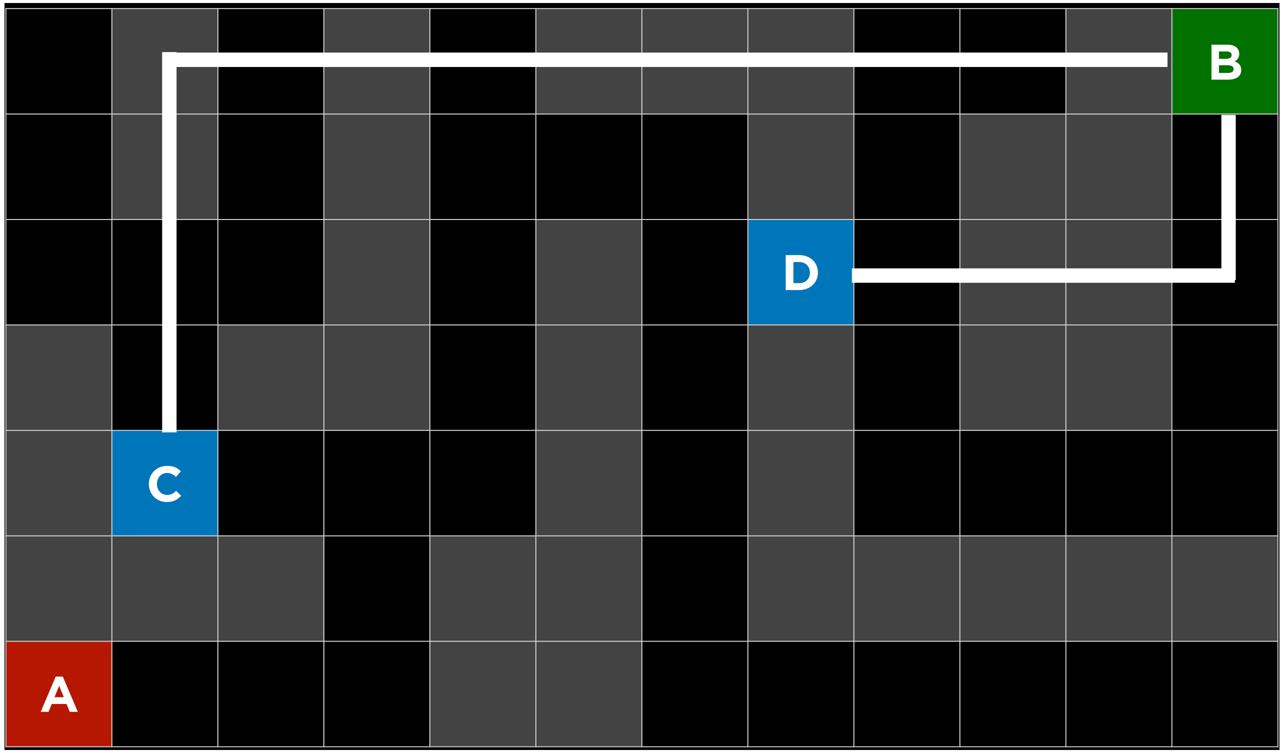
- Complete
  - Unless there are infinite paths or without repeated state-checking (can get stuck in loops)
- Non-optimal
  - For instance, a shorter route exists between Arad and Bucharest (through Sibiu and Rimnicu Vilcea)



- Exponential worst-case time and space complexity, $O(b^m)$
  - Time complexity could be lowered to O(bm), with good heuristic

Greedy Best-Search Example

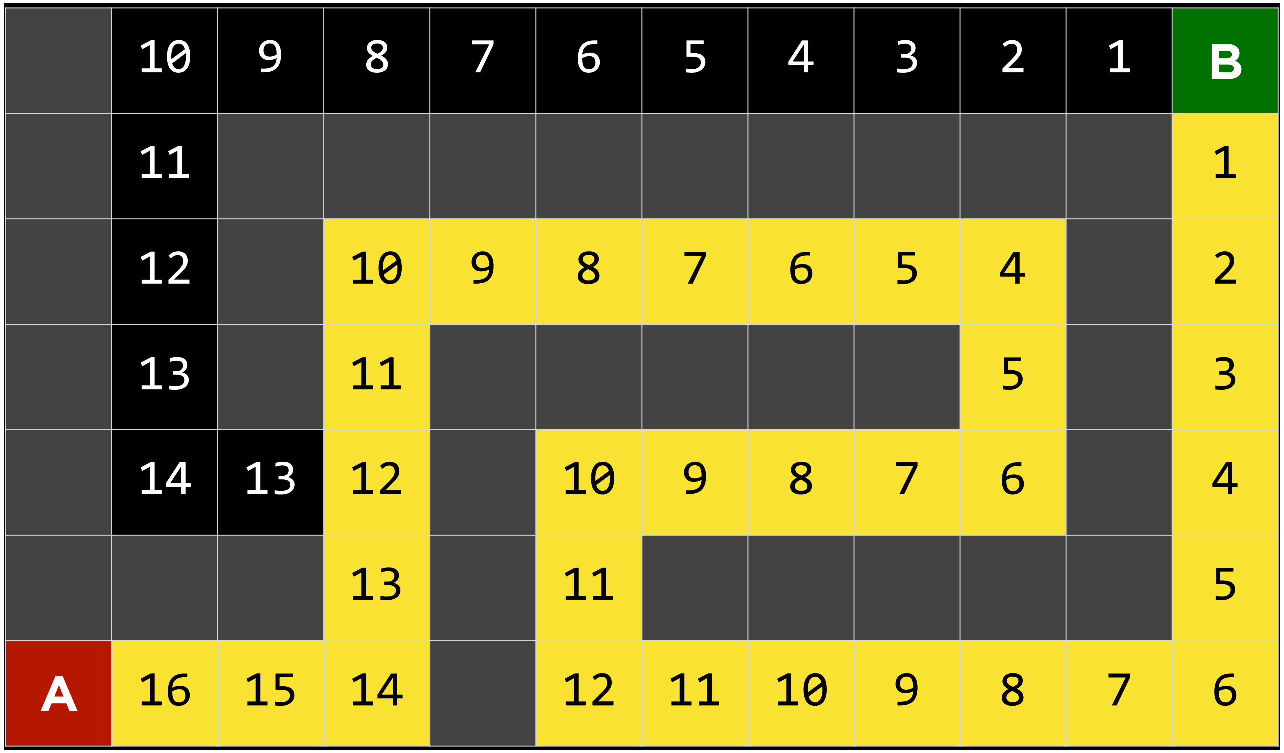
- Heuristic function? Manhattan Distance



Greedy Best-First Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

Greedy Best-Search is non-optimal

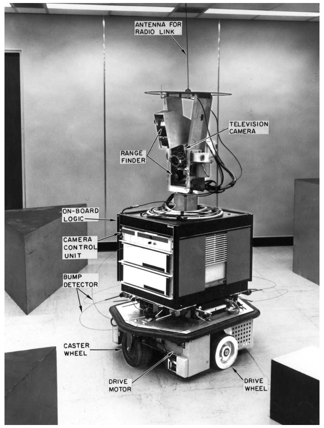


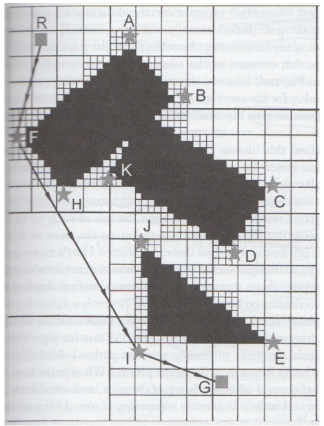
Greedy Best-Search is non-optimal

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

A* Search

- A* is the most relevant best-first search strategy
  - Devised for robot navigation in ‘60s





  - Many variants proposed to tune the trade-off between its effectiveness and efficiency

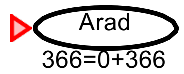
A* Search

- A greedy search estimates the costs of the actions from a node n to expand to a solution and chooses the closest one
- Idea
  - Avoid expanding paths that are already expensive
- A* estimates the total cost of the action sequence from the root to the goal state through n
  - Sum of the path cost to n and the estimated cost from n to a solution
  - The node evaluation function is defined as

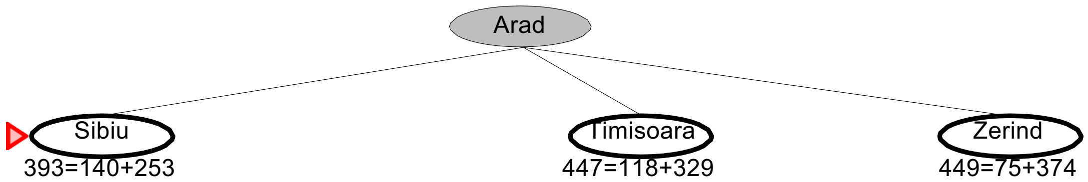
$$f(\mathtt{n}) = \underbrace{g(\mathtt{n})}_{\text{path cost from root to n}} + \underbrace{h(\mathtt{n})}_{\text{estimated cost of the shortest path from node n to a goal}}$$

A* Search: Example

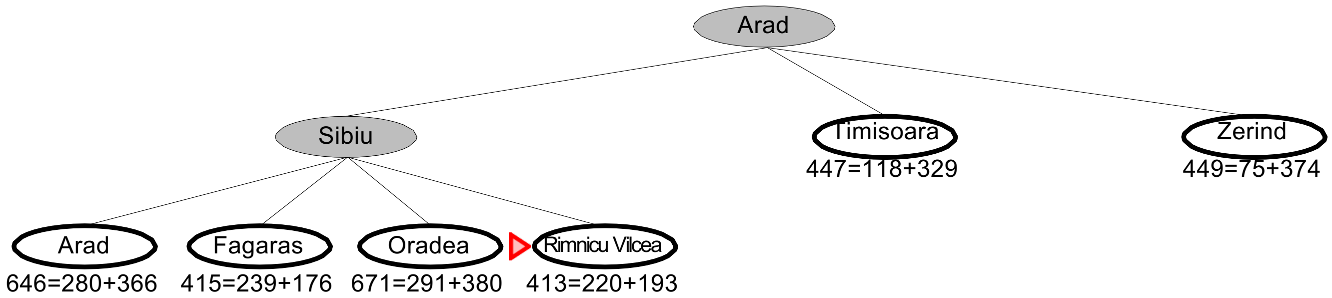
- The following search tree built by A* is shown from the previous example (Arad-Bucharest, using the straight-line distance heuristic)
  - The value of $f(n) = g(n) + h(n)$ is also shown for each node



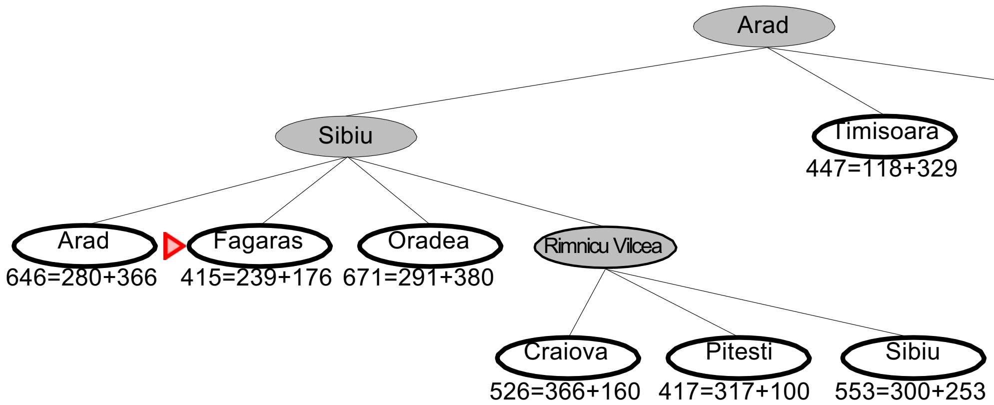
A* Search: Example



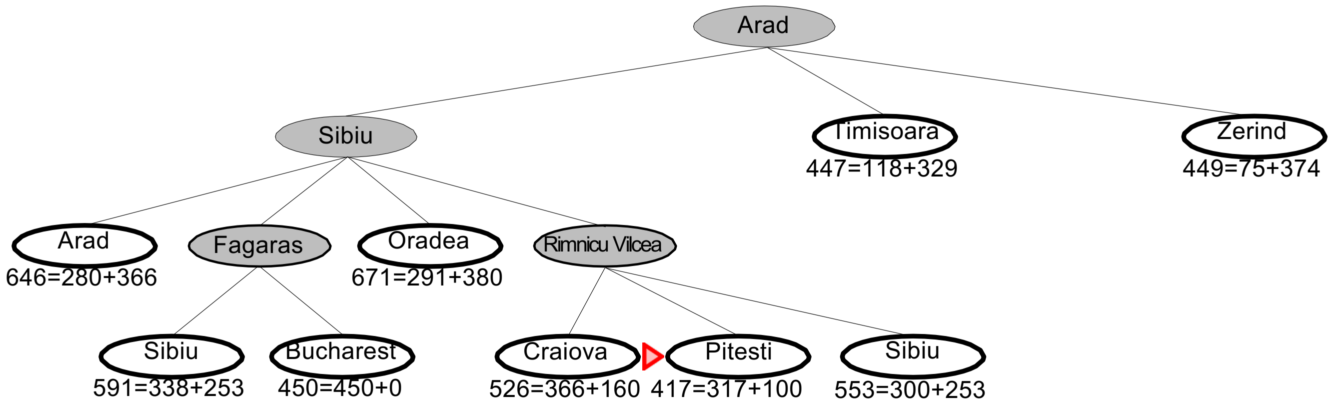
A* Search: Example



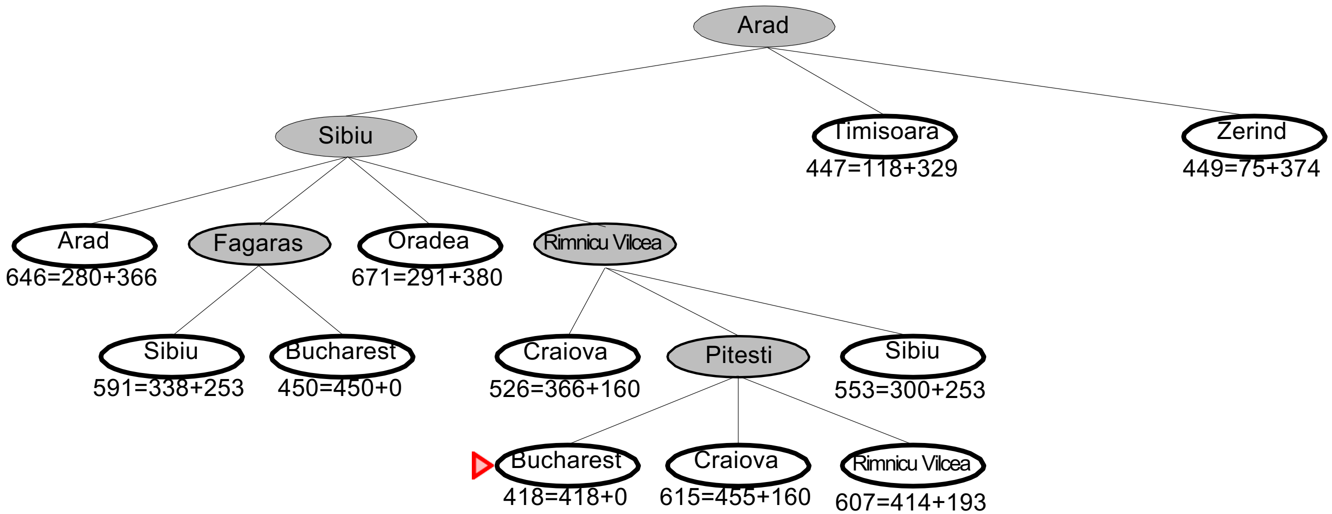
A* Search: Example



A* Search: Example



A* Search: Example



Proof (by contradiction) of Optimality of A*

- Let's pretend the optimal path cost is C*, but algorithm A* returns a path with cost C > C* (a suboptimal path)
  - There must exist some node n which is on the optimal path and unexpanded

$$
\begin{aligned}
f(n) &> C^* \quad \text{(otherwise } n \text{ would have been expanded)} \\
f(n) &= g(n) + h(n) \quad \text{(by definition)} \\
f(n) &= g^*(n) + h(n) \quad \text{(because } n \text{ is on an optimal path)} \\
f(n) &\leq g^*(n) + h^*(n) \quad \text{(because of admissibility, } h(n) \leq h^*(n)\text{)} \\
f(n) &\leq C^* \quad \text{(by definition, } C^* = g^*(n) + h^*(n)\text{)}
\end{aligned}
$$

A* Conditions for Optimality

- A slightly stronger property is called consistency
  - A heuristic h(n) is consistent if, for every node n and every successor n’ of n generated by an action a, we have: $h(n) \leq c(n,a,n') + h(n')$
- This is a form of the triangle inequality

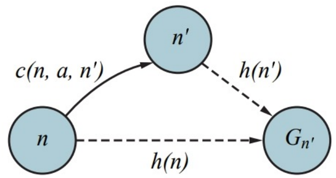
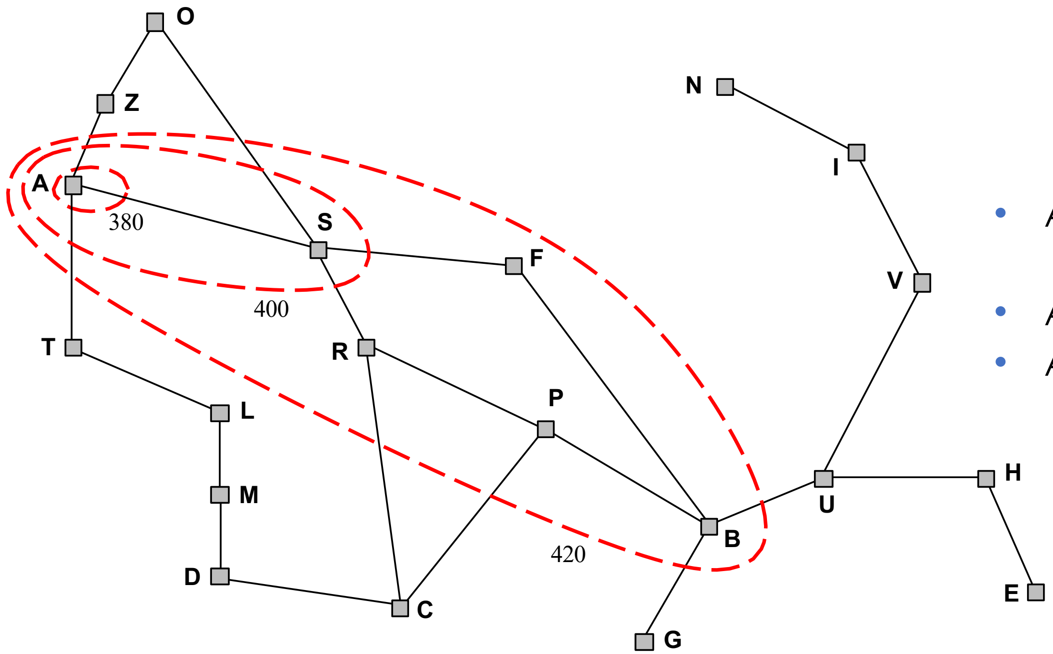


Figure 3.19 Triangle inequality: If the heuristic h is **consistent**, then the single number $h(n)$ will be less than the sum of the cost $c(n,a,a')$ of the action from n to n' plus the heuristic estimate $h(n')$.

- An example of a consistent heuristic is the straight-line distance we have seen earlier for getting to Bucharest
  - Every consistent heuristic is admissible (but not vice versa), so with a consistent heuristic A∗ is cost-optimal

Optimality of A*: Search contours

- A* expands node in order of increasing f value (if h is consistent)
  - Gradually adds f-contours of nodes
    - Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



- A∗ expands all nodes with $f(n) < C*$
  - Surely expanded nodes
- A∗ expands some nodes with $f(n) = C*$
- A∗ expands no nodes with $f(n) > C*$

Properties of A* Search

- Complete
  - Unless there are infinitely many nodes with f ≤ f(goal)
- Optimal
  - Provided that the heuristic is admissible
- Optimally efficient
  - Expand fewer nodes with respect to other optimal algorithms using the same heuristic
- Exponential worst-case time and space complexity
  - However, A* is much more efficient (generates a much smaller number of nodes) than other uninformed and informed search strategies

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	16	15	14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	1+16	15	14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	1+16	2+15	14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

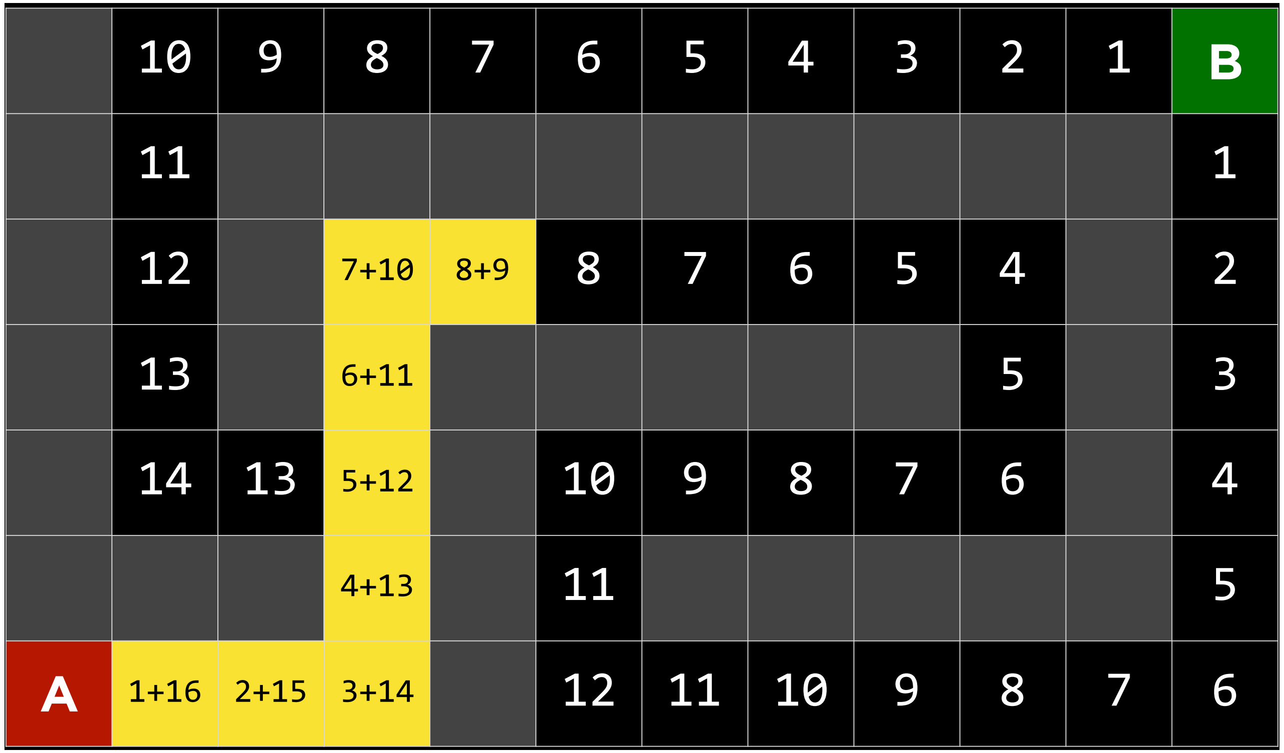
A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		10	9	8	7	6	5	4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	9	8	7	6	5	4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search


10 9 8 7 6 5 4 3 2 1 B
11 1
12 7+10 8+9 8 7 6 5 4 2
13 6+11 5 3
14 13 5+12 10 9 8 7 6 4
4+13 11 5
A 1+16 2+15 3+14 12 11 10 9 8 7 6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	7	6	5	4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	6	5	4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	5	4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	13		6+11						5		3
	14	13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	13		6+11						14+5		3
	14	6+13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	13		6+11						14+5		3
	14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	10	9	8	7	6	5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	9	8	7	6	5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	8	7	6	5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	7	6	5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	6	5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	16+5	4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	16+5	17+4	3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	16+5	17+4	18+3	2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	16+5	17+4	18+3	19+2	1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

A* Search

	11+10	12+9	13+8	14+7	15+6	16+5	17+4	18+3	19+2	20+1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

Improving A* Search

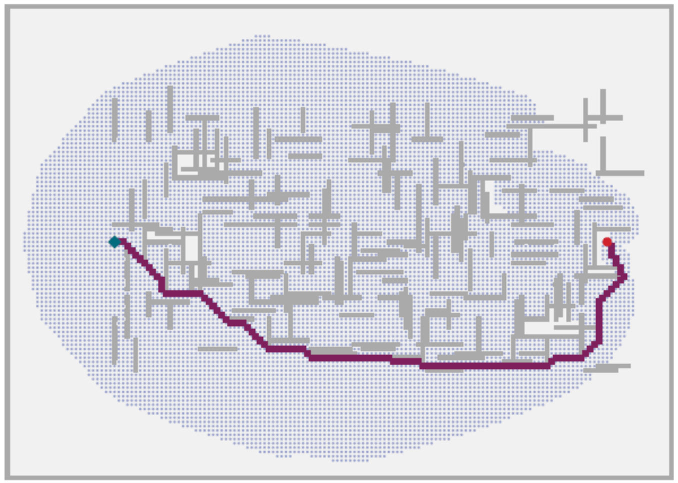
- Good heuristics can lower time and memory demand, particularly w.r.t. uninformed search
- Nonetheless, in many practical problems A* may result unfeasible
  - it expands a lot of nodes
  - Alternative approaches
    - We can explore fewer nodes (taking less time and space) if we are willing to accept solutions that are suboptimal, but are satisficing solutions ("good enough")
      - Non-optimal A* variants (i.e., find quickly suboptimal solutions)
    - Optimal A* variants with reduced memory requirements and a small increase in execution time

Weighted A* Search

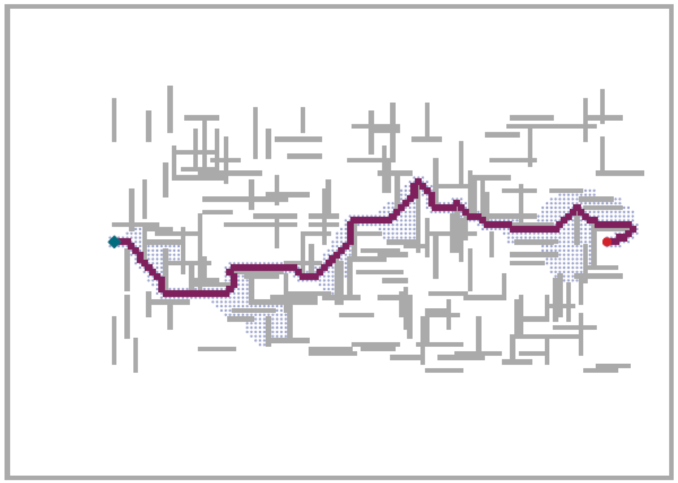
- Example (suboptimal solutions)
  - road engineers use the detour index, which is a multiplier applied to the straight-line distance to account for the typical curvature of roads.
    - A detour index of 1.3 means that if two cities are 10 miles apart in straight-line distance, a good estimate of the best path between them is 13 miles
    - For most localities, the detour index ranges between 1.2 and 1.6
- Generalizing
  - The heuristic value is weighted more heavily
    - $f(n) = g(n) + W \times h(n)$, for some $W > 1$
    - This is called Weighted A* search
      - If the optimal solution cost is C^*, the weighted A* solution cost is between C^* and $W \times C^*$

A* Search and Weighted A* Search

- Two searches on a grid
  - (a) A* search find the optimal solution exploring a large portion of the state space
  - (b) Weighted A* search (W=2) finds a costlier solution but with a faster search time



(a)



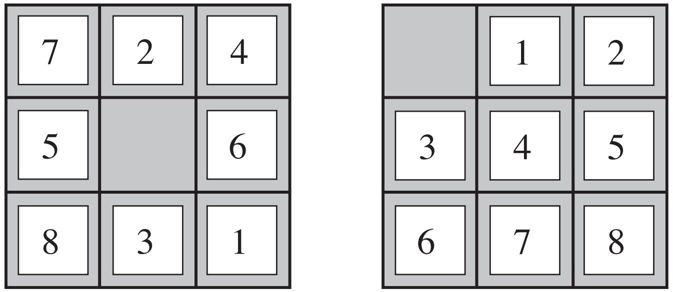
(b)

Defining Heuristic Functions

- Intuitively, the more accurate the estimate of the cost to the solution from a given node provided by the heuristic function, the more efficient a best-first algorithm is
- Defining a good (i.e., accurate) heuristics is therefore crucial for informed search
- Moreover, heuristics must be admissible to guarantee the optimality of A*

Defining Heuristic Functions: Example

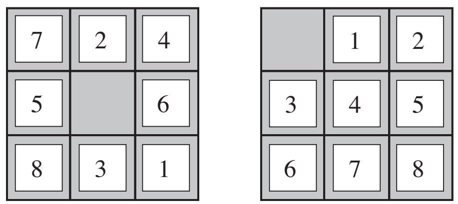
- We have seen that a possible heuristic for route finding in maps is the straight-line distance
- Consider now the 8-puzzle problem
  - Remember that about 3×10^{10} nodes are generated on average by breadth-first (uninformed) search therefore a good heuristic can be of great practical help also in this toy problem



Start State Goal State

Defining Heuristic Functions: Example

- Admissible heuristics for 8-puzzle:
  - h_1: Number of misplaced tiles
  - h_2: Sum of the distances of each tile from its goal position
    - City block or Manhattan distance
- For instance, the value of h_1 and h_2 for the state (left) w.r.t. the goal state (right):
  - h_1(start state): 8 (all 8 tiles are misplaced)
  - h_2 (start state): 3+1+2+2+2+3+3+2 = 18 (tiles 1 to 8)



Start State Goal State

Defining Heuristic Functions

- Generally, not straightforward to define a heuristic function
  - The approach is to set h(n) to the exact cost of a relaxed version of a problem at hand
- Examples
  - k-puzzle: by relaxing the constraint that tiles can move only to a free adjacent square, and allowing them to move to any adjacent square, one obtains $h_2(n)$
  - k-puzzle: similarly, allowing tiles to move to any square (even non-adjacent and occupied ones), one obtains $h_1(n)$
  - route finding on maps: by relaxing the constraint that an adjacent city can be reached only through the corresponding route, and allowing one to move straight to it, one obtains the straight-line distance heuristic

Choosing Heuristic Functions

- On the other hand, for some problems, it can be possible to define several admissible heuristics $h_1,\dots,h_p$ (e.g., h_1 or h_2 for 8-puzzle)
- In this case, one can choose or define a single heuristic h which dominates all the other ones, i.e.:
  - For each node n, $h(n)>=h_i(n)$, $i=1,\dots, p$
- It is easy to see that such a heuristic is admissible, and provides a more accurate estimate of the cost to the solution than $h_1,\dots,h_p$
- To this aim, h can be defined as follows
  - If there is a dominating heuristic among $h_1,\dots,h_p$, choose it as the heuristic for the problem at hand
  - Otherwise, for a given node n use the following heuristic
    - $h(n) = \max \{h_1(n),\dots h_p(n)\}$, which dominates by definition $h_1, \dots, h_p$

Evaluating Heuristic Functions

- To evaluate the quality of heuristic functions the concept of effective branching factor (denoted as b*) is used:
  - let N be the number of nodes generated by A* for a given problem, and d be the depth of the (optimal) solution
  - b* is defined as the branching factor of a **uniform** tree of depth d containing N nodes, which is the solution of the equation:
    - $N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$
- The lower the value of b*, the better the heuristic
- Since b* depends on the problem instance, it is usually evaluated empirically as the average over a set of instances

Evaluating Heuristic Functions

- Example: Empirical evaluation of the effective branching factor of heuristics *h*1 and *h*2 for the 8-puzzle (used in A*), and, for comparison, of an uninformed search strategy, BFS

	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

Figure 3.26 Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, A^* with h_1 (misplaced tiles), and A^* with h_2 (Manhattan distance). Data are averaged over 100 puzzles for each solution length d from 6 to 28.