Introduction to Artificial Intelligence

Lecture 3: Informed Search

Nina Gierasimczuk

Problem solving is an essential part of intelligence.

Problem solving can be represented as a search strategy.

Uniformed searching is **blindly** following transitions from states to states.

Informed search uses extra information in making choices of transitions.

This information is given in the form of **heuristic function**.

Recap: Search problems formally

Formally, a **search problem** consists of the following elements:

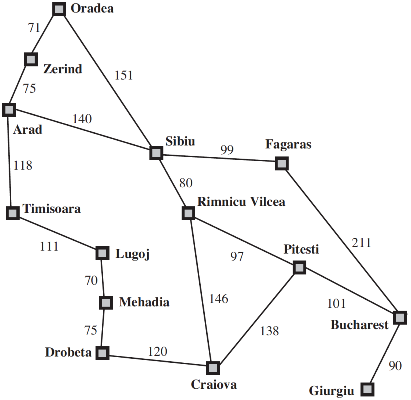
- s_0: Initial state
- $\textsc{Actions}(s)$: Returns the set of actions **applicable** in state s.
- $\textsc{Results}(s, a)$: Returns the state s' reached from s by executing action a.
- $\textsc{Goal-Test}(s)$: Returns `true` if s is goal state, otherwise `false`.
- $\textsc{Step-Cost}(s, a)$: The cost of executing action a in s. Most often we will assume $\textsc{Step-Cost}(s, a) = 1$ for all s and a.

A state g is called a **goal state** if $\textsc{Goal-Test}(g) = \texttt{true}$.

A **solution** to a search problem is a **sequence of actions** (a **path**) from s_0 to a goal state. It is **optimal** if it has minimum sum of step costs.

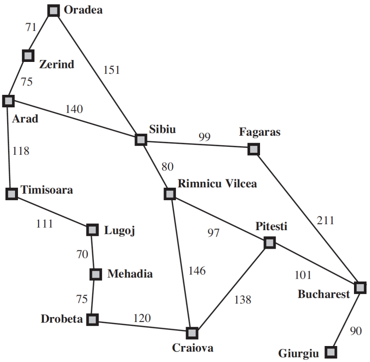
A Real World Problem

Route-finding



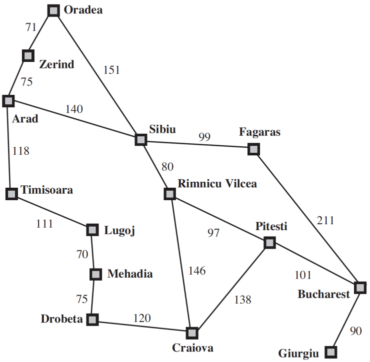
Route-finding Problem: From Arad to Bukarest

Problem Solving by BFS and DFS



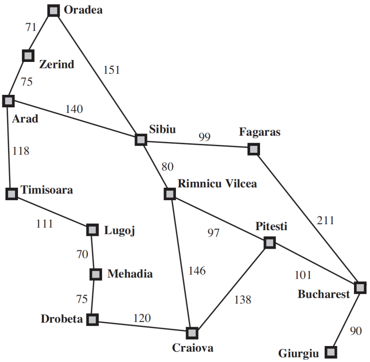
Route-finding Problem: From Arad to Bukarest

Problem Solving by BFS and DFS



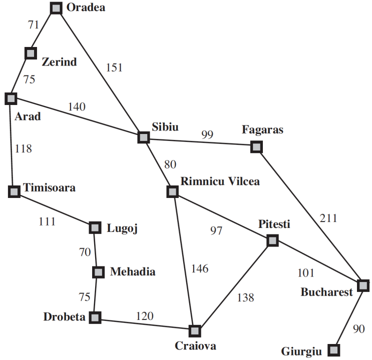
Route-finding Problem: From Arad to Bukarest

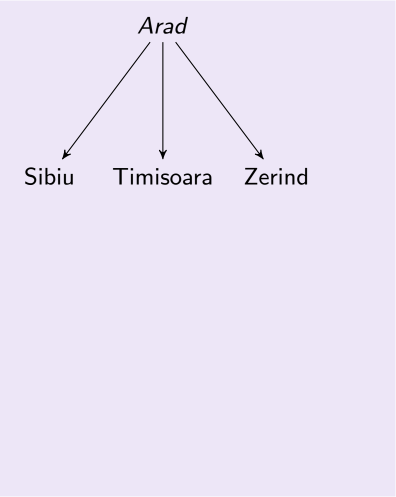
Problem Solving by BFS and DFS



Route-finding Problem: From Arad to Bukarest

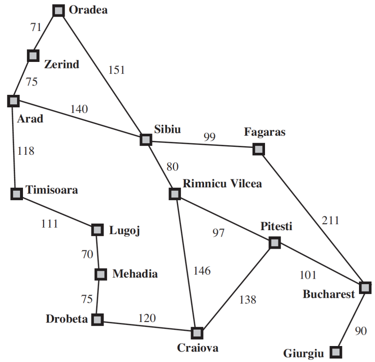
Problem Solving by BFS and DFS

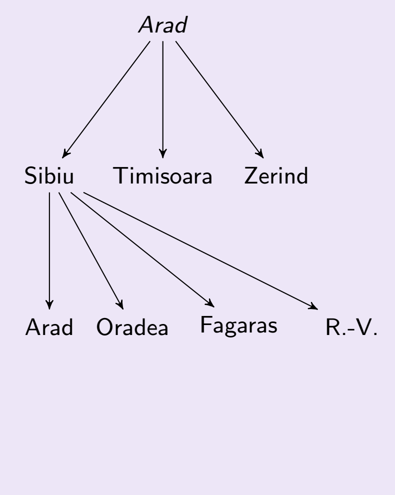




Route-finding Problem: From Arad to Bukarest

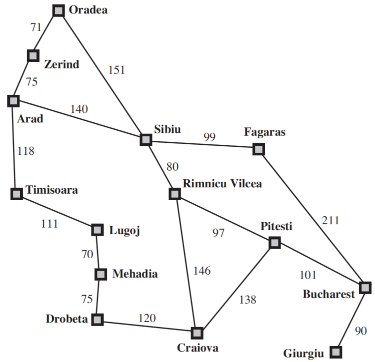
Problem Solving by BFS and DFS

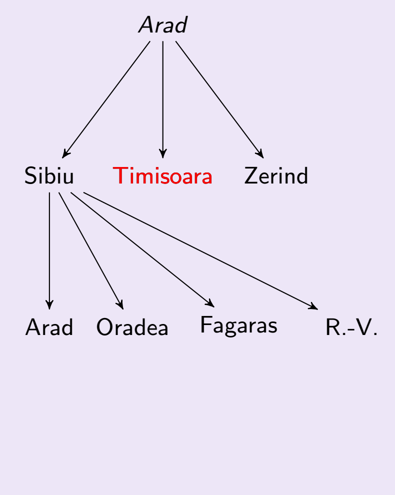




Route-finding Problem: From Arad to Bukarest

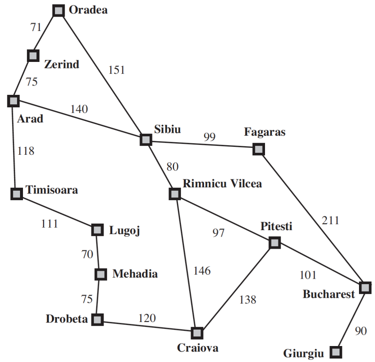
Problem Solving by BFS and DFS

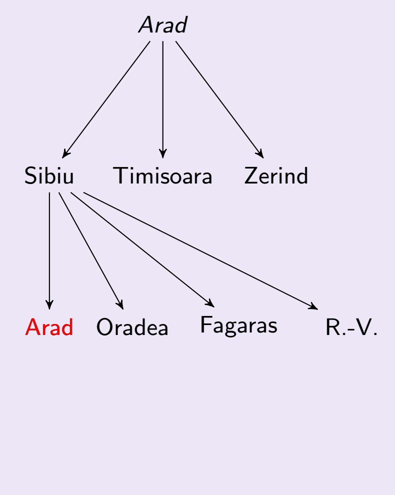




Route-finding Problem: From Arad to Bukarest

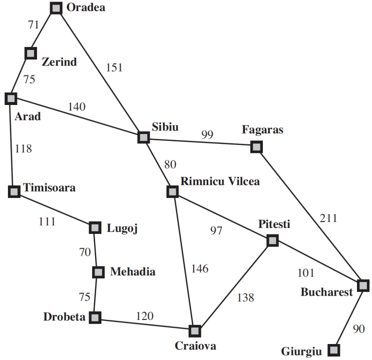
Problem Solving by BFS and DFS

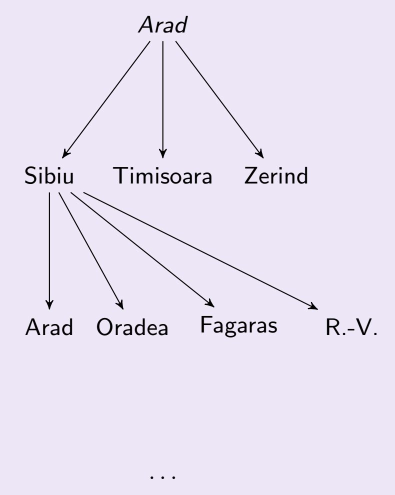




Route-finding Problem: From Arad to Bukarest

Problem Solving by BFS and DFS





The Tree-Search algorithm

function TREE-SEARCH (problem) returns a solution, or failure
  frontier := $\{s_0\}$ (initial state)   // we initialise the frontier
  loop do
    if *frontier* $= \emptyset$ **then return** failure
    choose a node *n* from *frontier*
    remove *n* from *frontier*
    if *n* is a goal state **then return** solution
    for each child *m* of *n*   // we expand *n*
      add child m to frontier

Search strategies of Tree-Search and Graph-Search

Different search strategies can be achieved by simply changing how choose node from frontier and add child to frontier work.

Breadth-first search (BFS):

- Frontier is **queue** (FIFO).
- choose node from frontier: dequeue node from frontier.
- add child to frontier: enqueue node to frontier.

Evaluation of Search Algorithms

For Tree-Search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional
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.

Uniform-cost search

When all step-costs are equal BFS is optimal.
UCS is an extension of BFS which is optimal for any step-cost function.

Uniform-cost search

When all step-costs are equal BFS is optimal.
UCS is an extension of BFS which is optimal for any step-cost function.

Instead of expanding the shallowest node
UCS expands the node n with the lowest path-cost g(n).

Uniform-cost search

When all step-costs are equal BFS is optimal.
UCS is an extension of BFS which is optimal for any step-cost function.

Instead of expanding the shallowest node
UCS expands the node n with the lowest path-cost g(n).

Frontier is a priority queue ordered by g.

Uniform-cost search

When all step-costs are equal BFS is optimal.
UCS is an extension of BFS which is optimal for any step-cost function.

Instead of expanding the shallowest node
UCS expands the node n with the lowest path-cost g(n).

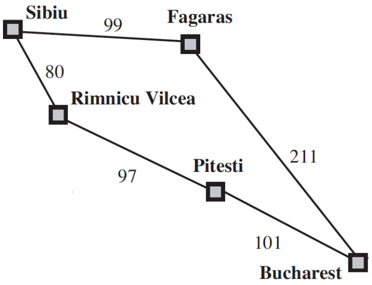
Frontier is a priority queue ordered by g.

Differences with BFS:

- Goal-Test is applied when a node is selected for expansion
  (rather than when it is first generated)
  (the first goal node that is generated may be on a suboptimal path)
- a test is added for a better path to a node currently on the frontier

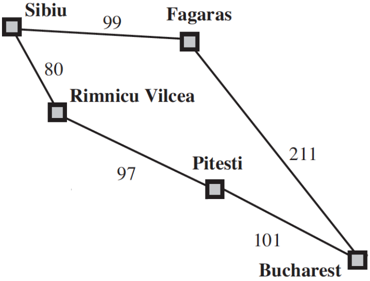
Back to route-finding: Sibiu to Bucharest

Uniform-cost Search



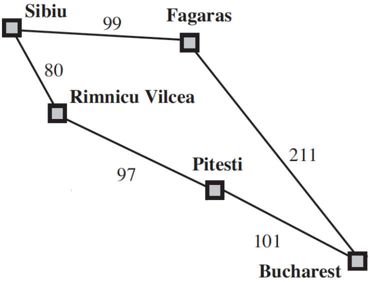
Back to route-finding: Sibiu to Bucharest

Uniform-cost Search



Back to route-finding: Sibiu to Bucharest

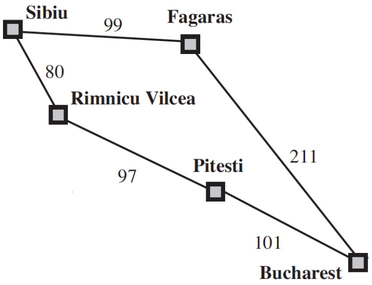
Uniform-cost Search

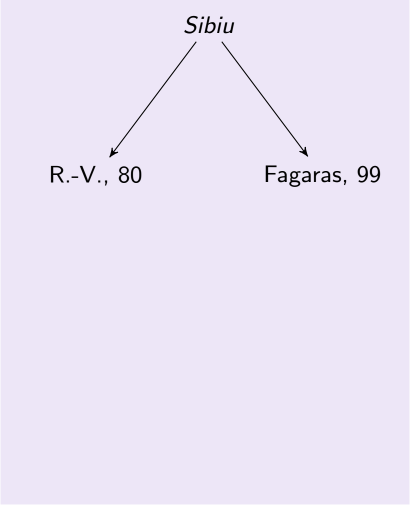


Sibiu

Back to route-finding: Sibiu to Bucharest

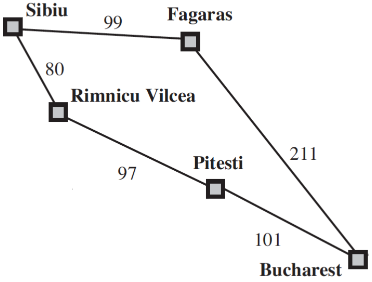
Uniform-cost Search

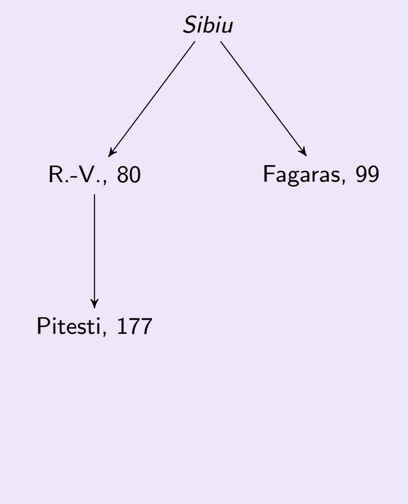




Back to route-finding: Sibiu to Bucharest

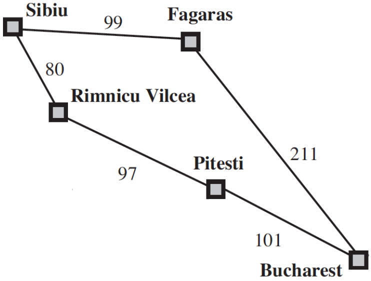
Uniform-cost Search

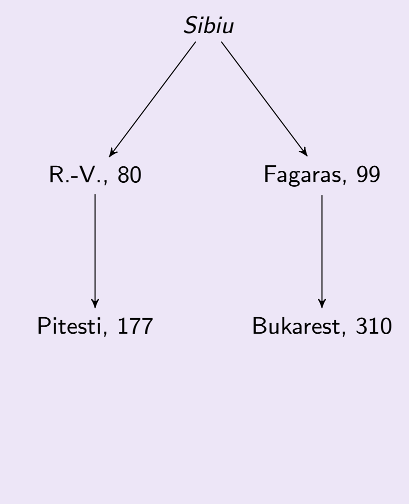




Back to route-finding: Sibiu to Bucharest

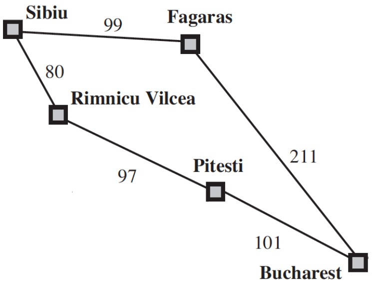
Uniform-cost Search

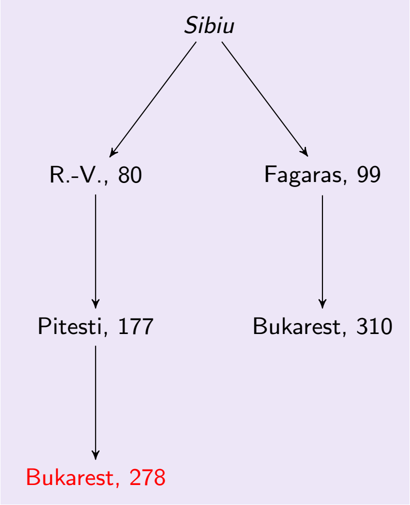




Back to route-finding: Sibiu to Bucharest

Uniform-cost Search





Best-first search

Best-first search is an instance of the general TREE- or GRAPH-SEARCH where node selection for expansion is based on an **evaluation function**, f.

The function f is a cost estimate: node n with lowest $f(n)$ is expanded first.

Best-first graph search the same as UCS but with f for priority queue.

Heuristic functions

The choice of f determines the search strategy.

Heuristic functions

The choice of f determines the search strategy.

Most best-first algorithms include as a part of f a heuristic function,

$h(n) =$ estimated cost of the cheapest path from node n to a goal state.

The choice of f determines the search strategy.

Most best-first algorithms include as a part of f a heuristic function,

$h(n) =$ estimated cost of the cheapest path from node n to a goal state.

Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.

Heuristic functions

The choice of f determines the search strategy.

Most best-first algorithms include as a part of f a heuristic function,

$h(n) =$ estimated cost of the cheapest path from node n to a goal state.

Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.

Heuristics are arbitrary, nonnegative, problem-specific functions, with one constraint: if n is a goal node, then $h(n) = 0$.

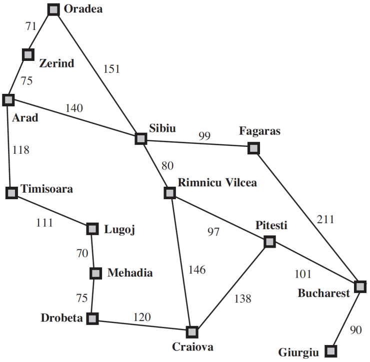
Greedy best-first search

Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.

Greedy best-first search evaluates nodes by using just the heuristic function:

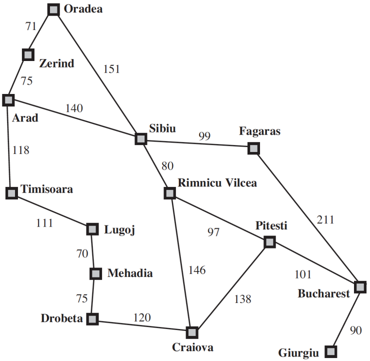
$$f(n) = h(n).$$

Greedy best-first search: Arad to Bucharest



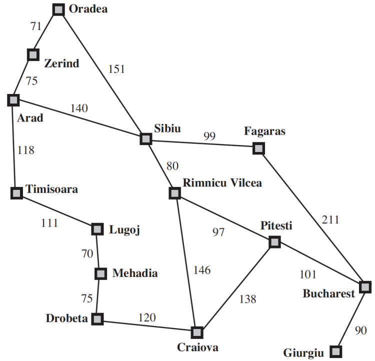
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy best-first search: Arad to Bucharest



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

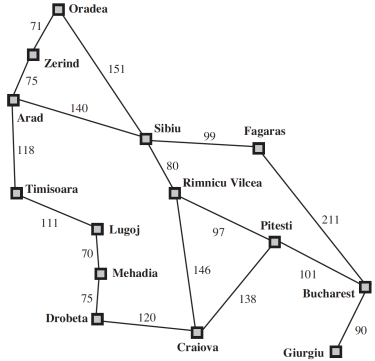
Greedy best-first search: Arad to Bucharest



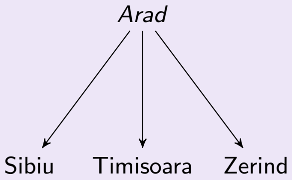
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Arad

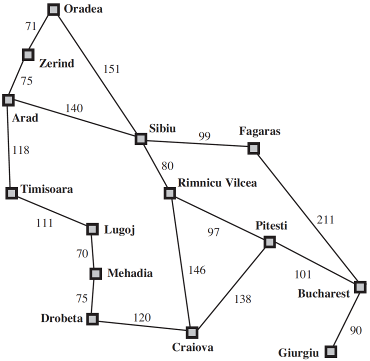
Greedy best-first search: Arad to Bucharest



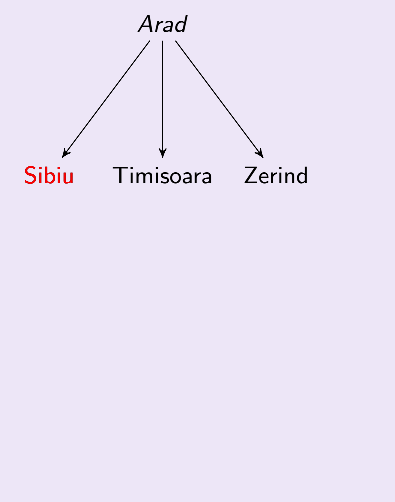
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



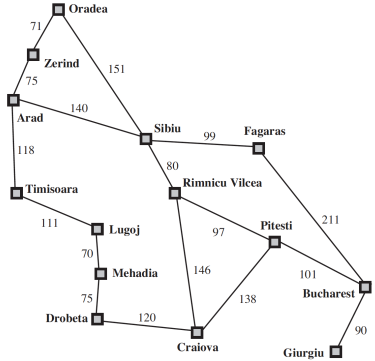
Greedy best-first search: Arad to Bucharest



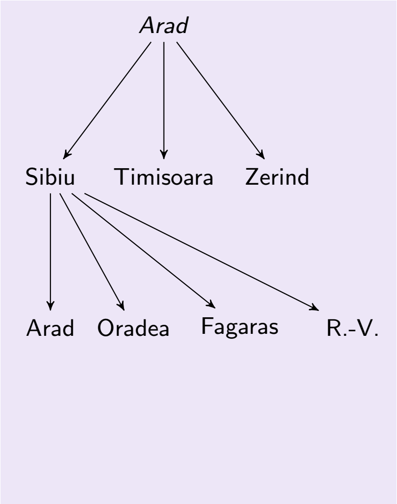
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



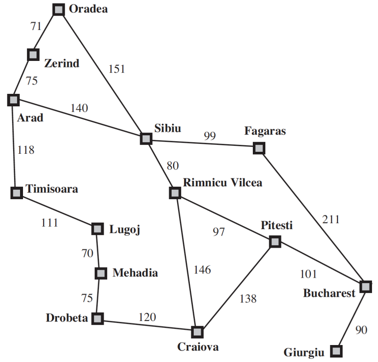
Greedy best-first search: Arad to Bucharest



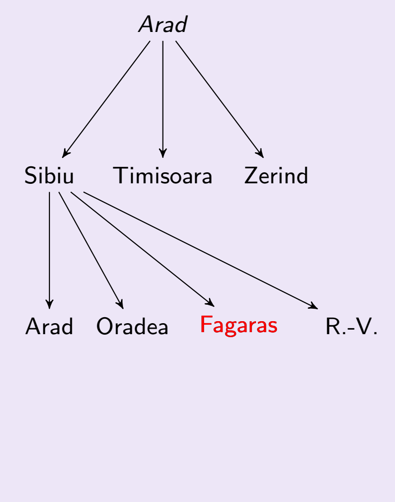
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



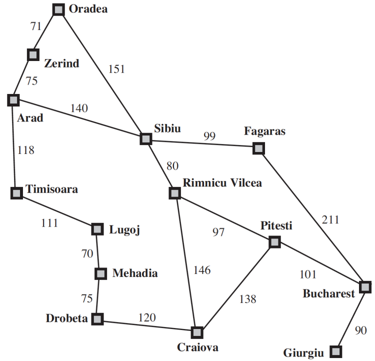
Greedy best-first search: Arad to Bucharest



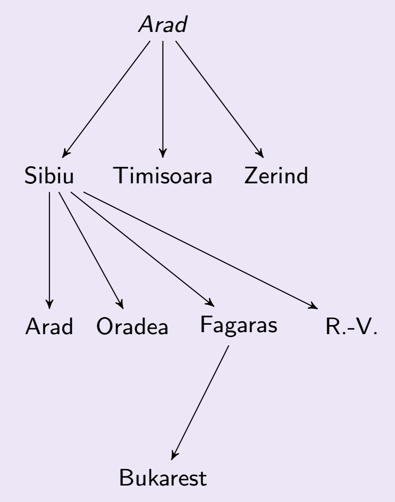
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Greedy best-first search: Arad to Bucharest



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

- It's greedy.
- It's incomplete.
- The complexity (for tree version) is as DFS.

$A^\star$ SEARCH

$A^\star$ evaluates a node n by combining:

- $g(n)$, the cost to reach the node, and
- $h(n)$, the estimated cost to get from the node to the goal:
- $f(n) = g(n) + h(n)$.

A* SEARCH

A* evaluates a node n by combining:

- $g(n)$, the cost to reach the node, and
- $h(n)$, the estimated cost to get from the node to the goal:
- $f(n) = g(n) + h(n)$.

Provided that the heuristic function $h(n)$ satisfies certain conditions,

A* search is both complete and optimal.

$A^\star$ SEARCH

$A^\star$ evaluates a node n by combining:

- $g(n)$, the cost to reach the node, and
- $h(n)$, the estimated cost to get from the node to the goal:
- $f(n) = g(n) + h(n)$.

Provided that the heuristic function $h(n)$ satisfies certain conditions,

$A^\star$ search is both complete and optimal.

The $A^\star$ algorithm is UCS except that it uses $g + h$ instead of g.

$A^\star$ for Super Mario

Video of $A^\star$ playing Super Mario

Designing heuristics

Designing good heuristic function $h(n)$ is in many cases a great art form.

Designing heuristics

Designing good heuristic function $h(n)$ is in many cases a great art form.

Q: What properties should a good heuristic function have?

Designing heuristics

Designing good heuristic function $h(n)$ is in many cases a great art form.

Q: What properties should a good heuristic function have?

A: Estimate cost to goal as precisely as possible; be as cheap as possible to compute (cheaper than computing the actual cost).

ADMISSIBILITY AND OPTIMALITY

Recall: STEP-COST(s, a) is the **cost** of executing a in s.

Optimal cost of a node n:
The minimal cost to achieve a goal from n, denoted $h^*(n)$.

Admissible heuristics: cost of reaching the goal is never overestimated:
the heuristics is **always optimistic**. Formally: $h(n) \leq h^*(n)$ for all nodes n.

Optimal search algorithm: The algorithm always returns an **optimal solution**:
a solution of minimal cost.

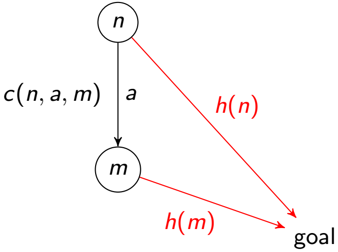
Theorem.
$A^\star$ TREE-SEARCH is optimal when h is admissible.

CONSISTENCY

Let $c(n, a, m)$ denote the STEP-COST of executing action a to go from n to m.

Consistent heuristics: If m is reached by executing a in n then

$$h(n) \leq c(n, a, m) + h(m)$$

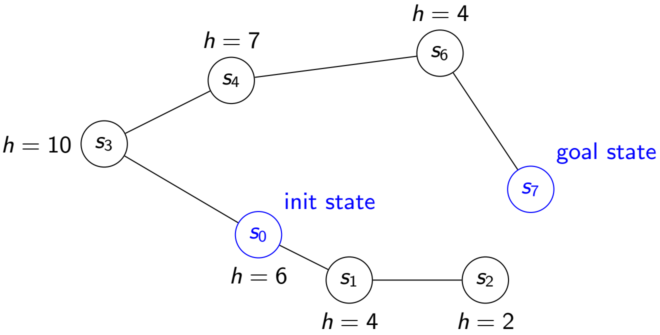


Theorem. Consistency $\Rightarrow$ admissibility.

Theorem. $A^\star$ GRAPH-SEARCH is optimal when h is consistent.
(See R&N, Chapter 3, for the discussion and proof outlines.)

Best-first tree-search vs graph-search

Consider the state space below. All step costs are 1.



Q1 How does greedy best-first tree search behave on the problem?

Q2 How about greedy best-first graph-search?

Q3 And A* tree-search?

Dominating heuristics

A heuristics h_2 is said to **dominate** a heuristics h_1 if

$$h_2(s) \geq h_1(s) \text{ for all } s.$$

If h_2 dominates h_1 and both are admissible, $A^\star$ will never expand more nodes using h_2 than using h_1. So h_2 is better in that sense.

If $h_1, \ldots, h_n$ are admissible heuristics, then so is $h(s) = \max\{h_1(s), \ldots, h_n(s)\}$.

Dominating heuristics

A heuristics h_2 is said to **dominate** a heuristics h_1 if

$$h_2(s) \geq h_1(s) \text{ for all } s.$$

If h_2 dominates h_1 and both are admissible, $A^\star$ will never expand more nodes using h_2 than using h_1. So h_2 is better in that sense.

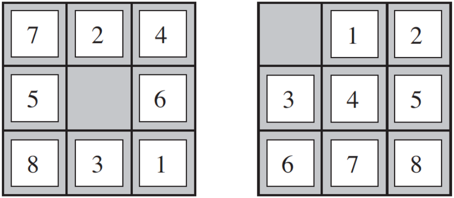
If $h_1, \ldots, h_n$ are admissible heuristics, then so is $h(s) = \max\{h_1(s), \ldots, h_n(s)\}$.

It is always better to use h as heuristics than either of the h_i,
unless the penalty in computation time of h is too high.

Often heuristics are generated via **relaxed problems**.

Relaxed problem:
A simplified version of a problem with fewer restrictions.
A solution to the original problem is also a solution to the relaxed problem.

Example



Start State Goal State

Relaxing the 8-puzzle:

1. A tile can move to any adjacent square.
2. A tile can move to any square.

Generating heuristics via relaxed problems

Take a problem P and let $h_P^*(n)$ be the optimal cost to get to goal from n in P.

Given any problem P and relaxation P', a heuristics h for P can be defined by:

$$h(n) = h_{P'}^*(n).$$

In words: The **estimated cost** of a solution to the **real problem** is taken to be the **actual cost** of a solution to the **relaxed problem**.

- Q1 Why is h defined above admissible?
- Q2 Why is h defined above consistent?
- Q3 Which heuristics do we get from the sliding puzzle relaxations of the previous slide (1. Move to any adjacent square; 2. Move to any square)?
- Q4 Does one of the heuristics from the previous question dominate the other?
- Q5 Given P', how do we calculate $h_{P'}^*$?

THE END OF LECTURE 3