UNIVERSITY OF BIRMINGHAM

AI1 & AIML - DFS, Variations and Informed Search

Dr Leonardo Stella

Aims of the Session

This session aims to help you:

- Explain the steps involved in Depth-First Search and its variations. Be able to apply these algorithms to solve search problems

- Understand the concept of a heuristic function

- Analyse the performance of A* and apply the algorithm to solve search problems

Overview

- **Recap: Search Problem Formulation and BFS**
- Depth-First Search and Variations
- Informed Search
- A* Algorithm

Search Problem Formulation

- Activity. Explain what each of the following terms mean:
  - **Observable**
  - **Discrete**
  - **Known**
  - **Deterministic**

Search Problem Formulation

- Activity. Explain what each of the following terms mean:
  - **Observable**, i.e., the agent is able to know the current state
  - **Discrete**, i.e., there are only finitely many actions at any state
  - **Known**, i.e., the agent can determine which states are reached by which action
  - **Deterministic**, i.e., each action has exactly one outcome

Search Problem Formulation

- Activity. Explain what the main components of a search problem are:

Search Problem Formulation

- Activity. Explain what the main components of a search problem are:
  - **Initial state**, **Actions**, **Transition model**, **Goal test** and **Path cost**

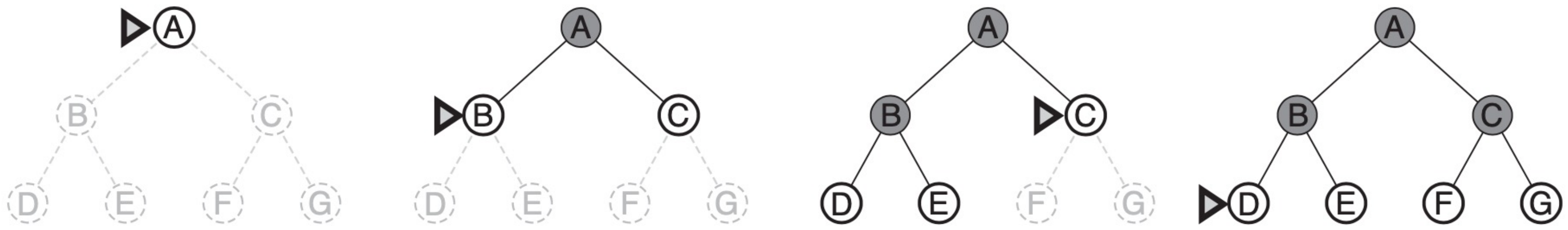
Search Problem Formulation

- Activity. Explain what the main components of a search problem are:
  - **Initial state**, **Actions**, **Transition model**, **Goal test** and **Path cost**

- The first three components together define the **state space** of the problem, in the form of a directed graph or network. A **path** in the state space is a sequence of states connected by a sequence of actions

Breadth-First Search

- Breadth-First Search steps (see Fig. 3.12):
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



BFS - Performance

Let us evaluate the performance of BFS:

- **Completeness**: if the goal node is at some finite depth d, then the BFS algorithm is complete as it will find it (given that b is finite)
- **Optimality**: BFS is optimal if the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)
- **Time complexity**: $O(b^d)$, assuming a uniform tree where each node has b successors, we generate $b + b^2 + \cdots + b^d = O(b^d)$
- **Space complexity**: $O(b^d)$, if we store all expanded nodes, we have $O(b^{d-1})$ explored nodes in memory and $O(b^d)$ in the frontier

Overview

- **Recap: Search Problem Formulation and BFS**
- **Depth-First Search and Variations**
- Informed Search
- A* Algorithm

Depth-First Search

- Depth-First search is another common search strategy:
  - The root node is expanded first
  - Then, the first (or one at random) successor of the root node is expanded
  - Then, the deepest node in the current frontier is expanded

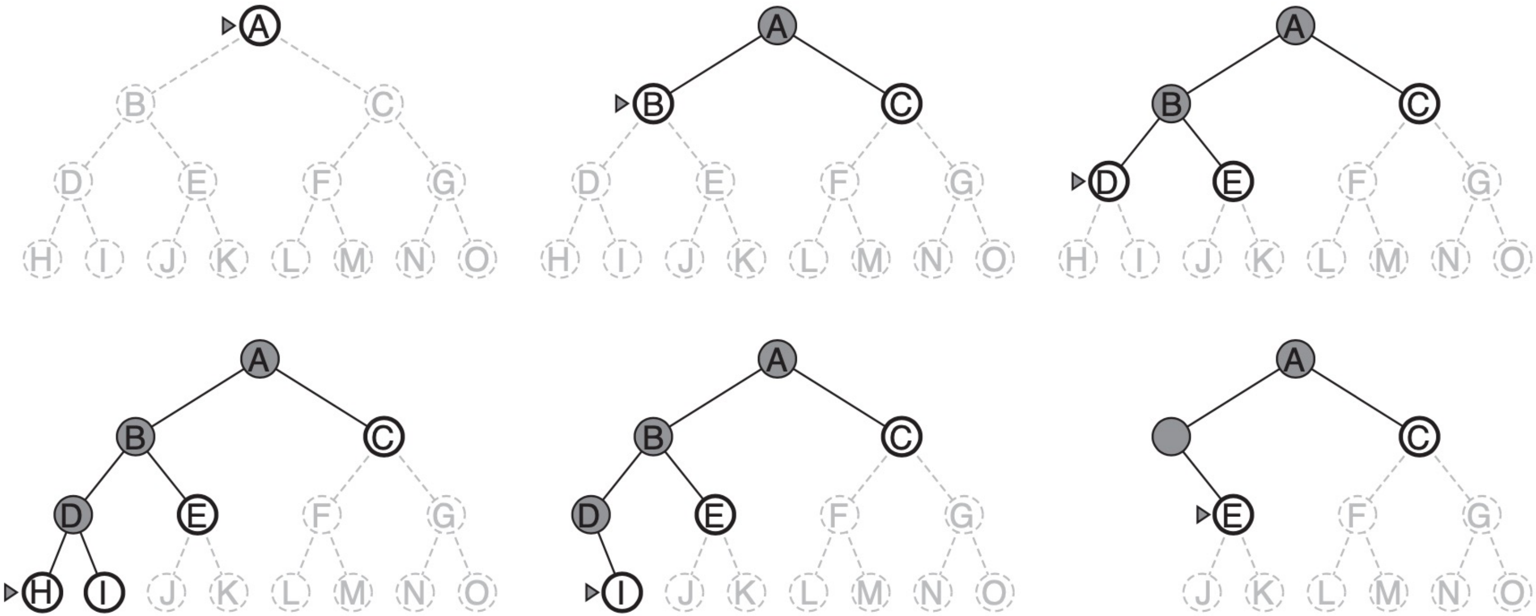
- This is equivalent to expanding the deepest unexpanded node in the frontier; simply use a **stack** (**LIFO**) for expansion

- Basically, the most recently generated node is chosen for expansion

Depth-First Search

- Depth-First Search steps:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

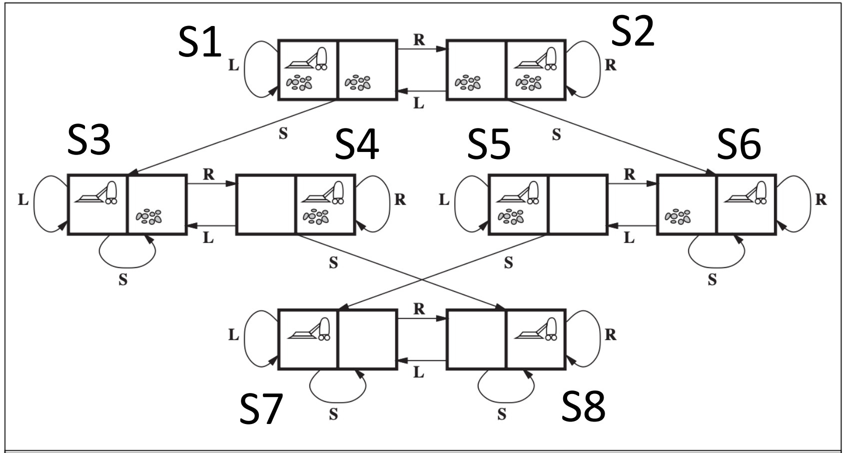
Depth-First Search (see Fig. 3.16)



Depth-First Search

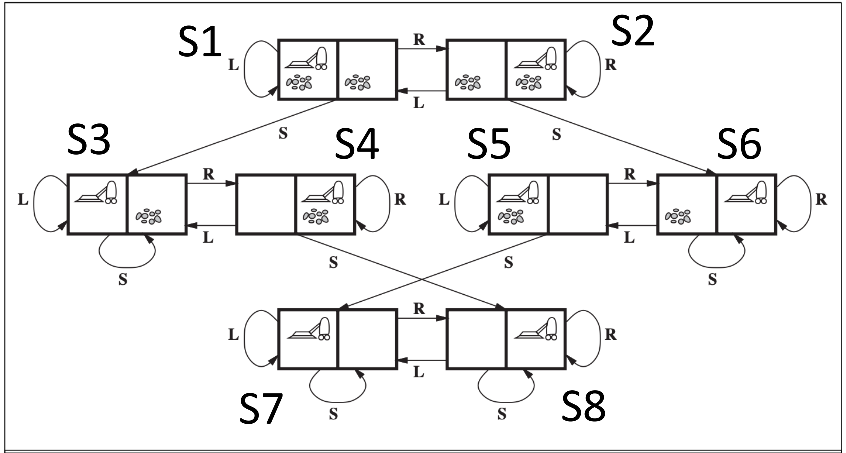
- Activity. Let us apply DFS to the Vacuum World example from the book:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

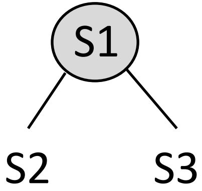
S1



Depth-First Search

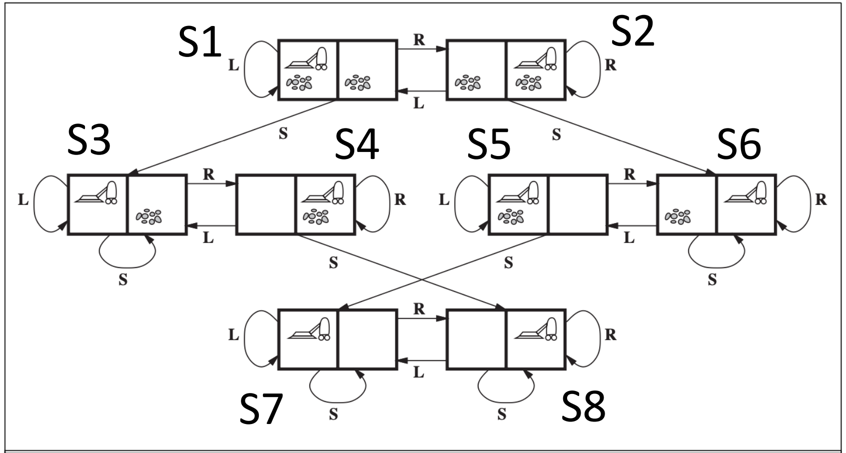
- Activity. Let us apply DFS to the Vacuum World example from the book:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

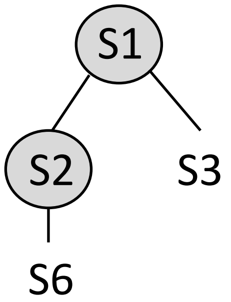




Depth-First Search

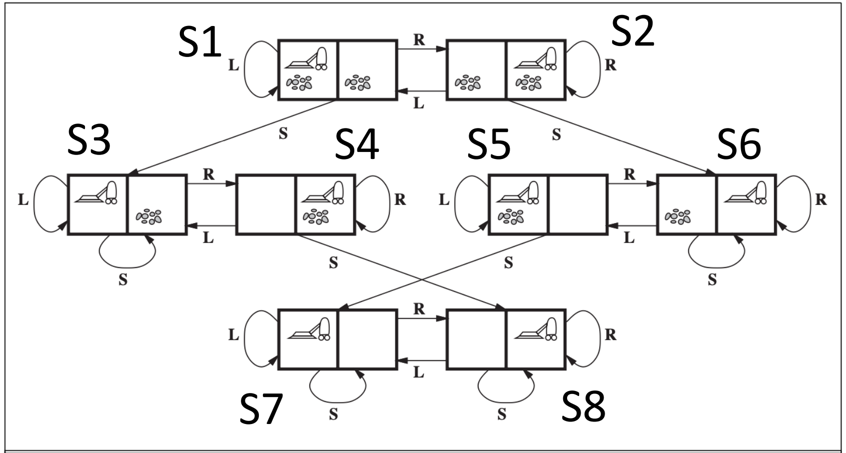
- Activity. Let us apply DFS to the Vacuum World example from the book:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

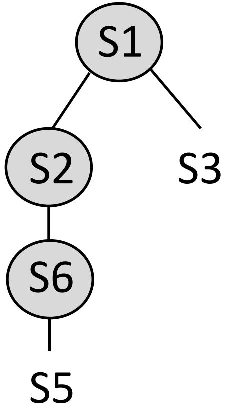




Depth-First Search

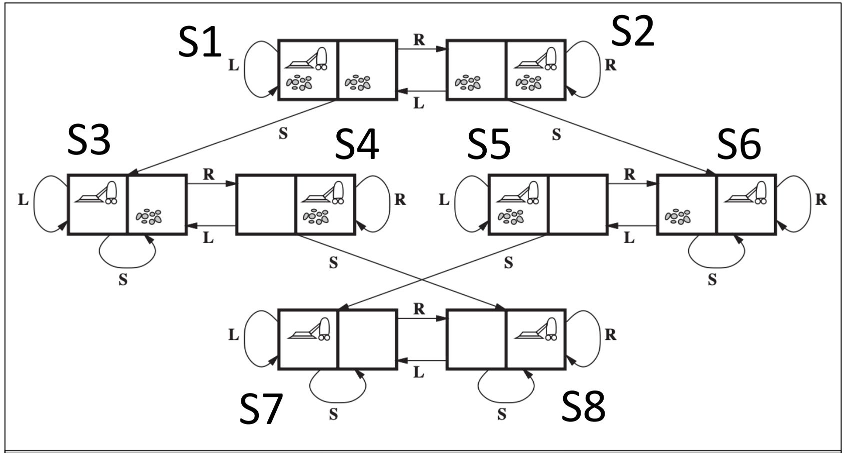
- Activity. Let us apply DFS to the Vacuum World example from the book:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

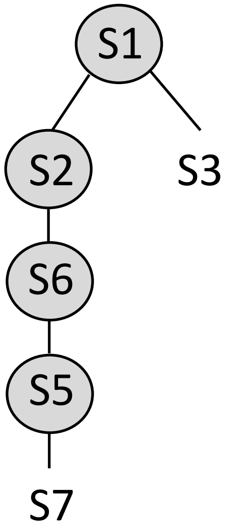




Depth-First Search

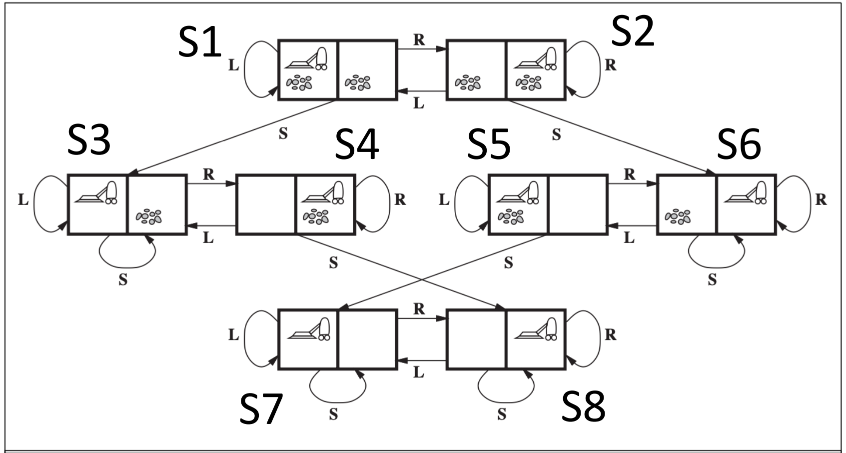
- Activity. Let us apply DFS to the Vacuum World example from the book:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

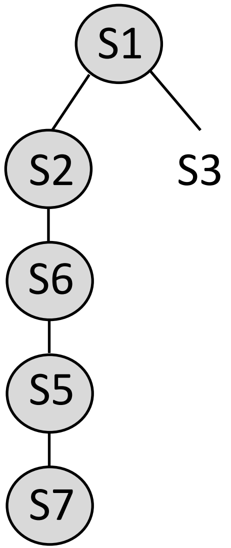




Depth-First Search

- Activity. Let us apply DFS to the Vacuum World example from the book:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited





Depth-First Search

- Solution:

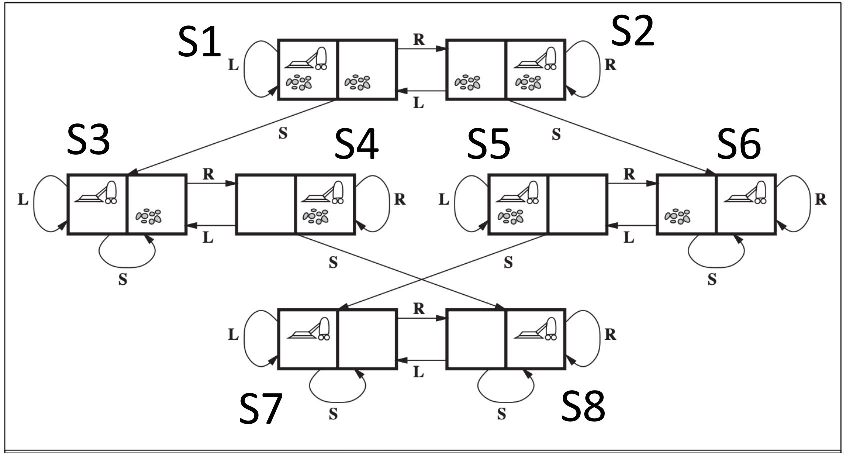
R, S, L, S

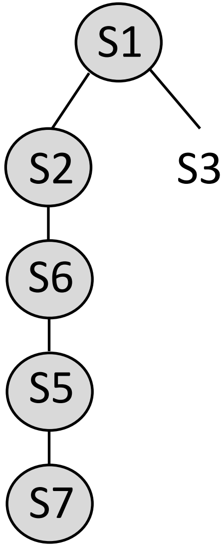
- Cost of the solution:

1 + 1 + 1 + 1 = 4

- Order of nodes visited

S1, S2, S6, S5, S7





DFS - Performance

Activity. Let us evaluate the performance of DFS:

- **Completeness**:

- **Optimality**:

- **Time complexity**:

- **Space complexity**:

DFS - Performance

Activity. Let us evaluate the performance of DFS:

- **Completeness**: DFS is not complete if the search space is infinite or if we do not check infinite loops; it is complete if the search space is finite
- **Optimality**: DFS is not optimal as it can expand a left subtree when the goal node is in the first level of the right subtree
- **Time complexity**:
- **Space complexity**:

DFS - Performance

Activity. Let us evaluate the performance of DFS:

- **Completeness**: DFS is not complete if the search space is infinite or if we do not check infinite loops; it is complete if the search space is finite
- **Optimality**: DFS is not optimal as it can expand a left subtree when the goal node is in the first level of the right subtree
- **Time complexity**: $O(b^m)$, as it depends on the maximum length of the path in the search space (in general m can be much larger than d)
- **Space complexity**: $O(b^m)$, as we store all the nodes from each path from the root node to the leaf node

Depth-First Search - Variations

- Depth-First Search comes with several issues
  - Not optimal
  - High time complexity
  - High space complexity

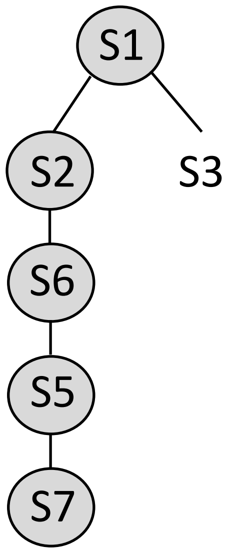
- DFS with less memory usage (saving space complexity)

- Depth-Limited Search

Depth-First Search – Less Memory Usage

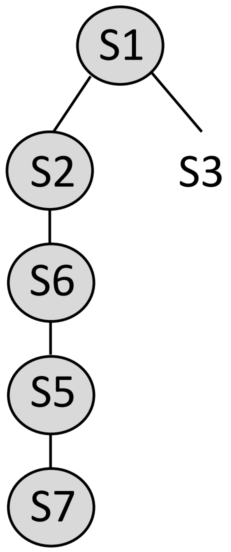
- Imagine we have a tree similar the one in the example

- Now, S7 is not a goal node and it has no children



Depth-First Search – Less Memory Usage

- Imagine we have a tree similar the one in the example
- Now, S7 is not a goal node and it has no children
- The next step of the algorithm would be to expand S3



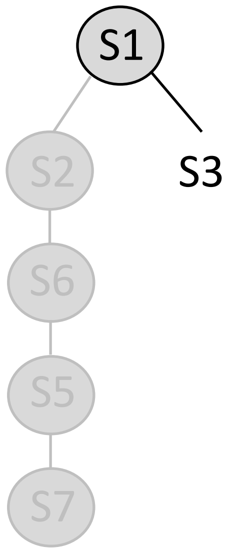
Depth-First Search – Less Memory Usage

- Imagine we have a tree similar the one in the example

- Now, S7 is not a goal node and it has no children

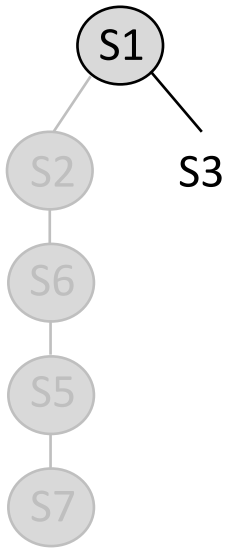
- The next step of the algorithm would be to expand S3

- Since we explored all the left subtree, we can remove it from memory



Depth-First Search – Less Memory Usage

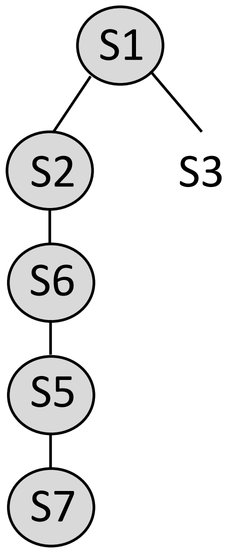
- This would reduce the space complexity to $O(bm)$
- We need to store a single path along with the siblings for each node on the path
- Recall that b is the branching factor and m is the maximum depth of the search tree



Depth-Limited Search

- The issue related to depth-first search in infinite state spaces can be mitigated by providing a depth limit ℓ

- This approach is called **depth-limited search**

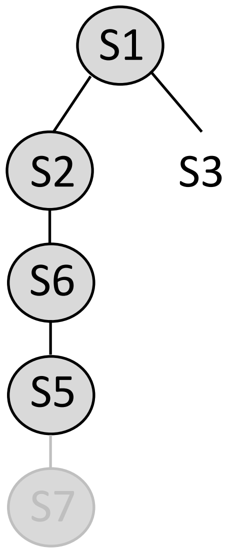


Depth-Limited Search

- The issue related to depth-first search in infinite state spaces can be mitigated by providing a depth limit ℓ

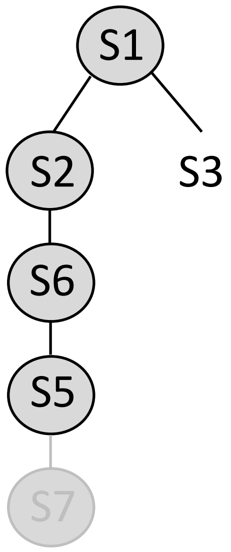
- This approach is called **depth-limited search**

- For $\ell = 3$, we would have



Depth-Limited Search

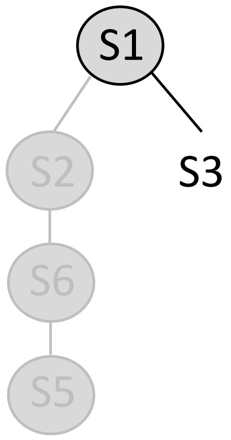
- This adds an additional source of incompleteness if we choose $\ell < d$, namely, the shallowest goal is beyond the depth limit
- This approach is nonoptimal also in the case $\ell > d$
- Time complexity is $O(b^{\ell})$



Depth-Limited Search – Less Memory Usage

- As before, we can remove the explored paths from memory after we have reached the depth limit ℓ

- Space complexity is $O(b\ell)$



Comparing Uninformed Search Strategies

Criterion / Algorithm	Breadth-First	Depth-First	Depth-First (less memory)	Depth-Limited (less memory)
Completeness	Yes*	Yes***	Yes***	Yes, if $\ell \geq d$
Optimality	Yes**	No	No	No
Time	$O(b^d)$	$O(b^m)$	$O(b^m)$	$O(b^\ell)$
Space	$O(b^d)$	$O(b^m)$	$O(bm)$	$O(b\ell)$

* If b is finite

** If the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)

*** If the search space is finite (also, loopy paths are removed)

Summary

- Depth-First Search is a search algorithm that expands the nodes in the frontier starting from the deepest, similar to a stack (LIFO)
- This algorithm is complete (for finite search space), but not optimal; also, it has high time complexity and space complexity $O(b^m)$
- Depth-First Search can be improved in terms of its time and space complexity
- Depth-First Search with less memory usage only keeps in memory the current path and the siblings of the nodes. Depth-Limited Search is another variation, where a depth limit is specified; this adds an additional source of incompleteness

Overview

- **Recap: Search Problem Formulation and BFS**
- **Depth-First Search and Variations**
- **Informed Search**
- A* Algorithm

Informed Search Strategies

- **Informed search** strategies use problem-specific knowledge beyond the definition of the problem itself

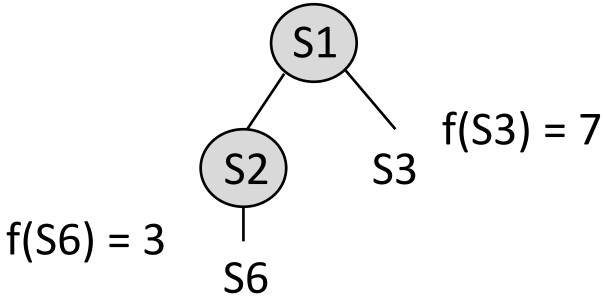
- Informed search strategies can find solutions more efficiently compared to uninformed search

Informed Search Strategies

- The general approach, called **best-first search**, is to determine which node to expand based on an **evaluation function**

$$f(n): node \rightarrow cost\ estimate$$

- This function acts as a cost estimate: the node with the lowest cost is the one that is expanded next



Informed Search Strategies - Heuristics

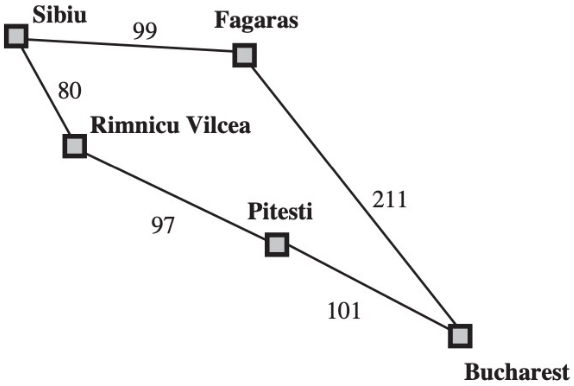
- The evaluation function $f(n)$ for most best-first search algorithms includes a **heuristic function** as a component:
  $h(n)$ = *estimated cost of the cheapest path from node* n *to a goal node*

- Heuristic functions are the most common form in which new knowledge is given to the search algorithm. If n is a goal node, then $h(n) = 0$

- A heuristic can be a rule of thumb, common knowledge; it is quick to compute, but not guaranteed to work (nor to yield optimal solutions)

Informed Search Strategies - Heuristics

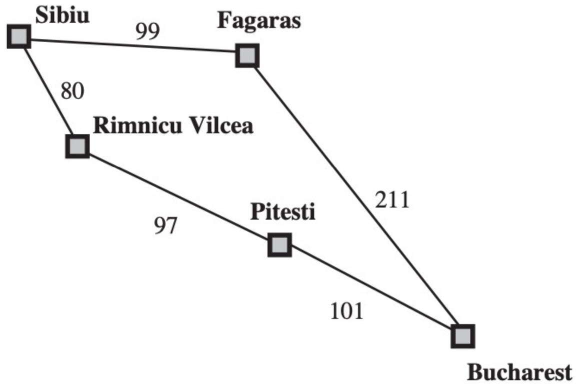
- Consider the problem to find the shortest path to Bucharest in Romania



- We can use the straight-line distance heuristic, denoted by h_{SLD}
- This is a useful heuristic as it is correlated with actual road distances

Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania

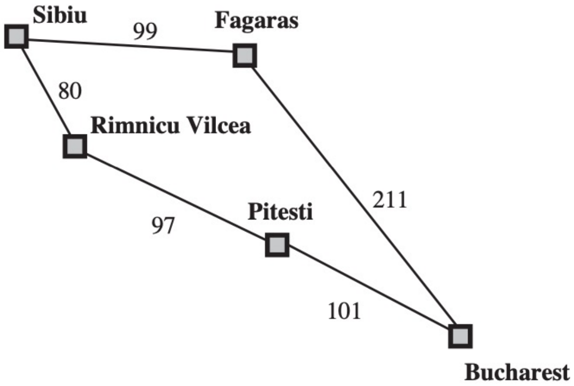


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- The straight-line distances h_{SLD} are shown in the table above
- For example, the SLD from Sibiu would be 253

Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania

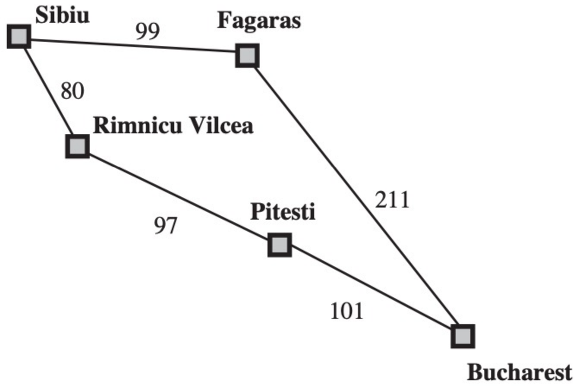


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- If we use $f(n) = h_{SLD}(n)$, then from Sibiu we expand Fagaras
- This is because Fagaras has SLD 176, while Rimnicu Vilcea 193

Informed Search Strategies - Heuristics

- Consider the problem to find the shortest path to Bucharest in Romania



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

- When $f(n) = h(n)$, we call this strategy **Greedy Best-First Search**

Overview

- Recap: Search Problem Formulation and BFS
- Depth-First Search and Variations
- Informed Search
- **A* Algorithm**

A* Search

- The most widely known informed search strategy is **A***

- This search strategy evaluates nodes using the following cost function

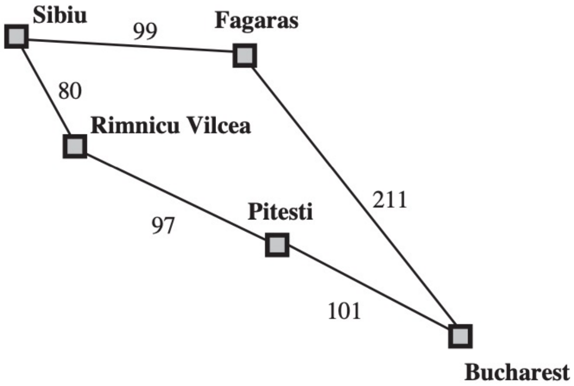
$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost to reach the node and $h(n)$ is the heuristic from the node to the goal

- This is equivalent to *the cost of the cheapest solution through node* n

A* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



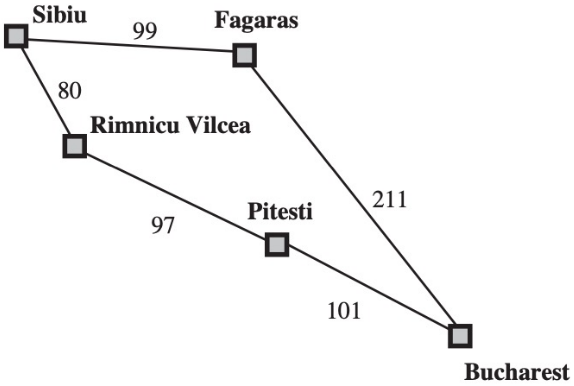
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

- Activity. Consider the above problem where Sibiu is the initial state. Calculate $f(n)$ to choose which node to expand, starting with Fagaras

$$f(Fagaras) = g(Fagaras) + h(Fagaras)$$

A* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



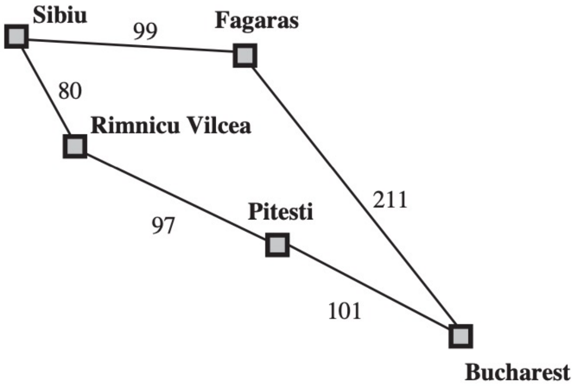
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- Activity. Consider the above problem where Sibiu is the initial state. Calculate $f(n)$ to choose which node to expand, starting with Fagaras

$$f(Fagaras) = 99 + 176 = 275$$

A* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



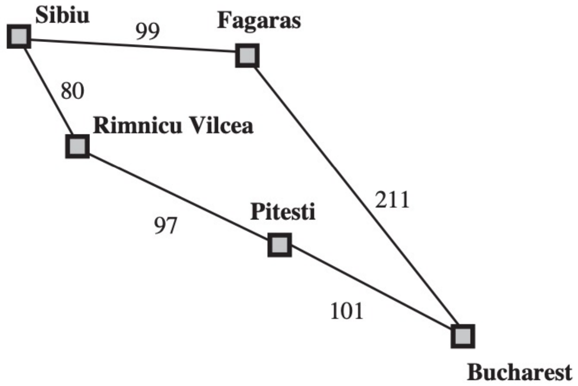
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- Activity. Consider the above problem where Sibiu is the initial state. Calculate $f(n)$ to choose which node to expand:

$$f(Rimnicu\ Vilcea) =$$

A* Search - Example

- Consider the problem to find the shortest path to Bucharest in Romania



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

- Activity. Consider the above problem where Sibiu is the initial state. Calculate $f(n)$ to choose which node to expand:

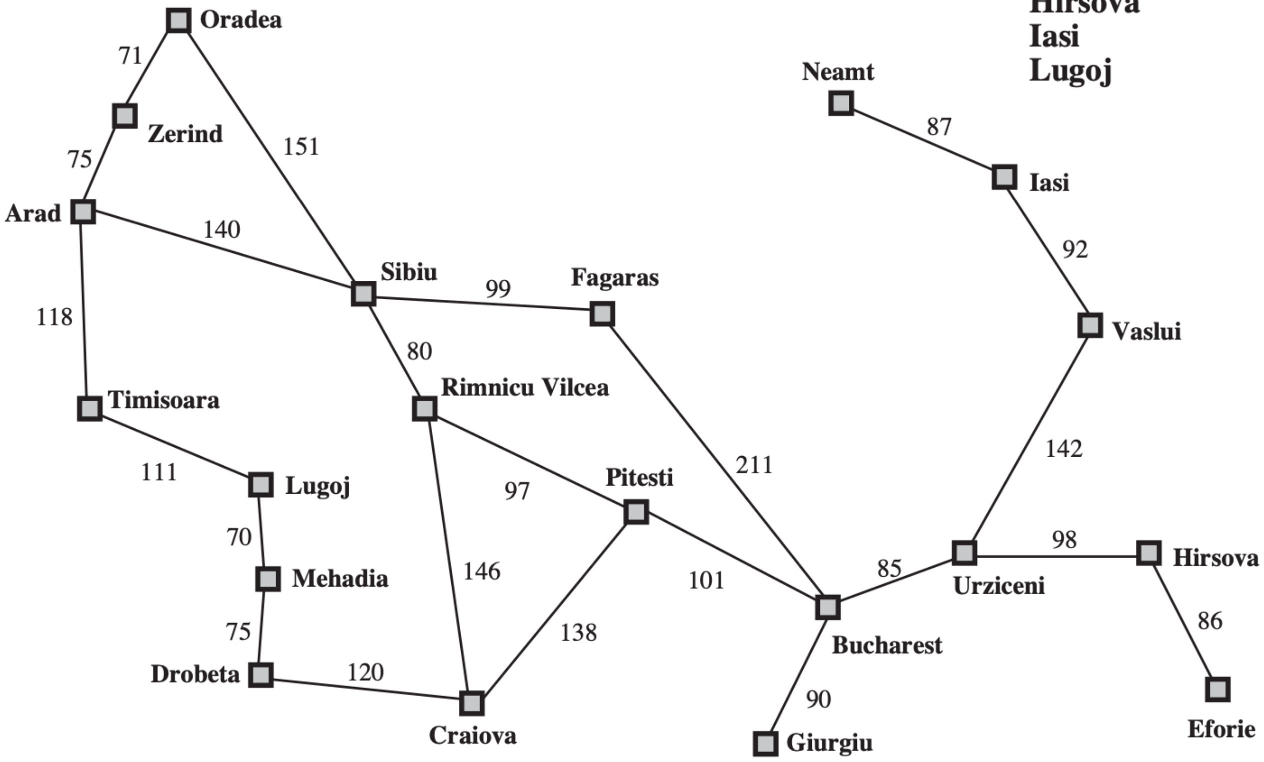
$$f(Rimnicu\ Vilcea) = 80 + 193 = 273$$

A* Search - Algorithm

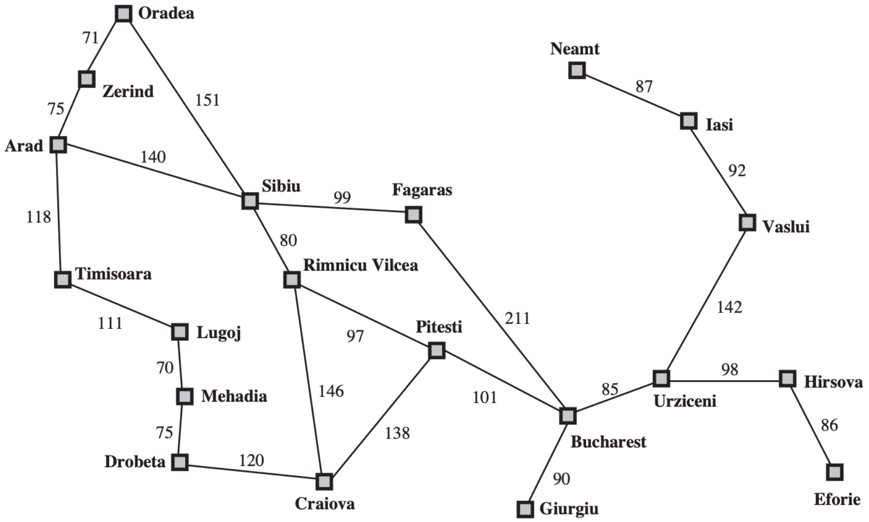
- A* search algorithm:
  - **Expand** the node in the frontier with smallest cost $f(n) = g(n) + h(n)$
  - **Do not add** children in the frontier if the node is in the list of visited nodes (to avoid loopy paths)
  - If the state of a given child is in the frontier
    - If the frontier node has a larger $g(n)$, place the child into the frontier and remove the node with larger $g(n)$ from the frontier
  - **Stop** when a goal node is visited

A* Search

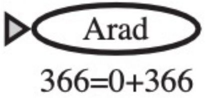
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



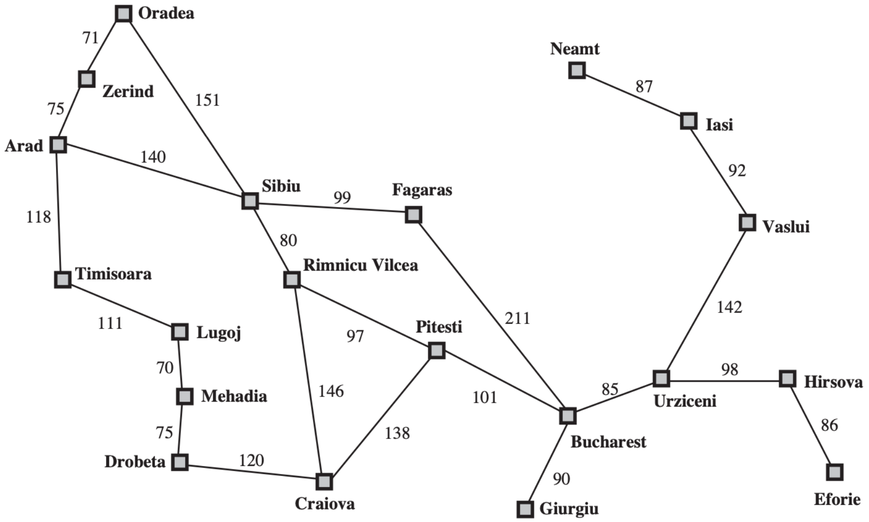
A* Search



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

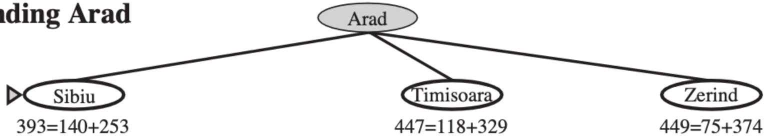


A* Search



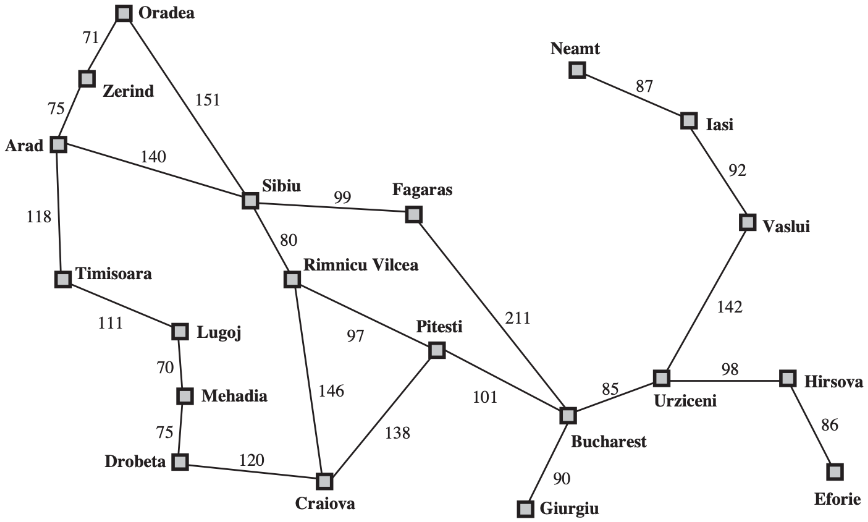
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

After expanding Arad

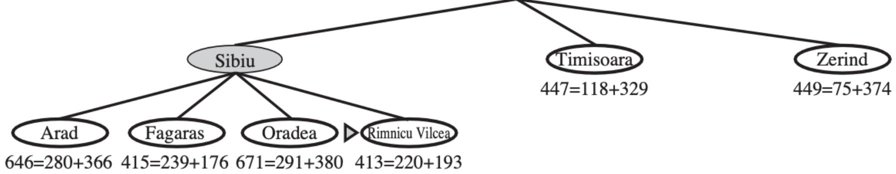


A* Search

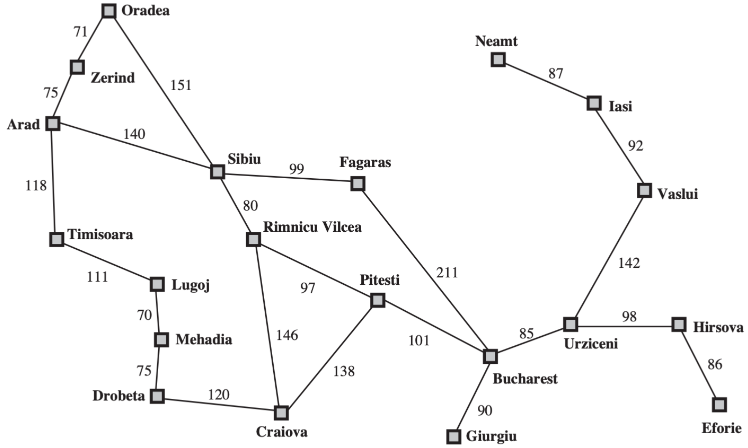
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



(c) After expanding Sibiu

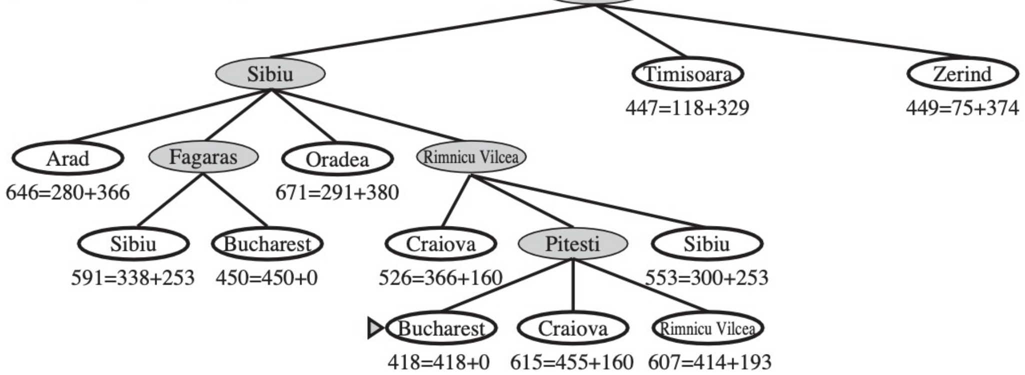


A* Search



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

(f) After expanding Pitesti



A* Search - Completeness and Optimality

- The A* search is **complete** and **optimal** if $h(n)$ is consistent

A* Search - Completeness and Optimality

- The A* search is **complete** and **optimal** if $h(n)$ is consistent

- Definition: a heuristic is said to be consistent (or monotone), if the estimate is always no greater than the estimated distance from any neighbouring node to the goal, plus the cost of reaching that neighbour.

$$h(n) \leq cost(n, n') + h(n')$$

A* Search - Time and Space Complexity

- The number of states for the A* search is **exponential** in the length of the solution, namely, for constant step costs: $O(b^{\epsilon d})$
- When h^* is the actual cost from root node to goal node, $\epsilon = \frac{(h^* - h)}{h^*}$ is the relative error

A* Search - Time and Space Complexity

- The number of states for the A* search is **exponential** in the length of the solution, namely, for constant step costs: $O(b^{\epsilon d})$
- When h^* is the actual cost from root node to goal node, $\epsilon = \frac{(h^* - h)}{h^*}$ is the relative error
- Space is the main issue with A*, as it keeps all generated nodes in memory, therefore A* is not suitable for many large-scale problems

A* Search - Summary

Let us summarise the performance of the A* search algorithm

- **Completeness**: if the heuristic $h(n)$ is consistent, then the A* algorithm is complete
- **Optimality**: if the heuristic $h(n)$ is consistent, A* is optimal

A* Search - Summary

Let us summarise the performance of the A* search algorithm

- **Completeness**: if the heuristic $h(n)$ is consistent, then the A* algorithm is complete
- **Optimality**: if the heuristic $h(n)$ is consistent, A* is optimal
- **Time complexity**: $O(b^{\epsilon d})$, where ϵ is the relative error of the heuristic

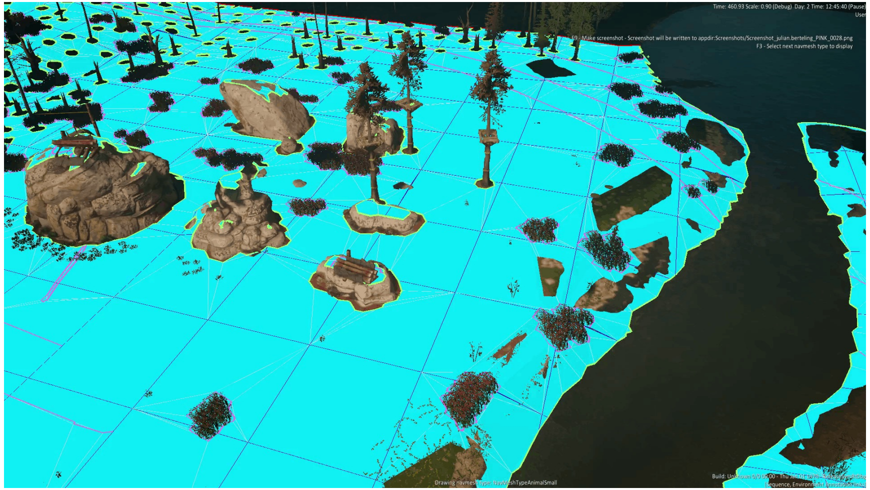
A* Search - Summary

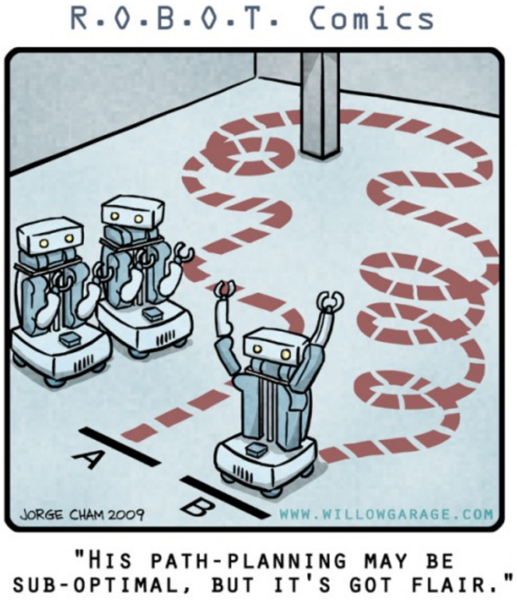
Let us summarise the performance of the A* search algorithm

- **Completeness**: if the heuristic $h(n)$ is consistent, then the A* algorithm is complete
- **Optimality**: if the heuristic $h(n)$ is consistent, A* is optimal
- **Time complexity**: $O(b^{\epsilon d})$, where ϵ is the relative error of the heuristic
- **Space complexity**: $O(b^d)$, since we keep in memory all expanded nodes and all nodes in the frontier

A* - Applications

- A* has a large number of applications
- In practice, the most common ones are in games and in robotics





Summary

- A* is complete and optimal, given a consistent heuristic

- However, A* has typically high time/space complexity, regardless of the heuristic chosen

- Heuristics have a considerable impact on the performance of informed search algorithms, and they can drastically reduce the time and space complexity in comparison to uninformed search algorithms

Aims of the Session

You should now be able to:

- Explain the steps involved in Depth-First Search and its variations. Be able to apply these algorithms to solve search problems
- Understand the concept of a heuristic function
- Analyse the performance of A* and apply the algorithm to solve search problems

References

- Russell, A. S., and Norvig, P. (2010), *Artificial Intelligence A Modern Approach*, 3rd Edition. Prentice Hall.
  - Chapter 3 – Solving Problems by Searching (Section 3.4.3 up to 3.4.4, Section 3.4.7, Section 3.5 up to 3.5.2)