Universidade do Minho
Escola de Engenharia
Departamento de Informática

PROBLEM-SOLVING AND SEARCH METHODS

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial
2025/26

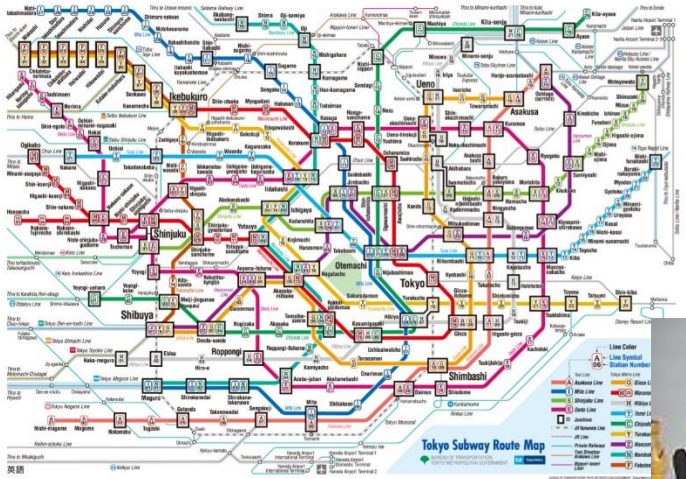
ISLab
Synthetic Intelligence Lab

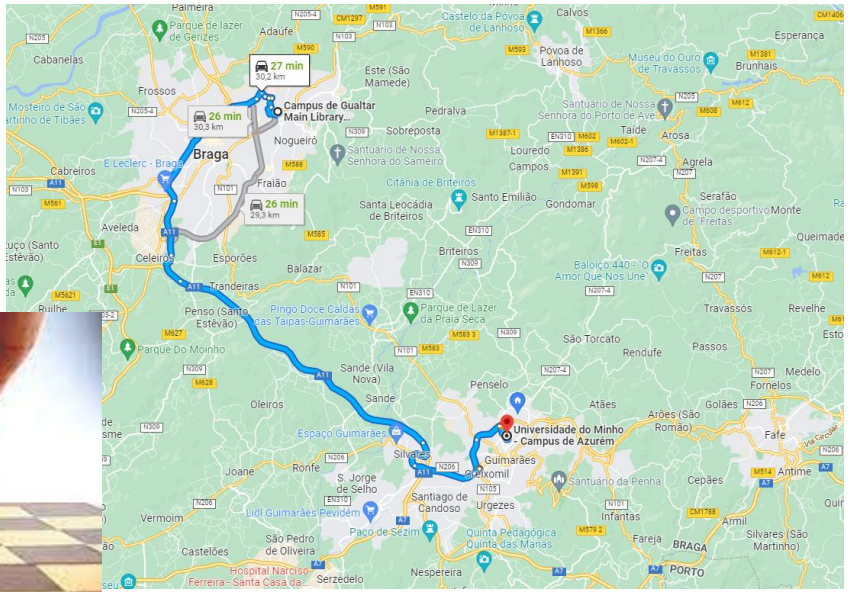
Agenda

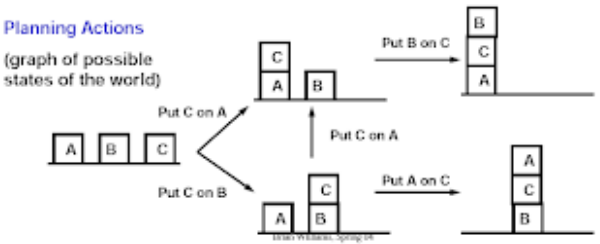
- Problem formulation
- Problem solving
- Types of problems
- Examples of real-world problems
- Searching for solutions
- Search strategies
  - Uninformed (blind) search
  - Informed Search (heuristic
- Beyond Classical Demand

ISLab
Synthetic Intelligence Lab

Search







Introduction

- **Agent:** is something that perceives the environment through sensors and acts on the environment through actuators

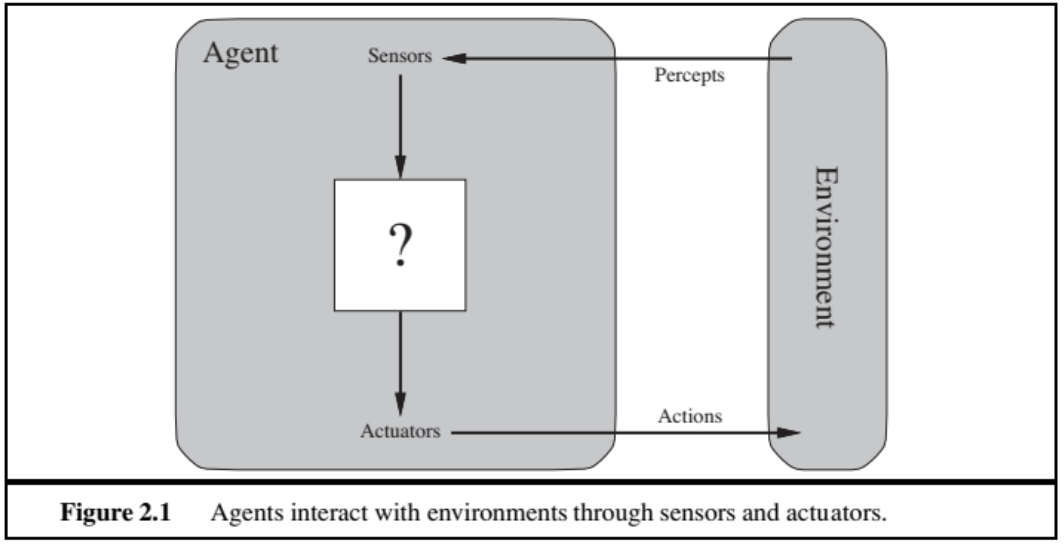


Figure 2.1 Agents interact with environments through sensors and actuators.

Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Introduction

- The term perception refers to the agent's perceptions at a given instant. A sequence of the agent's perceptions is the total history of everything the agent has perceived.
- The agent's behaviour is described mathematically by the agent function, which maps any sequence of perceptions into an action.
- **Agent : architecture* + program****

* computing device with sensors and actuators

** materialisation of the agent's role

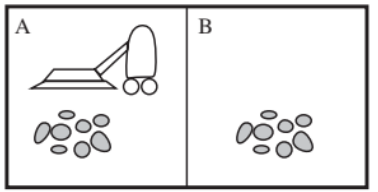


Figure 2.2 A vacuum-cleaner world with just two locations.

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Introduction

- How an agent can act, setting objectives and considering possible sequences of actions to achieve these objectives?

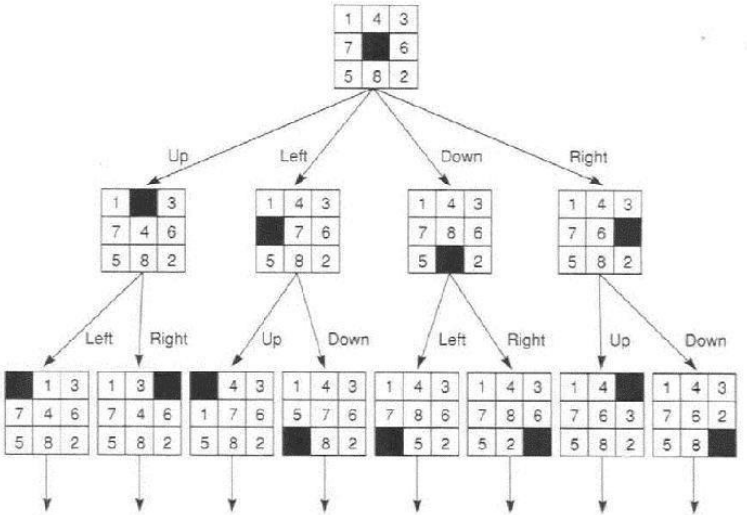
- The implementation of a simple problem-solving agent first needs to formulate an **objective** and a **problem**, look for the sequence of actions that solve the problem, executing them one at a time. When complete, it will formulate another objective and execute it again.

- In this context, problem solving is seen as formulating a problem as a **search problem**.

ISLab
Synthetic Intelligence Lab

Problem Formulation

- "Problem Solving Agent": It seeks to find the sequence of actions that leads to a desirable state!
- Problem formulation:
  - What actions are possible (what effect do they have on the state of the world)?
  - What are the possible states? (how to represent them?)
  - How to evaluate states
- Search Problem:
  - Solution: action sequence
- The final stage is execution!
- Formulate → Search → Execcute



https://cis.temple.edu/~wangp/3203-AI/Lecture/Search-1.htm

Problem Formulation

- Many computer science problems can be formulated as :
  - A set S of STATES (possibly infinite)
  - An INITIAL state $s \in S$
  - A TRANSITION relation T along this state space
  - A set of END states (goals): THE: $O \in S$

- A problem can be formally defined in five components :
  1. State Representation
  2. Initial State (Current)
  3. Objective State (defines the desired states)
  4. Operators (Name, Preconditions, Effects, Cost)
  5. Solution Cost

ISLab
Synthetic Intelligence Lab

Problem Solving

- The problem can be solved by finding a path between the initial state and an objective state
- In summary, problem formulation involves deciding which actions and states to consider given an objective

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

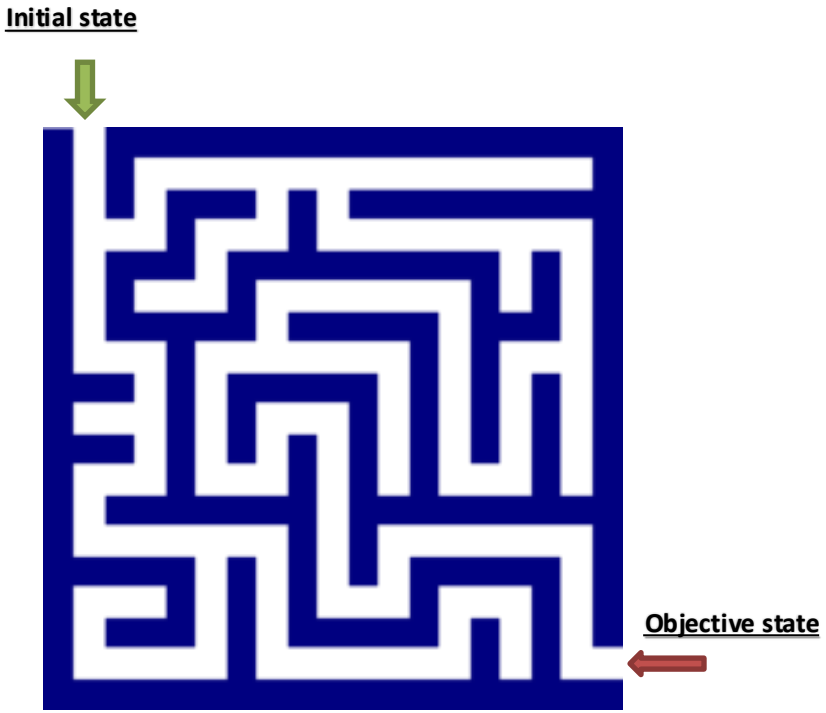
Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

ISLab
Synthetic Intelligence Lab

Problem Formulation
Components of a Search Problem

- Initial State
- Actions
- Transition model
  - What state results from executing a certain action in a certain state?
- Objective state
- Path cost
  - Suppose it's a sum of the non-negative costs of each stage

- The ideal solution is the sequence of actions that provides the lowest path cost to reach the goal



ISLab
Synthetic Intelligence Lab

Problem Formulation

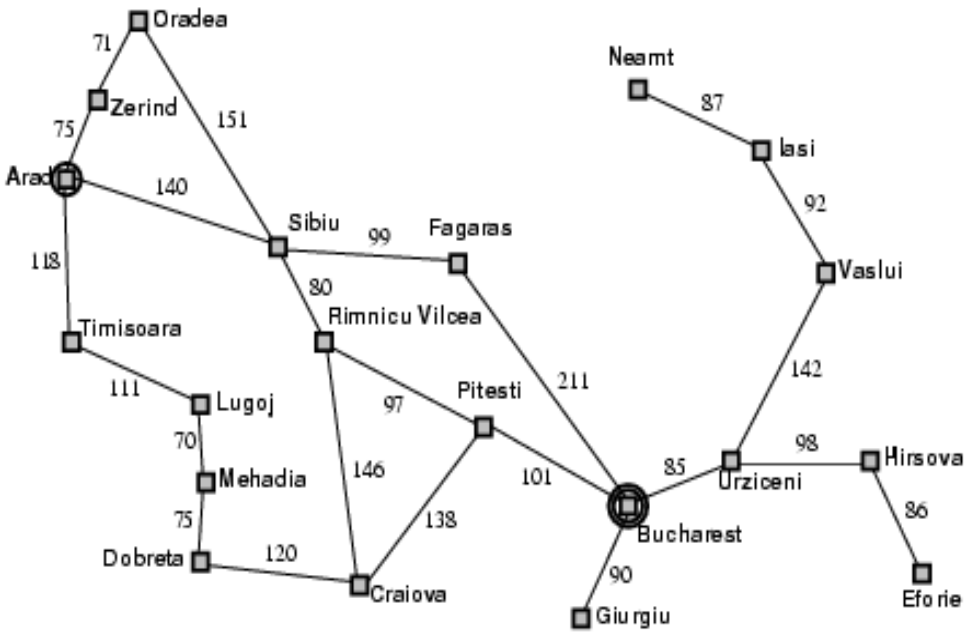
Example: Travelling from Arad to Bucharest

Initial State: Arad;
Objective State: Bucharest
States: the different cities
Operations: driving between cities

Solution:

- A path*: sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
  - The shortest route
  - The fastest way
  - The "greenest" way

* The cost of the solution is determined by the cost of each path, which reflects a measure of the solution's performance (in this case the cost will be the distance between cities).



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

ISLab
Synthetic Intelligence Lab

Problem Formulation

- A solution to a problem is the path from the initial state to the goal state;
- The quality of the solution is measured by the path cost function and an "ideal" solution has the lowest path cost among all possible solutions;
- The "real" world is obviously more complex:
  - state space is an abstraction for problem solving;
- Sate (abstract) = set of real states;
- Action (abstract) = complex combination of real actions;
  - Eg., "Arad ⇒ Zerind" represents (an abstraction) a more complex set of possible routes, detours, stops, etc.

- For guaranteed realisation, any state "in Arad" must reach some state "in Zerind";
- Solution (abstract) = set of real paths that are solutions in the real world;
- Each abstract action must be "easier" than the original problem.

ISLab
Synthetic Intelligence Lab

Types of Problems

- Deterministic, fully observable environment → **problem of the single state**
  - The agent "**Know**" exactly what state it will be in; the solution is a sequence.
- Deterministic environment, not accessible → **multiple state problem**
  - The agent "**don't know**" where it is; the solution is a sequence
- Non-deterministic and/or partially accessible environment → **contingency problem**
  - Perceptions provide new information about the current state
  - Often interleave search and execution
- Unknown state space → **problem of exploitation**

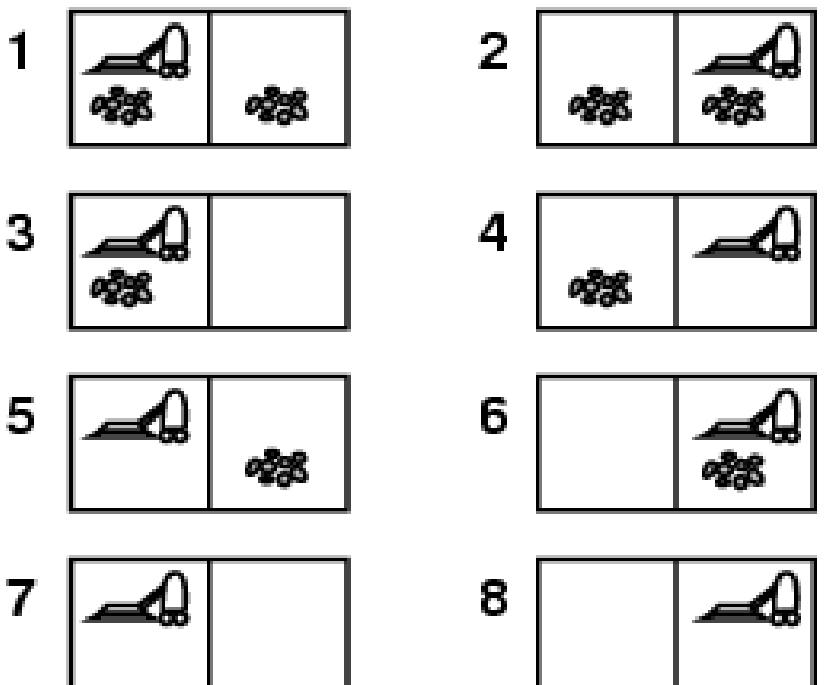
ISLab
Synthetic Intelligence Lab

Types of Problems

Example: the vacuum cleaner problem.

- 2 locations (waste and no waste)
- 3 actions (left, right, suck)
- objective: cleaning up waste
- Cost: one unit per action

- Problem of:
  - Single State if …
  - Multiple State if…
  - Contigency if…
  - Exploration if...



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Types of Problems

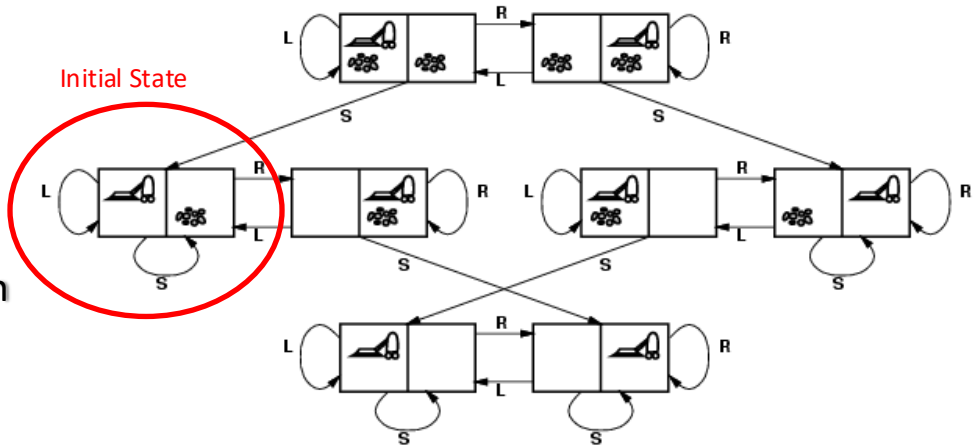
Example: the vacuum cleaner problem

(single state problem)

If the environment is completely observable, the vacuum cleaner will always know where it is and where the waste is.

The solution is then reduced to finding a path from the initial state to the goal state.

Solution? [Right, Suck]



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Types of Problems

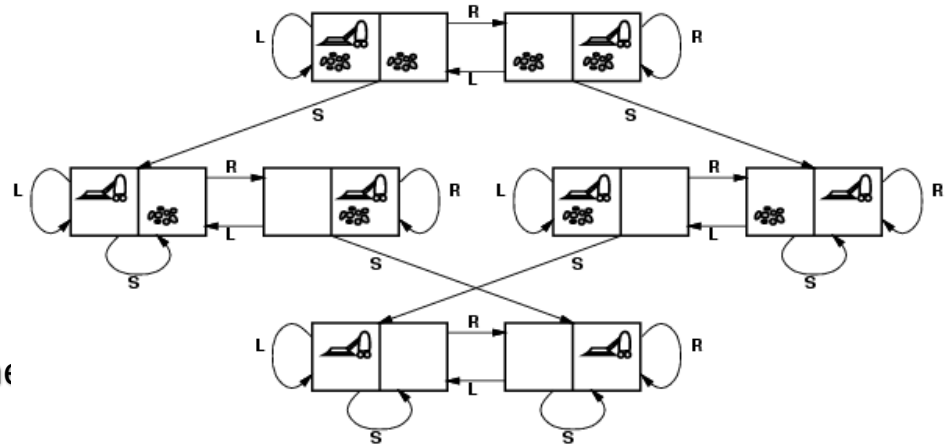
Example: the vacuum cleaner problem

(single state problem)

Formulating the problem:

State: 8 states represented (defined by robot position and garbage)

- Intial satel: anyone
- Operators: left, right, suction
- Objective Test: There is no waste in any of the squares
- Solution Cost: Each action costs 1 (total cost = number of solution steps)



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Types of Problems

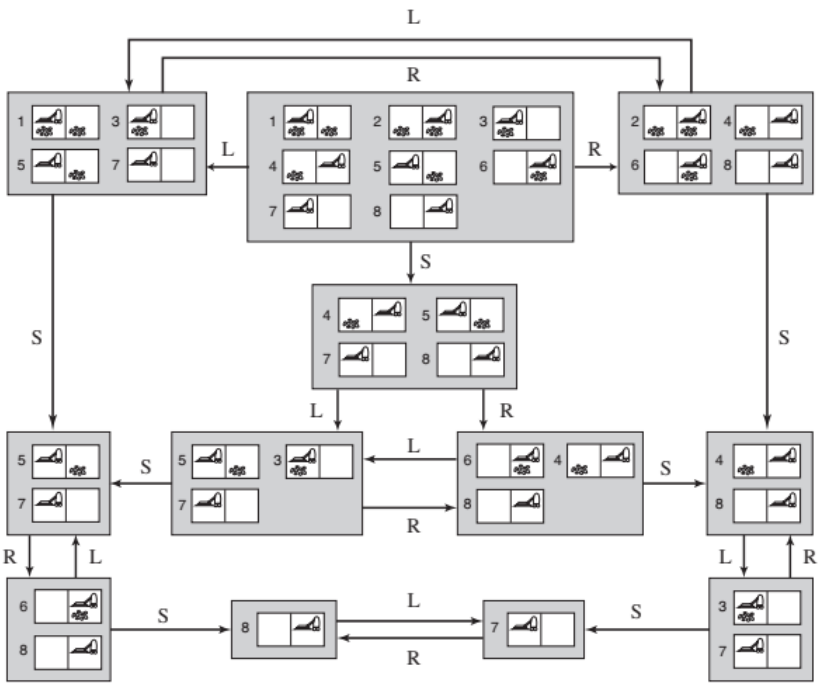
Example: the vacuum cleaner problem

(multiple state problem)

If the vacuum cleaner doesn't have sensors, then it won't know where it is or where the waste is. Even so, it will be able to solve the problem.

Solution? {1,2,3,4,5,6,7,8}

- Right – {2,4,6,8}
- Suck –{4,8}
- Left – {3,7}
- Suck – {7}



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

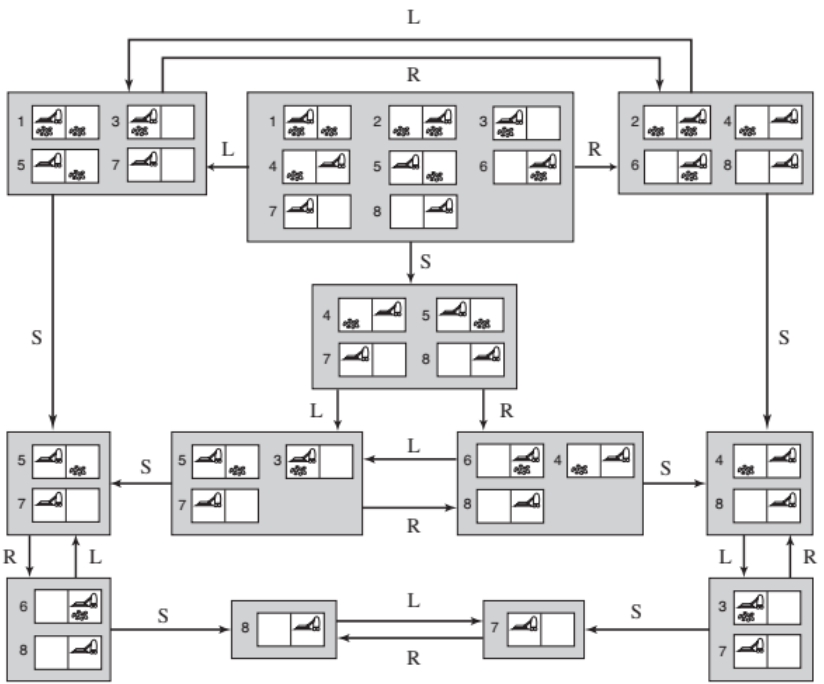
Types of Problems

Example: the vacuum cleaner problem

(multiple state problem)

Formulating the problem:

- Set of States: subset of the states represented
- Operators: left, right and suction
- Test Objective: All states in the set cannot have waste
- Solution Cost: Each action costs 1



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Types of Problems

Example: the vacuum cleaner problem

(contingency problem)

The agent doesn't know what effect its actions will have, because the environment is partially observable.

Therefore, the sequence of actions must be planned according to each perception of the environment, i.e. for each action taken, the agent gathers information through its sensor and only then decides on the next action to take.

The solution will be a list of conditional actions that will interleave search and execution.

Example: Start at {5} or {7} - [Right, if there's waste then Suck]

Types of Problems

Example: the vacuum cleaner problem

(contingency problem)

The agent doesn't know what effect its actions will have, because the environment is partially observable.

Therefore, the sequence of actions must be planned according to each perception of the environment, i.e. for each action taken, the agent gathers information through its sensor and only then decides on the next action to take.

The solution will be a list of conditional actions that will interleave search and execution.

Example: Start at {5} or {7} - [Right, if there's waste then Suck]

Example: the vacuum cleaner problem

(exploration problem)

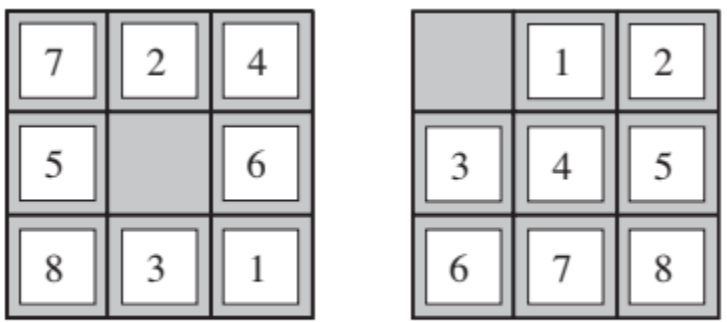
It is an extreme case of a contingency problem that is solved by considering additional information for

Example Real World Problems

8-piece puzzle problem

States: description of the location of the eight pieces and blank square

- Initial state: the initial configuration of the puzzle
- Actions: Move the white square left, right, up and down
- Objective state: state that corresponds to the puzzle configuration on the right
- Cost: each step costs 1 unit, the cost of the solution is the number of steps to solve the problem



Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Example Real World Problems

Routes/Paths problem

- Finding the best route from one point to another (applications: google maps, computer networks, military planning, air travel)
- Visiting each point at least once in a given space (e.g. traveling salesman visiting each city exactly once, finding the shortest route)

Others:

- Autonomous navigation (with a few degrees of freedom)
  - The difficulty increases rapidly with the number of degrees of freedom. Possible complications include: errors in perception of the environment, unknown environments, etc.
- Automatic assembly sequence
  - Planning the assembly of complex objects (by robots)
  - etc.

Searching for Solutions

- A solution to a given problem is defined as the sequence of actions from the initial state to the goal state. The "quality" of the solution is measured by the path cost function and can be :
  - A satisfactory solution is any solution;
  - a semi-optimal solution is one that has approximately the lowest cost of all the solutions;
  - an optimal solution is one that has the lowest cost of all the solutions.

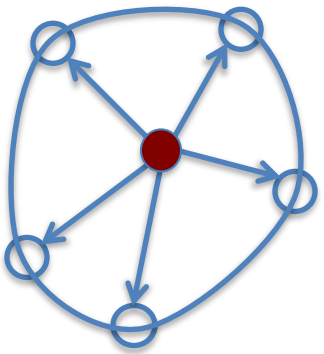
Search
Basic Idea

- Starting from the initial state (node) and expanding it by making a list of all possible successor states;
- Keeping a list of unexpanded states (nodes):
- At each step, choose a state from the list of unexpanded states to expand;
- Continue until you reach the target state;
- Try to expand as few states as possible.

Search
Basic Idea

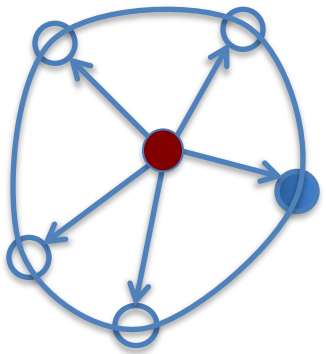
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



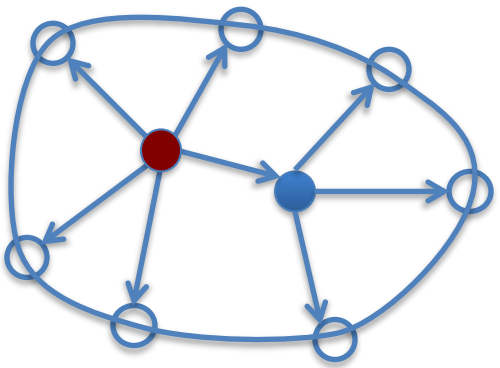
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



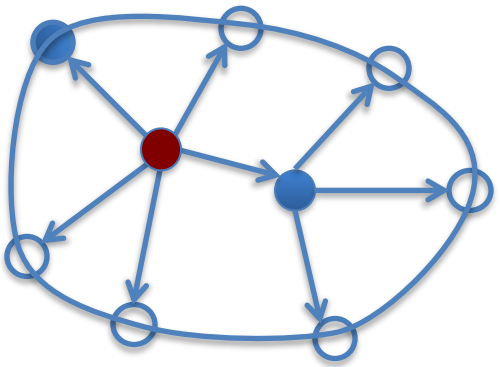
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



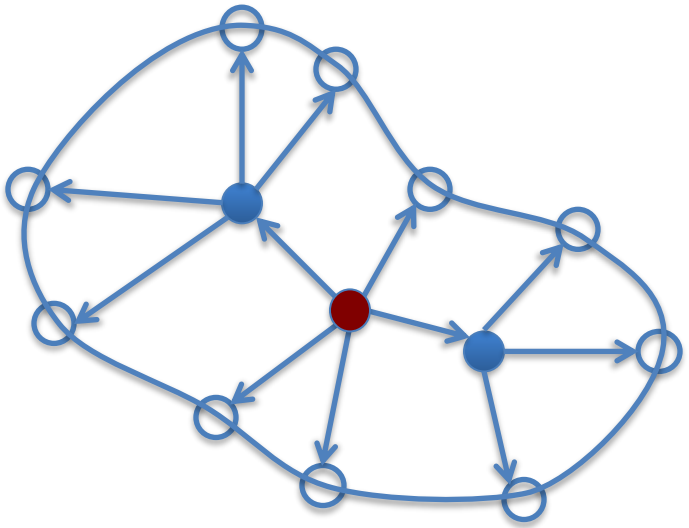
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



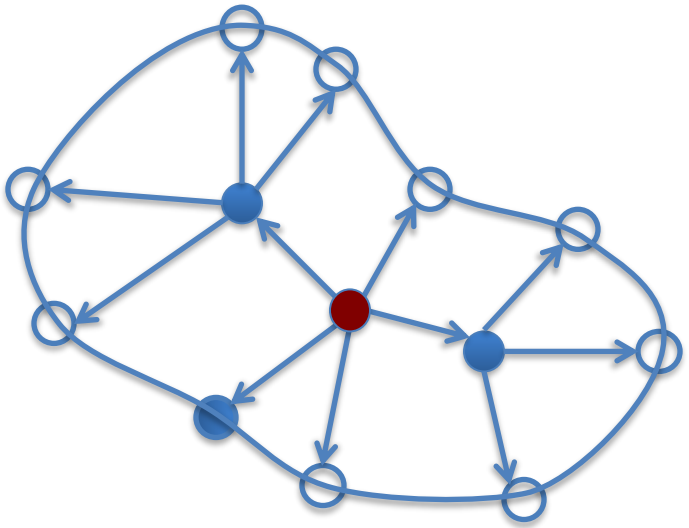
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



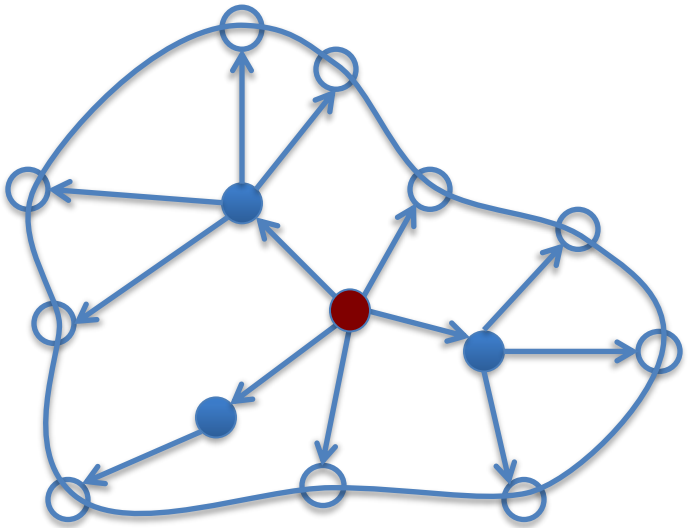
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



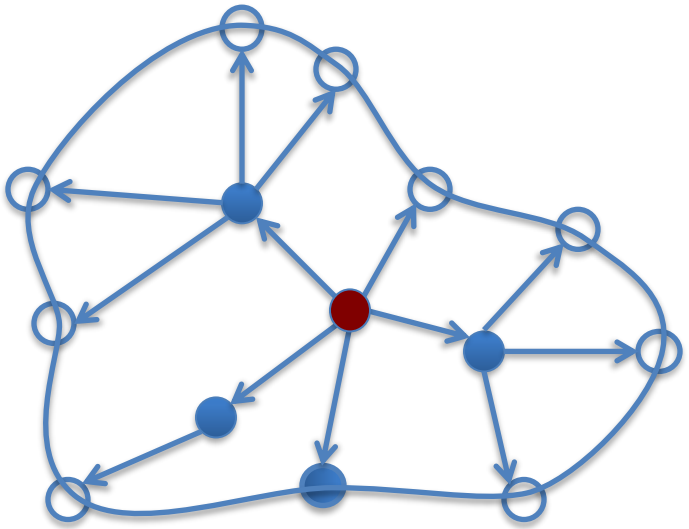
Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

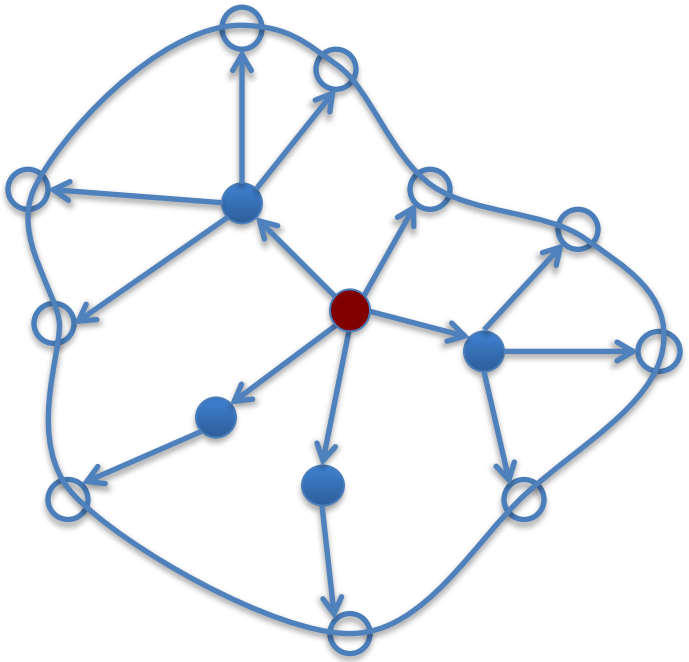
Search
Basic Idea



Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

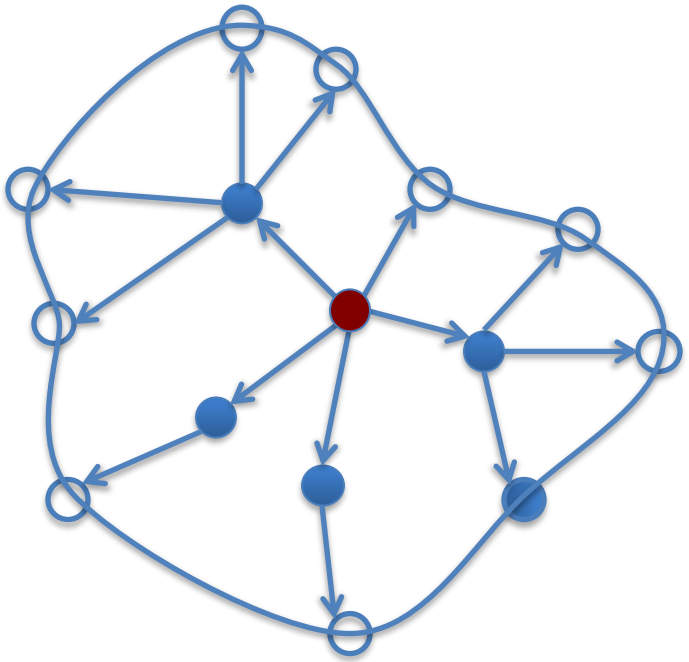
Search
Basic Idea



Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

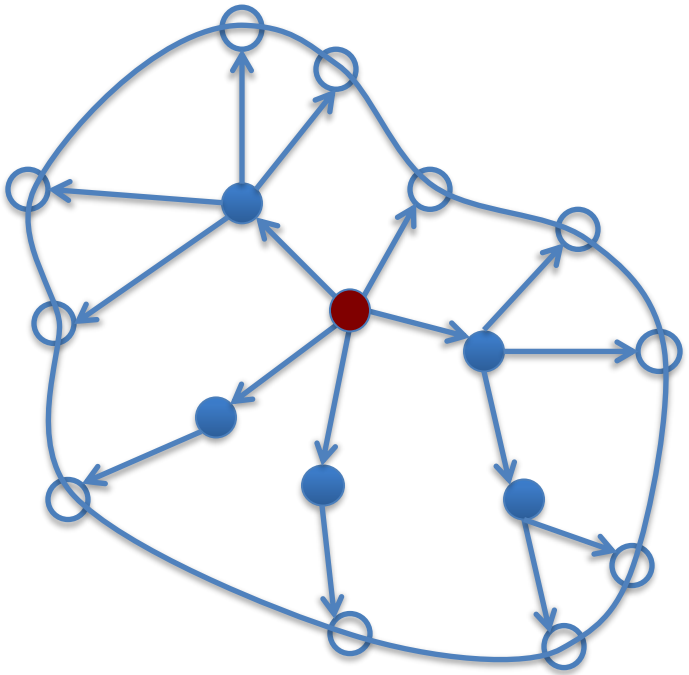
ISLab
Synthetic Intelligence Lab

Search
Basic Idea



Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Search
Basic Idea



Source : Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

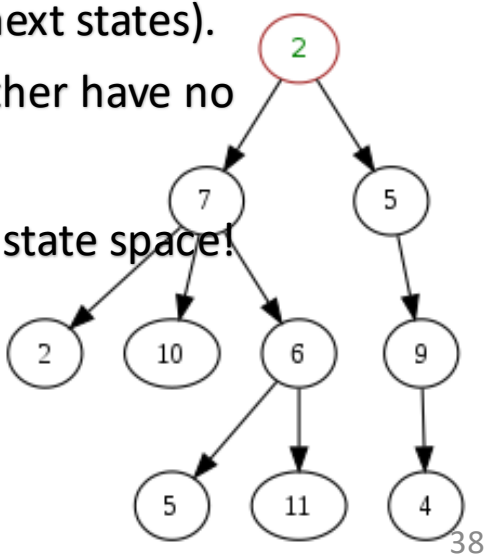
Searching for Solutions

Methodology for Solution Search:

1. Start with the initial state
2. Run the goal test
3. If no solution is found, use the operators to expand the current state by generating new successor states (expansion)
4. Run the goal test
5. If we haven't found the solution, choose which state to expand next (search strategy) and carry out this expansion
6. Back to 4.

Solution Search
Search Tree

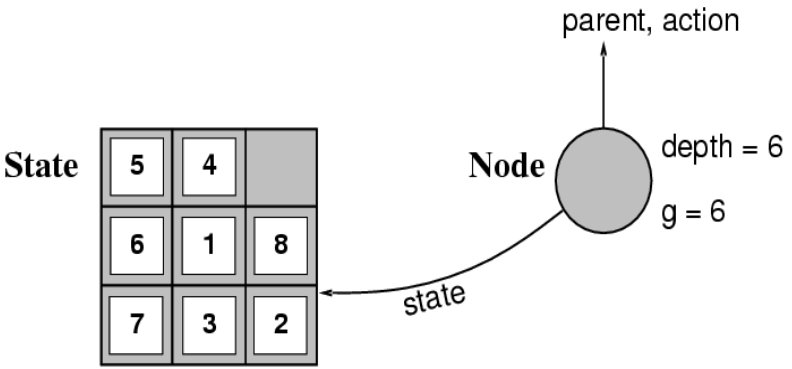
- Using the problem's **state space**, we can form a **search tree** to help us find the solution.
- The **initial state** determines the **root node** of the tree and the **branches** of each node are the possible actions from the node (state) to the leaves (next states).
- It is therefore a search tree made up of nodes. Leaf nodes either have no successors or have not yet been expanded.
- It's important to distinguish between the search tree and the state space!



Solution Search
Search Tree

States versus Nodes

- A **state** is the representation of a physical configuration
- A **node** is a data structure consisting of part of the search tree that includes **state, parent node*, action**, path cost g(x)***, depth**



By expanding the tree, new nodes are created by filling in the various fields and using the "successors" of the problem to create the corresponding states.

* originating node

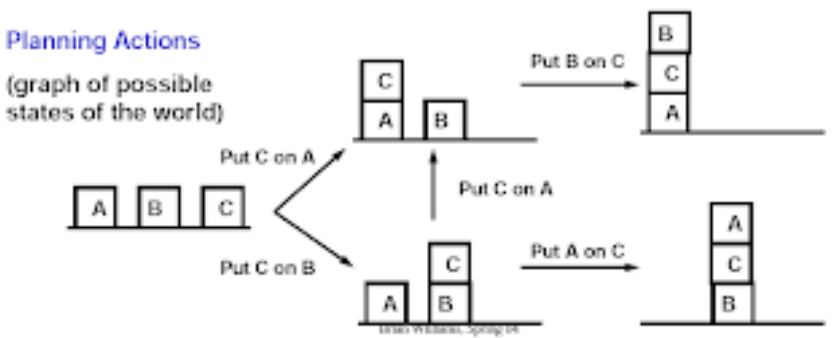
** operator applied to generate it

*** cost of the path from the initial node

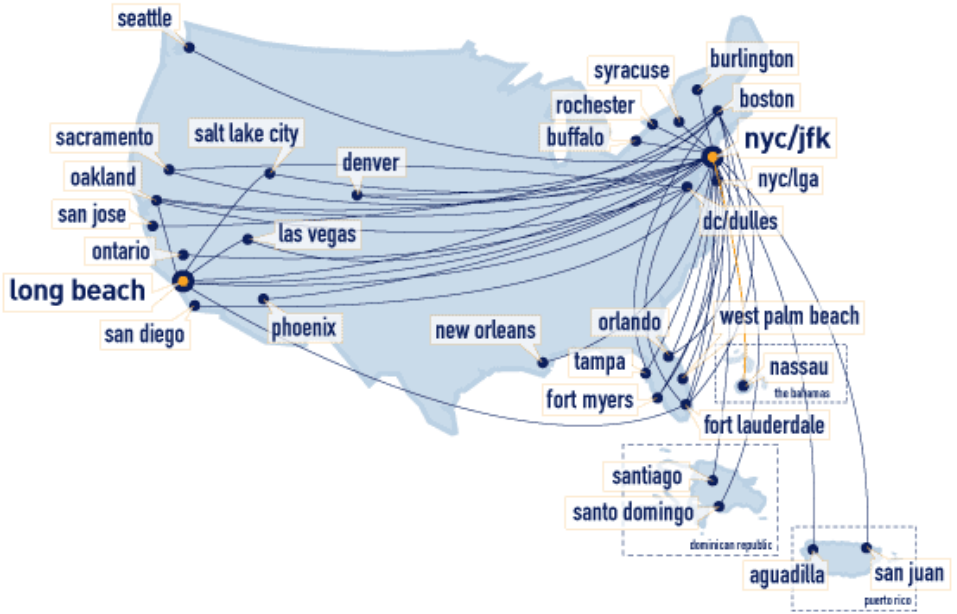
Source : Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

ISLab
Synthetic Intelligence Lab

Examples of Graphs



Fonte: Brian Williams, 2014

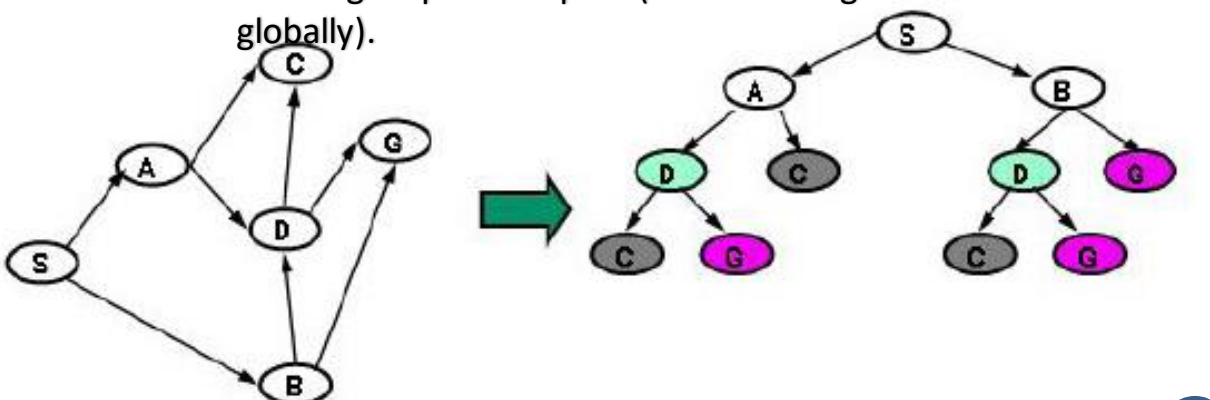


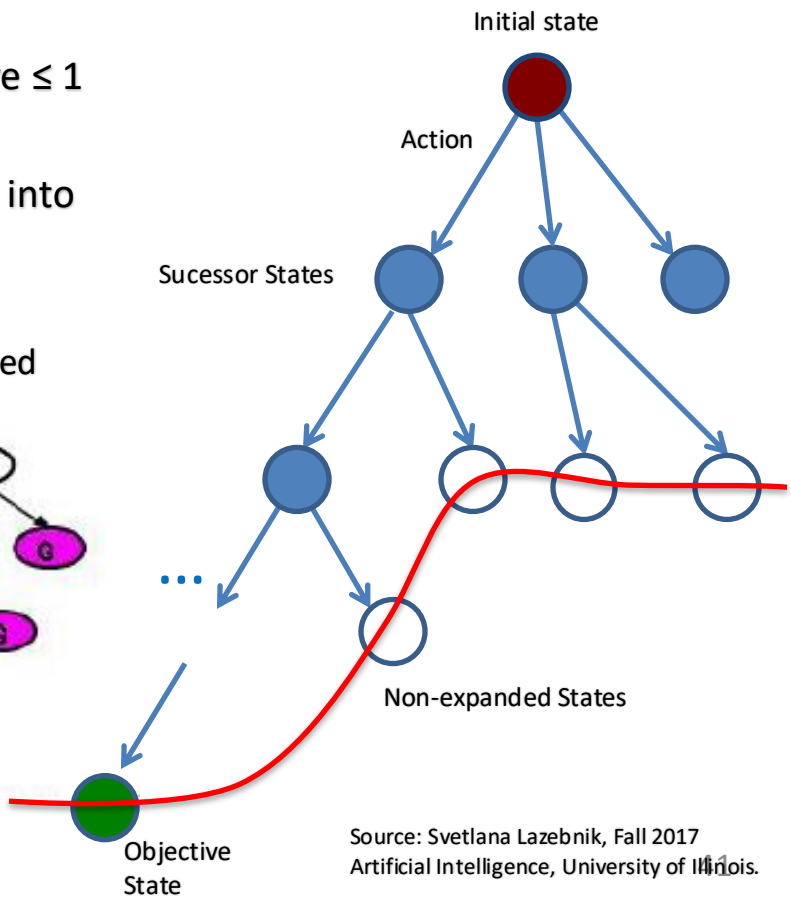
Source : Jet Blue airlines: http://www.jetblue.com/travelinfo/routemap.html

ISLab
Synthetic Intelligence Lab

Search Graphs as Search Trees

- Trees are graphs without cycles in which the nodes have ≤ 1 parent;
- We can transform graph search problems (from S to G) into tree search problems :
  - replacing undirected links with 2 directed links;
  - avoiding loops in the path (or monitoring the nodes visited globally).





Fonte: https://ocw.mit.edu/courses/health-sciences-and-technology/hst-947-medical-artificial-intelligence-spring-2005/lecture-notes/ch2_search1.pdf

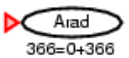
Source: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Tree Search Algorithm

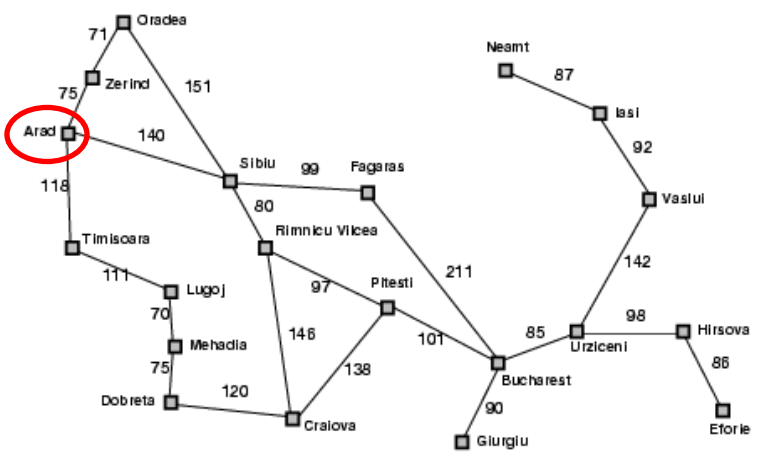
- Initialize the list of unexpanded states using the initial state
- As long as the list of unexpanded states is not empty
  - Choose a node from the list of non-expanded states according to the search strategy and remove it from the list
  - If the node contains the goal state, return the solution
  - Otherwise, expand the node and include its children in the list of unexpanded states

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



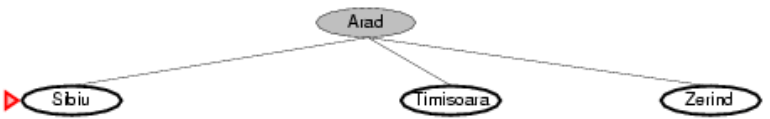
Initial: Arad
Objetive: Bucharest



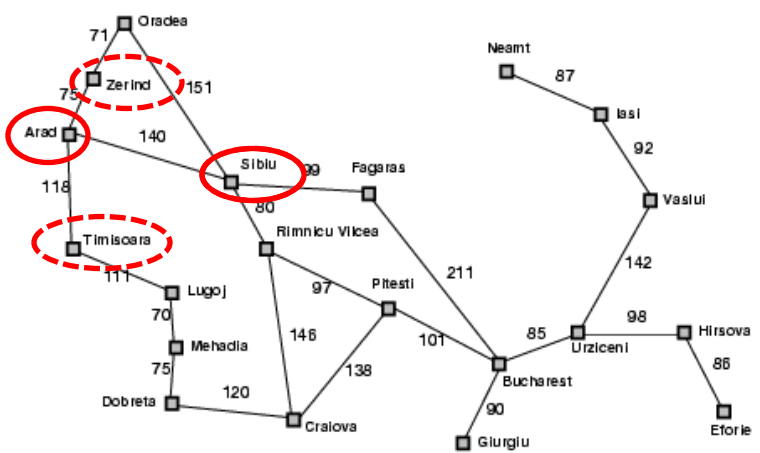
Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



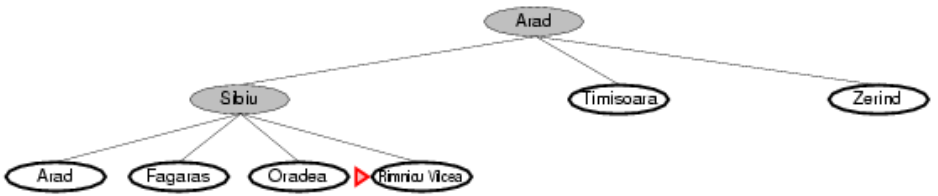
Initial: Arad
Objetive: Bucharest



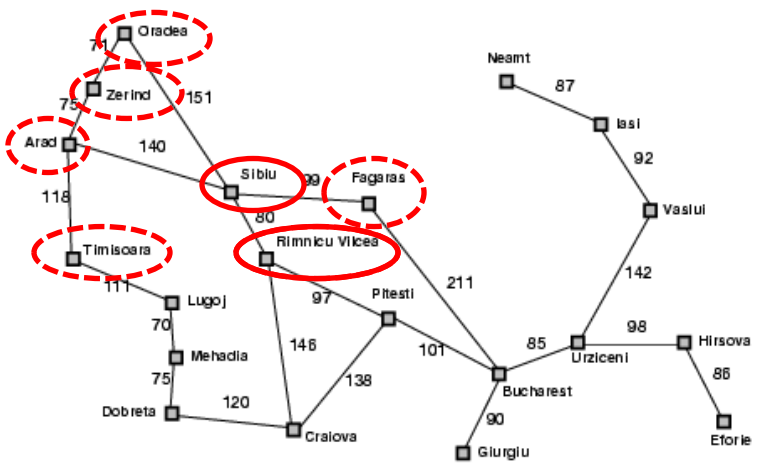
Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



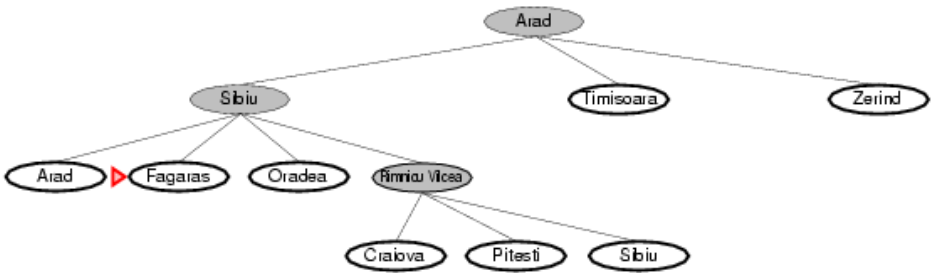
Initial: Arad
Objetive: Bucharest



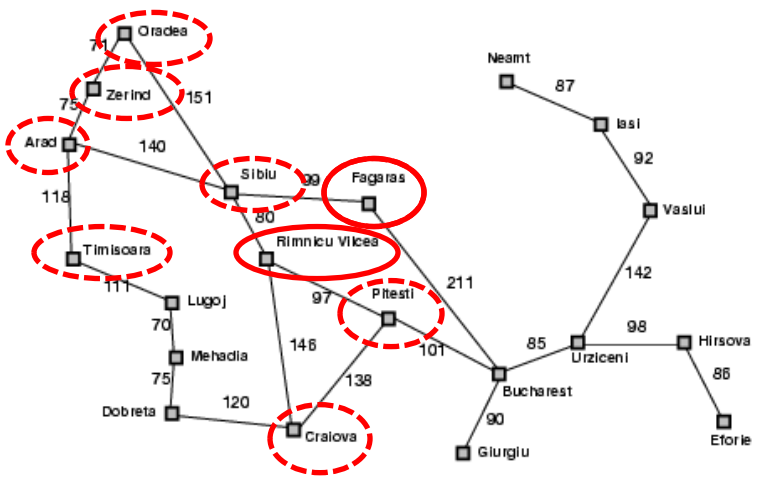
Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



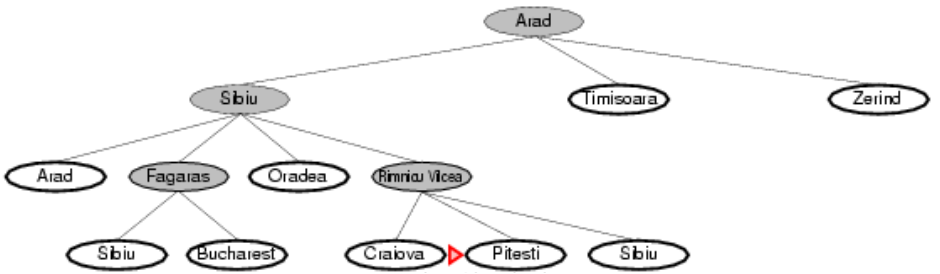
Initial: Arad
Objetive: Bucharest



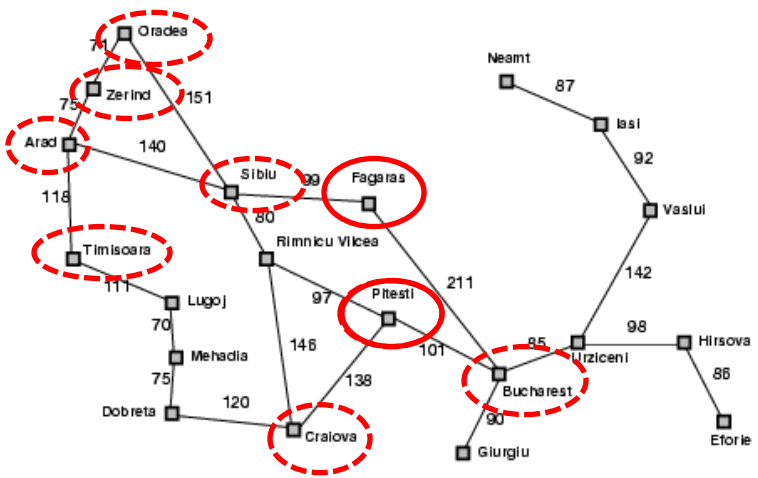
Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



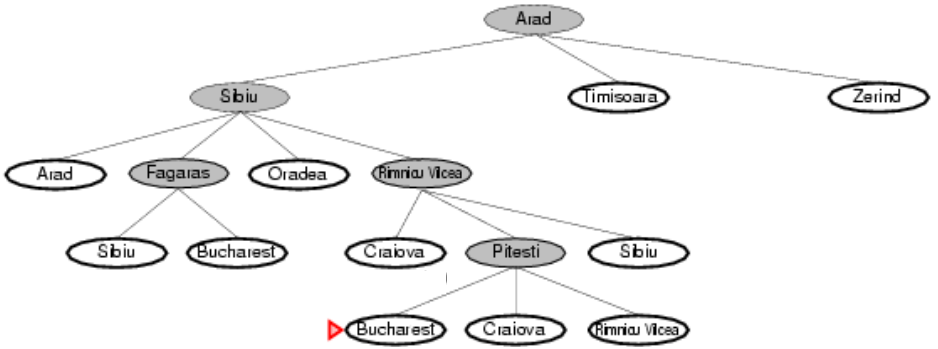
Initial: Arad
Objetive: Bucharest



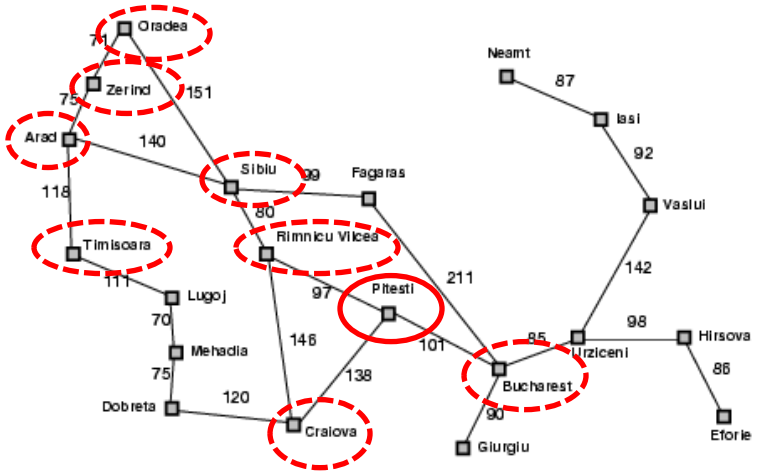
Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



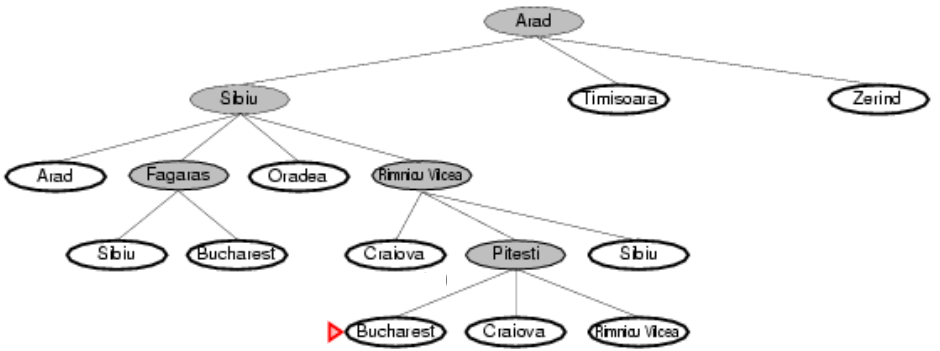
Initial: Arad
Objetive: Bucharest



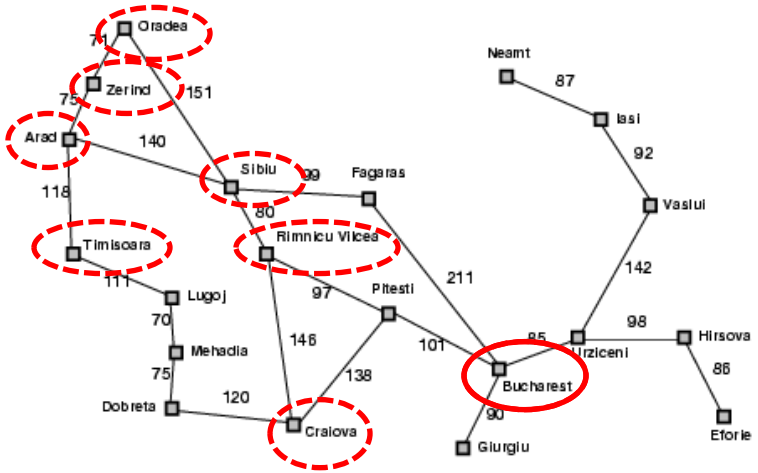
Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Example of a Tree Search



Initial: Arad
Objetive: Bucharest



Source : Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

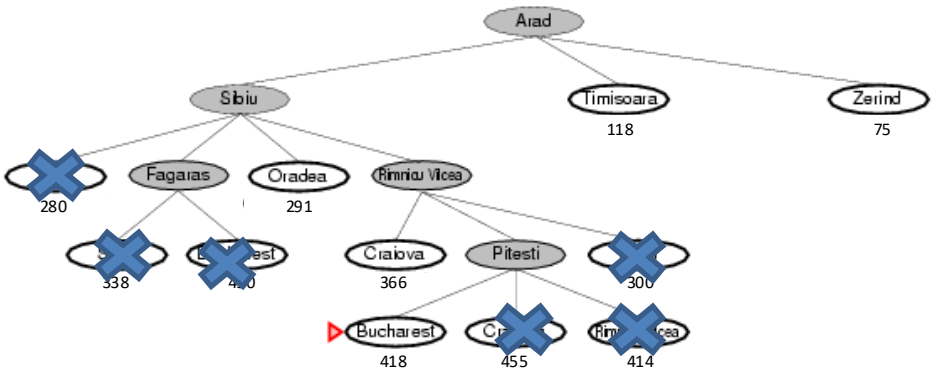
Tree Search Algorithm
Manipulation of Repeated States

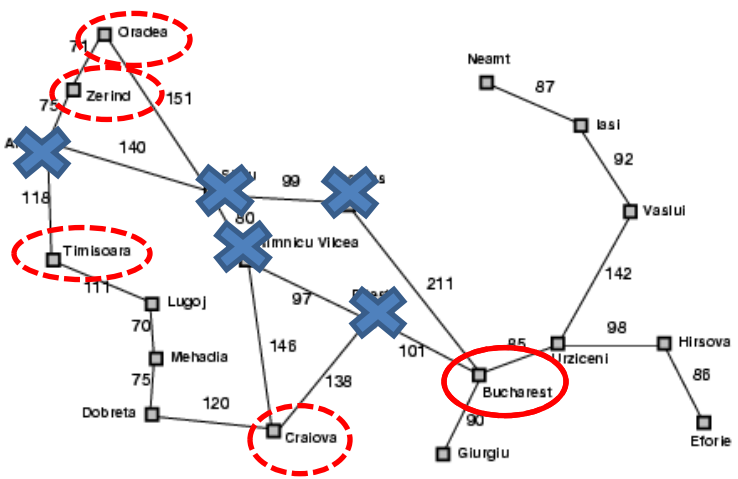
- Initialize the list of unexpanded states using the initial state
- As long as the list of unexpanded states is not empty
  - Choose a node from the list of non-expanded states according to the search strategy and remove it from the list
  - If the node contains the goal state, return the solution
  - Otherwise, expand the node and include its children in the list of unexpanded states

- To deal with repeated states :
  - Each time you expand a node, add that state to the explored set; don't put explored states in the list of unexpanded states again
  - Whenever you add a node to the list of unexplored states, check whether it already exists in the list of unexplored states with a higher path cost and, if so, replace that node with the new one.

ISLab
Synthetic Intelligence Lab

Search Without Repeated States





Initial: Arad
Objetive: Bucharest

Source: Svetlana Lazebnik, Fall 2017
Artificial Intelligence, University of Illinois.

Search Strategies

We can evaluate the performance of the search algorithm using the following criteria :

- Completeness: Are you guaranteed to find the solution?
- Complexity in Time: How long does it take to find the solution?
- Complexity in Space: How much memory do you need to do the search?
- Optimization: Find the best solution?

Time and space complexity are always taken into account when measuring the difficulty of the problem (e.g. the size of the graph and state space).

Search Strategies

- The strategy: the order of node expansion
- Evaluation criteria:
  - Completeness: Are you sure you've found the solution?
  - Optimization: Do you find the best solution?
  - Complexity in Time: How long does it take to find the solution?
  - Complexity in Space: How much memory do you need to do the search?
- Time and the complexity of space are measured in terms of:
  - b: the maximum branching factor (the maximum number of successors to a node) of the search tree
  - d: the depth of the best solution
  - m: the maximum depth of the state space

Search Strategies

- **Types of Search Strategies:**
  - Uninformed Search (blind)
    - First in Width (BFS)
    - Depth First (DFS)
    - Uniform Cost
    - Iterative
    - Bidirectional
  - Informed Search (heuristic)
    - Greedy
    - Algorithm A*

- A **search strategy** is defined by choosing the order of node expansion;
- **Uninformed search strategies** use only the information available in the problem definition;
- In **informed search strategies**, the algorithm is given "hints" about the suitability of different states.

Search Strategies

- **Breadth-first search**
  - Strategy: All shallower nodes are expanded first
  - Good: Very systematic search
  - Bad: Usually takes a long time and above all takes up a lot of space
- Properties:
  - Complete: Yes, if b (branching factor) is finite
  - Time: assuming branching factor b then $n=1+b+b2+b3+ \ldots +b^n = O(b^d)$ is exponential in d
  - Space: Stores each node in $O(b^d)$ memmory
  - Optimal: Yes, if the cost of each step is 1
- In general, only small problems can be solved this way!

- b: the maximum branching factor (the maximum number of successors to a node) of the search tree
- d: the depth of the best solution
- m: the maximum depth of the state space

Breadth-First Search

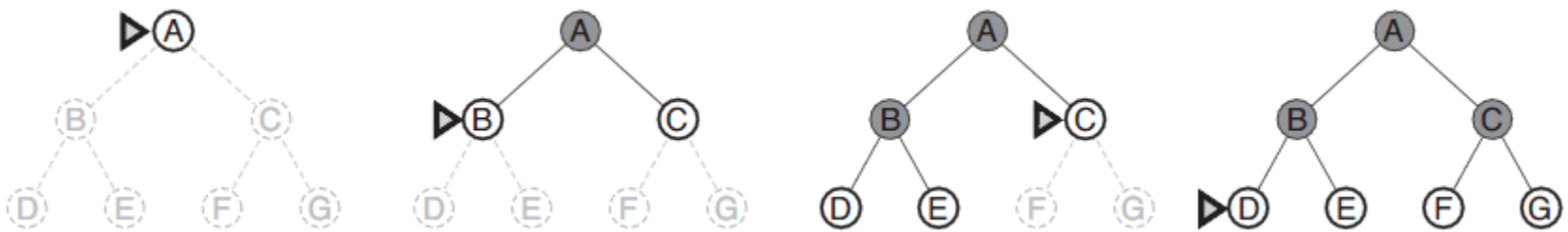
- **Breadth-first search (2)**

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)  /* chooses the shallowest node in frontier */
        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
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Breadth-First Search

- **Breadth-first search (3)**



Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

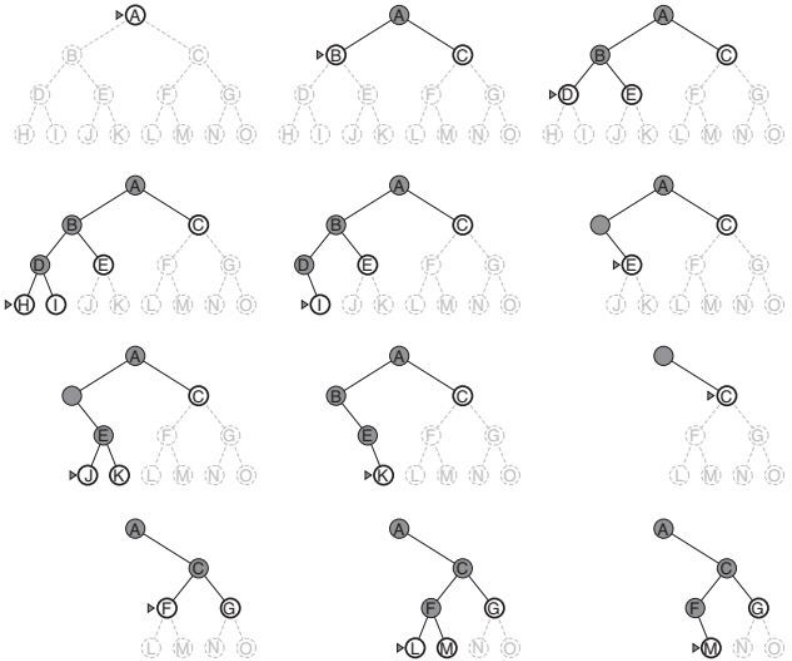
Search Strategies

- **Depth-First Search**
  - Strategy: Always expand one of the deepest nodes in the tree
  - Good: Very little memory required, good for problems with many solutions
  - Bad: Cannot be used in trees with infinite depth, can get stuck on wrong branches
- Properties:
  - Complete: No, fails in spaces of infinite depth, with loops
    - Change to avoid repeated states along the way
  - Time: $O(b^m)$, bad if $m > d$
  - Space: O(bm), linear space
  - Optimal: No (in principle it returns the 1st solution it finds)
- Sometimes a limit depth (l) is set and it becomes a Limited Depth Search

- b: the maximum branching factor (the maximum number of successors to a node) of the search tree
- d: the depth of the best solution
- m: the maximum depth of the state space

Depth-First Search

- **Depth-First Search (2)**



Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

For problems with several solutions, this strategy can be much faster than breadth-first search;

However, this strategy should be avoided when the trees generated are very deep or generate infinite paths.

ISLab
Synthetic Intelligence Lab

Depth-First Search
Depth-limited Search

= depth search with depth limit l, i.e. nodes at depth l have no successors

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

One of the problems with depth-first search is its inability to deal with infinite paths;

Depth-first search with a depth limit seeks to avoid this problem by setting the maximum level of demand.

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

DFS vs BFS

- **Depth-First Search**
  - choose the first element from the list of unexpanded states
  - add path extensions to the front of the list of unexpanded states

- **Breadth-first search**
  - choose the first element of the list of non-expanded states
  - add path extensions at the end of the list of non-expanded states

Uniform Cost Search

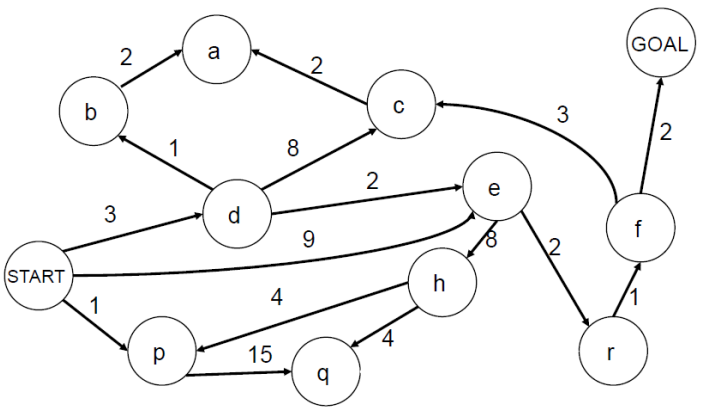
- Strategy:
  - For each node in the list of unexpanded states, save the total cost of the path from the initial state to that node
  - Always expand the node with the lowest cost in the list of unexpanded states (measured by the solution cost function)
- Demand First in Width equals Uniform Cost Search if g(N)=Depth(N)
- Equivalent to **Breadth-first search**, if all costs are equal
- Implementation: list of non-expanded states is a priority list ordered by path costT
- We must ensure that g(successor)>= g(N)
- Equivalent to Dijkstra's algorithm in general

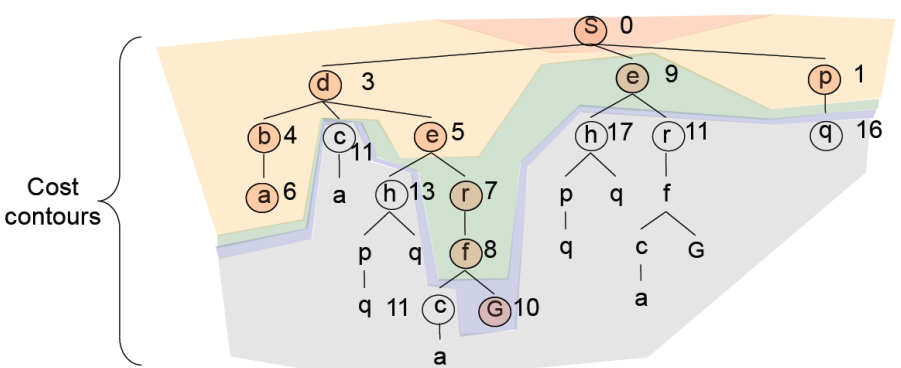
In all nodes N, g(N) is the known cost of going from the root to node N.

Dijkstra's algorithm (Edsger Dijkstra, 1956) solves the shortest path problem in a directed or undirected graph with non-negative weight edges.

Uniform Cost Search (2)

- Order of expansion:
  (S,p,d,b,e,a,r,f,e,G)

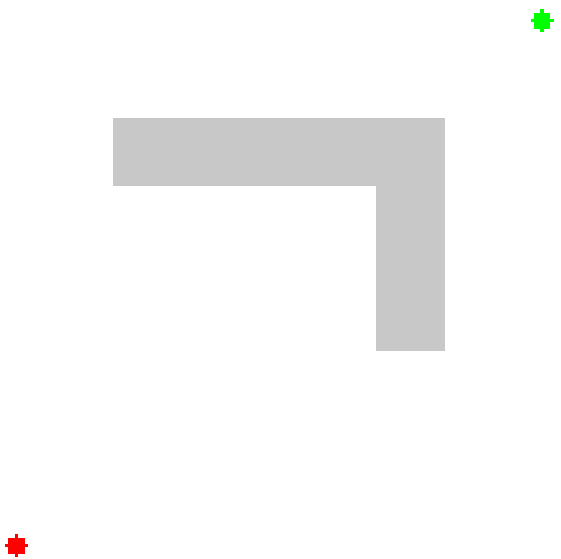




Source: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

ISLab
Synthetic Intelligence Lab

Uniform Cost Search (3)



Source_https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Practical_optimizations_and_infinite_graphs

Uniform Cost Search

- Properties:
  - Complete: Yes, if the cost of the step is greater than some positive constant ε (we don't want infinite sequences of steps with a finite total cost)
  - Time :
    - *Number of nodes with path cost ≤ cost of ideal solution (C *), $O(b^{C^*/\varepsilon})$*
    - *I can be great than O (bd): search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps*
  - Space: $O(b^{C^*/\varepsilon})$
  - Optimal: Yes

- b: the maximum branching factor (the maximum number of successors to a node) of the search tree
- d: the depth of the best solution
- m: the maximum depth of the state space

ISLab
Synthetic Intelligence Lab

Iterative Search
Progressive Deepening

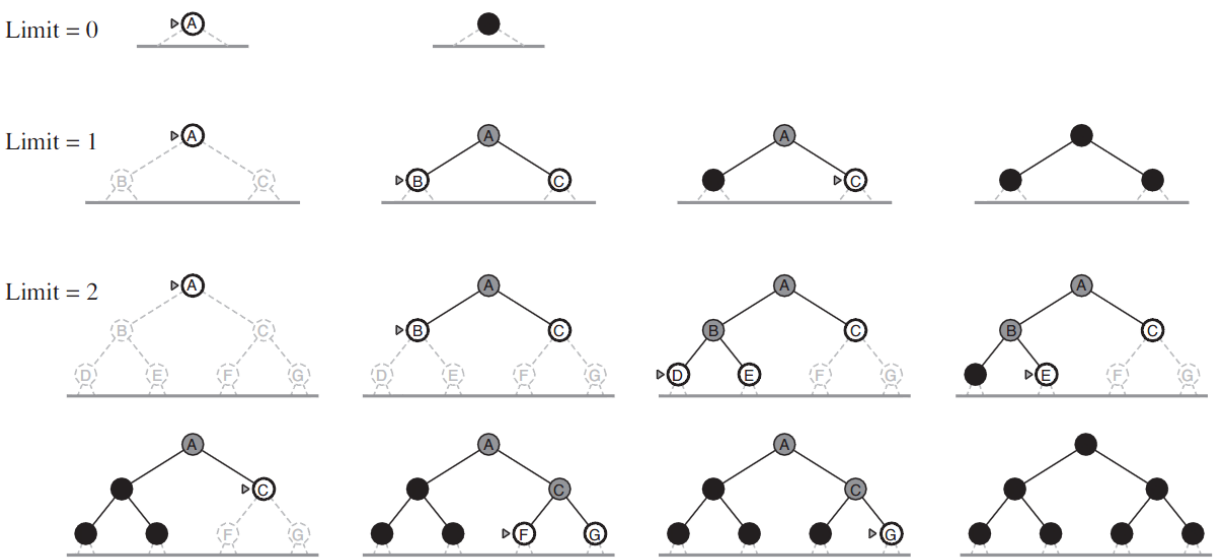
If we don't know the maximum limit value, we'll be condemned to a depth-first search strategy and have to deal with the problem of possible infinite paths. The answer is to change the principle of limited search by varying this limit between zero and infinity.

Use **Depth-First Search** as a subroutine

- Verificar a raiz
- Desenvolver um DFS procurando um caminho de comprimento 1
- Se não houver um caminho de comprimento 1, desenvolver um DFS procurando um caminho de comprimento 2
- Se não houver um caminho de comprimento 2, desenvolver um DFS procurando um caminho de comprimento 3…

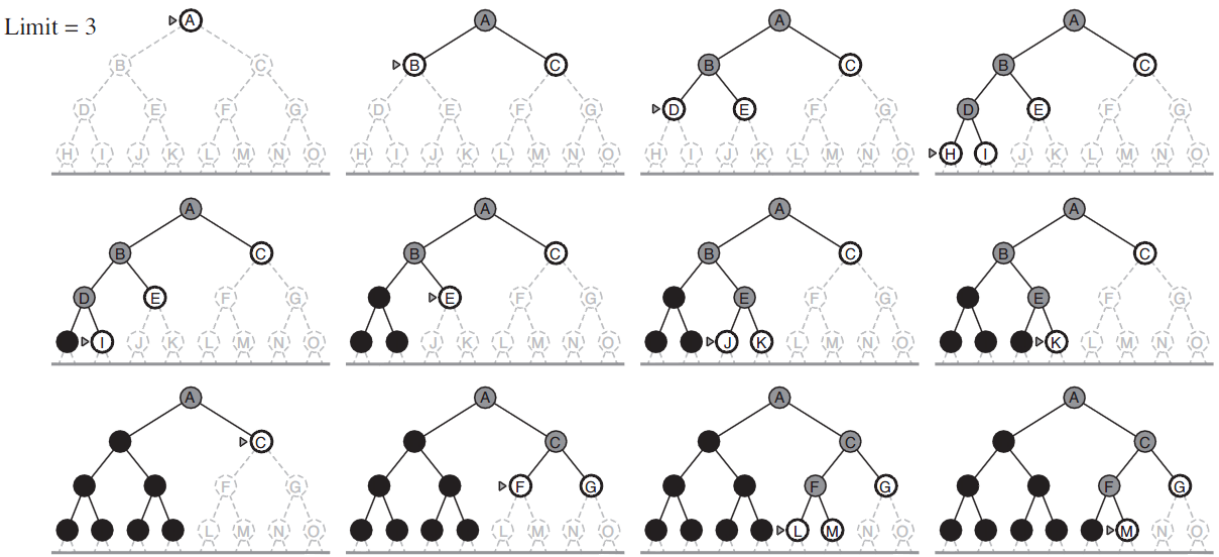
Iterative Search
Progressive Deepening

- **Iterative Depth Search (2)**



Iterative Search
Progressive Deepening

- **Iterative Depth Search(3)**



The iterative search algorithm is a good option for problems where we have to resort to a blind method.

The search space is large, but we don't know the maximum level at which a solution can be found.

ISLab
Synthetic Intelligence Lab

Iterative Search
Progressive Deepening

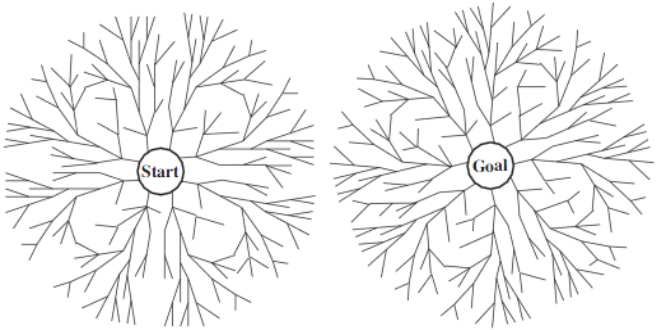
- Strategy: Perform limited depth searches, iteratively, always increasing the depth limit
- Properties:
  - Completed: Yes
  - Time: $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
  - Space: $O(b^d)$
  - Optimal: Yes, if cost is 1.

- b: the maximum branching factor (the maximum number of successors to a node) of the search tree
- d: the depth of the best solution
- m: the maximum depth of the state space

- In general, it is the best (uninformed or blind) strategy for problems with a large search space and where the depth of the solution is unknown.

Bidirectional Search

- Strategy: Perform a search forward from the initial state and backward from the goal, simultaneously.
  - Good: Can greatly reduce complexity in time $O(b^{d/2})$
  - Problems: Is it possible to generate predecessors? What if there are too many goal states? How can we match the two searches? What kind of search should be done in the two halves?
- Eg., To find a route in Romania, there is only one objective state, so backward search is very similar to forward search; but if the objective is an abstract description, such as "no queen attacks a[illegible] bidirectional search is difficult to use.



Comparing Search Strategies

- Evaluation of search strategies:
  - B is the branching factor
  - d is the depth of the solution
  - m is the maximum depth of the tree
  - l is the limit depth of the search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Search Strategies

- **Informed search (Heuristics)**
  - It uses information about the problem to prevent the search algorithm from getting "lost wandering in the dark”
- **Search strategies:**
  - Defined by choosing the order of node expansion
- **Searching for the Best First (Best-First Search)**
  - Uses an evaluation function that returns a number indicating the interest of expanding a node
  - Greedy-Search - f(n) = h(n) (function that estimates the distance to the solution)
  - A* algorithm - f(n) = g(n) + h(n) (estimates the cost of the best solution passing through n)

Heuristics

- As a search technique for obtaining goals in non-algorithmic problems that generate combinatorial "explosions";
- As an approximation method for solving problems using heuristic (hint) type evaluation functions;
- As a pruning (cutting) method for game program strategies.

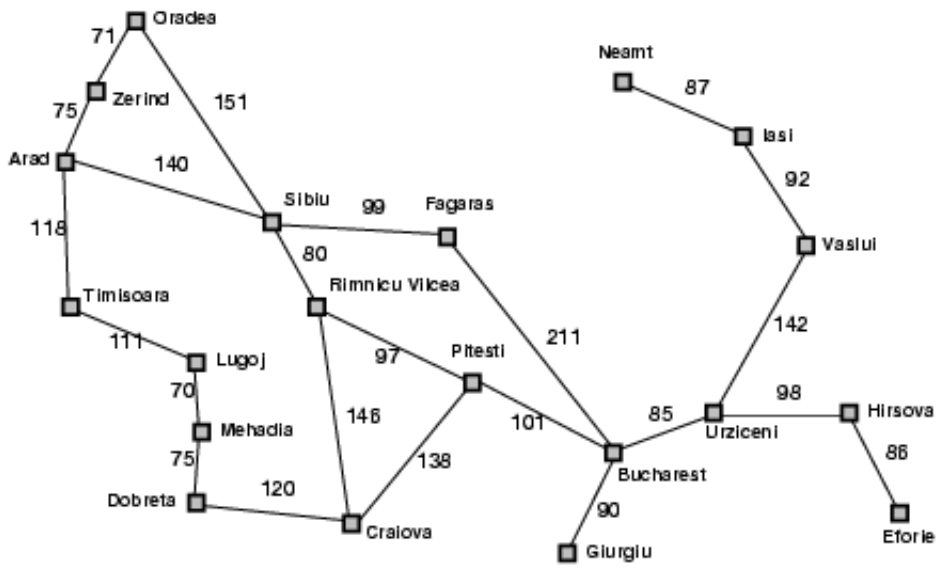
- In a search, we can apply two types of generic heuristics, which node will be expanded and which nodes should be discarded.
  - If the universe is fully known, the heuristic will be developed by assigning numbers;
  - If the universe is not fully known, the heuristic will be developed by applying rules.

- Heuristics are specific to each problem. These functions can be inaccurate or not even capable of finding the best answer, but their use makes it possible not to use combinatorial analysis.
- Implementing heuristics is difficult! To the extent that it is difficult to accurately measure the value of a given solution, it is also difficult to measure certain knowledge in a way that allows a mathematical analysis of its effect on the search process to be carried out.

ISLab
Synthetic Intelligence Lab

Heuristics for the Romania Problem

- Initial State: Arad; Objetive: Bucharest; h(n)= distance in a straight line



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

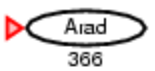
Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

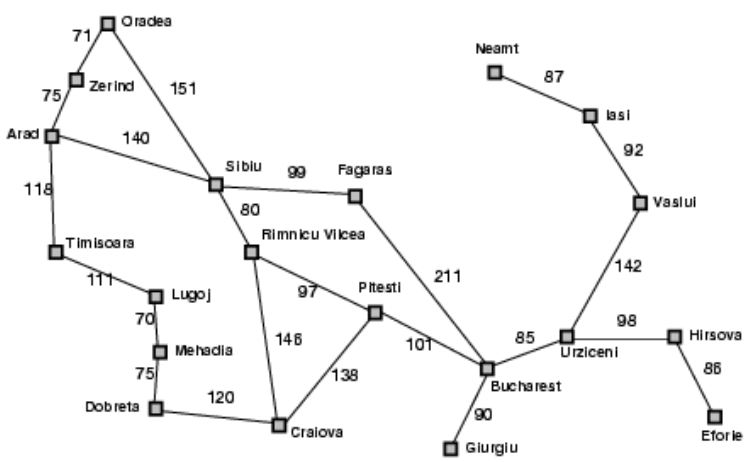
Search Strategies

- **Informed (Greedy-Search)**
  - **Strategy:**
    - Expand the node that seems closest to the solution
  - h(n) = estimated cost of the shortest path from state n to the goal (heuristic function)
  - function GREEDY-SEARCH(problem) returns a solution or failure
    return BEST-FIRST-SEARCH(problem,h)
  - **Example:**
    - h(n)= straight line distance between n and the goal

Greedy-Search

- Initial State: Arad; Objetive: Bucharest; h(n)= distance in a straight line



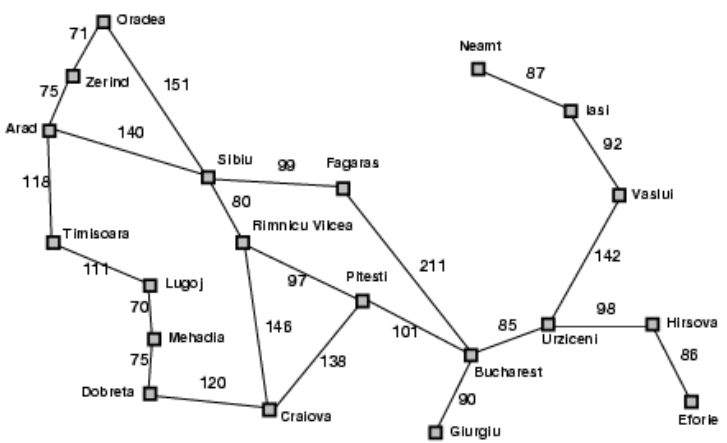


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Greedy-Search

- Initial State: Arad; Objetive: Bucharest; h(n)= distance in a straight line

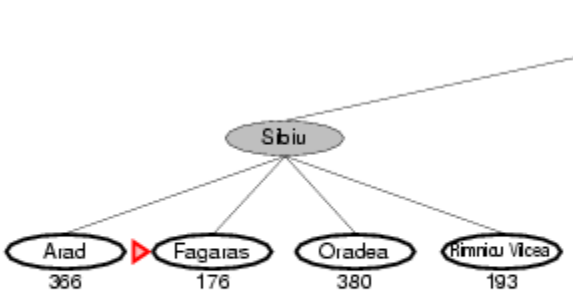


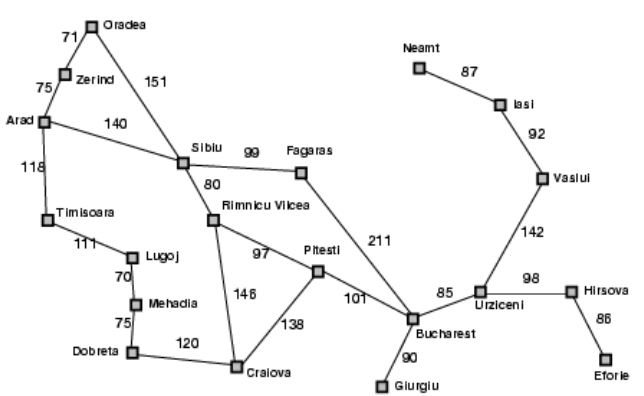
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Greedy-Search

- Initial State: Arad; Objetive: Bucharest; h(n)= distance in a straight line



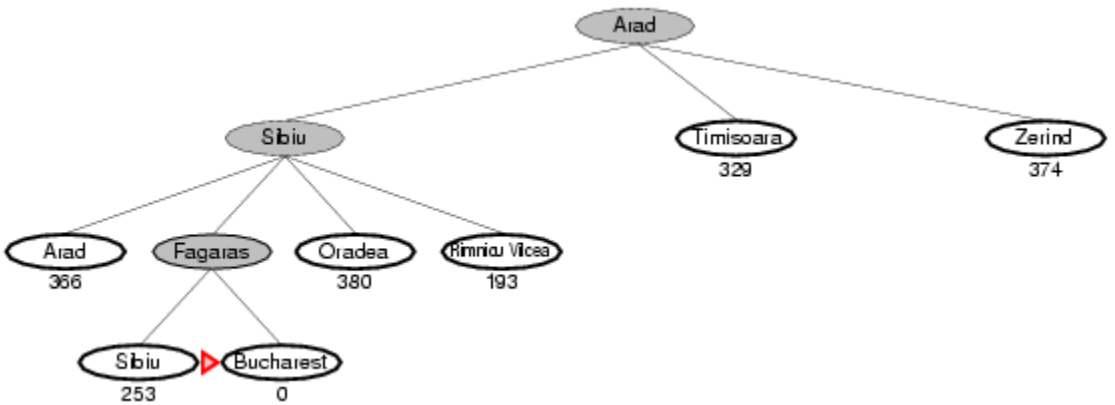


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Greedy-Search

- Initial State: Arad; Objetive: Bucharest; h(n)= distance in a straight line



Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Greedy-Search

- **Properties**
  - Complete? No, it can go into a cycle.
    - Susceptible to false starts
  - Complexity in time? $O(b^m)$
    - but with a good heuristic function you can reduce it considerably
  - Complexity in space? $O(b^m)$
    - Keeps all nodes in memory
  - Optimal? No, because it doesn't always find the optimal solution.
  - It needs to detect repeated states.

Search Strategies

- **Informed (Search A*)**
- **Strategy:**
  - avoid expanding expensive paths
  - The A* algorithm combines greedy search with uniform search, minimizing the sum of the path already taken and the minimum expected path to the solution. It uses the:

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

  - g(n) = total cost so far to reach state n (cost of the journey)
  - h(n) = estimated cost of reaching the goal (must not overestimate the cost of reaching the solution (heuristic))

f(n)= estimated cost of the least expensive solution passing through node n

Search A*

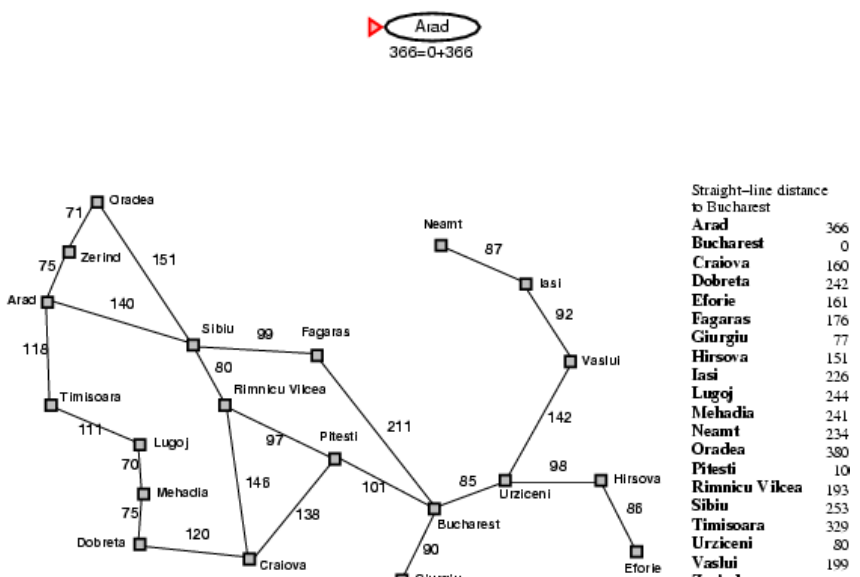
- **function** A*-SEARCH(problem) **returns** a solution or failure

  return BEST-FIRST-SEARCH(problem,g+h)

  - Algorithm A* is optimal and complete.
  - Complexity in time exponential in (relative error of h*solution length)
  - Complexity in space: Keeps all nodes in memory.

Search A*

- Initial State: Arad; Objetive: Bucharest; f(n) = g(n) + h(n);
- g(n) = cost until n; h(n) = straight line distance to the goal

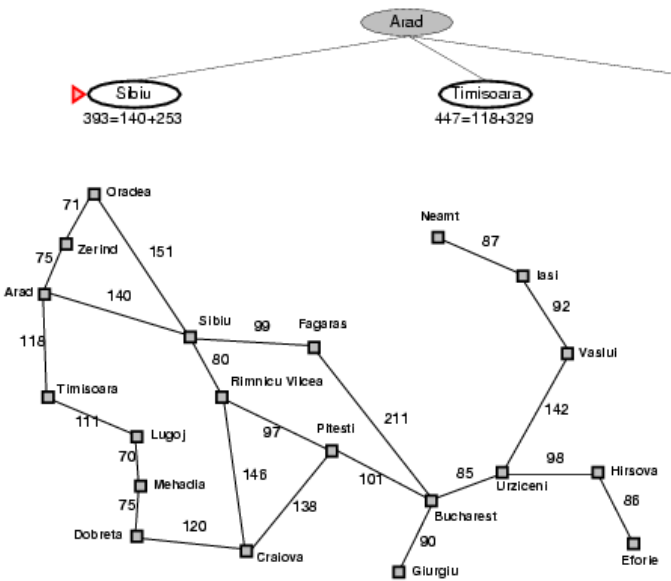


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Search A*

- Initial State: Arad; Objetive: Bucharest; f(n) = g(n) + h(n);
- g(n) = cost until n; h(n) = straight line distance to the goal

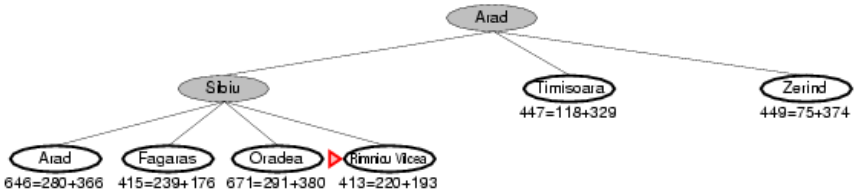


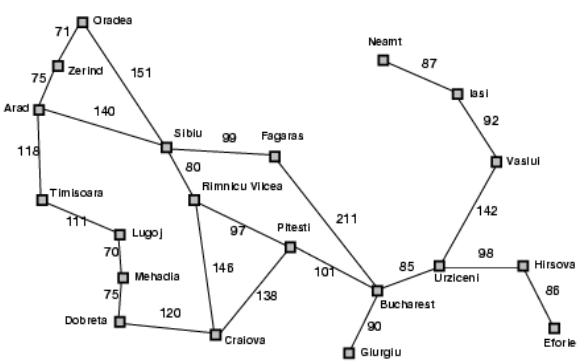
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Search A*

- Initial State: Arad; Objetive: Bucharest; f(n) = g(n) + h(n);
- g(n) = cost until n; h(n) = straight line distance to the goal



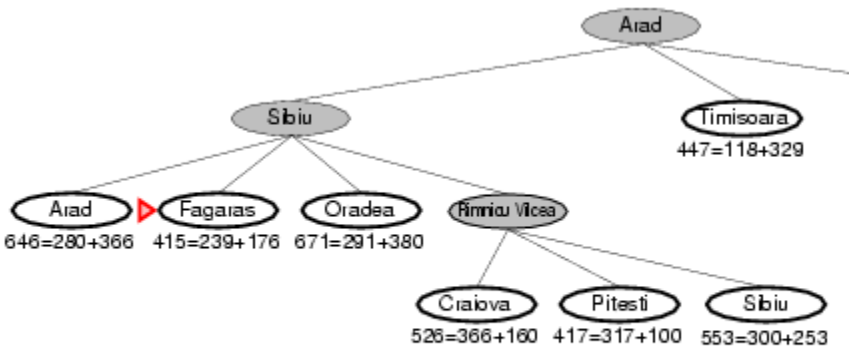


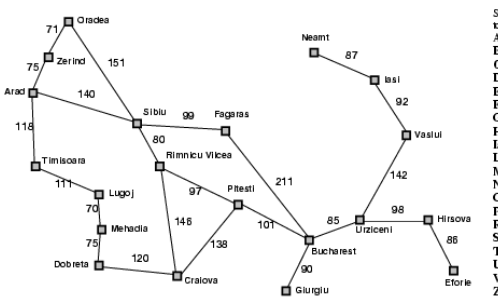
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Search A*

- Initial State: Arad; Objetive: Bucharest; f(n) = g(n) + h(n);
- g(n) = cost until n; h(n) = straight line distance to the goal



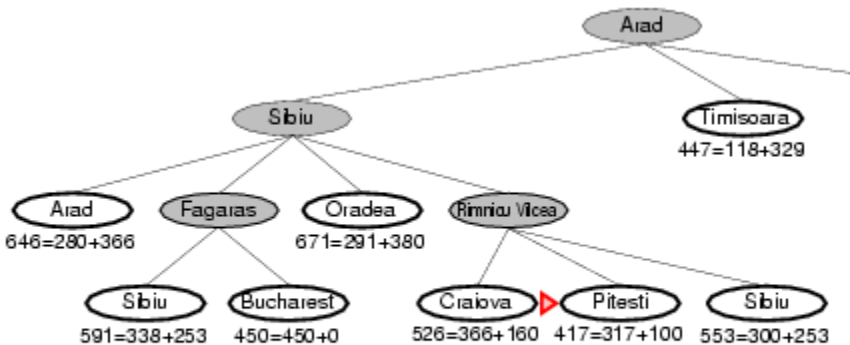


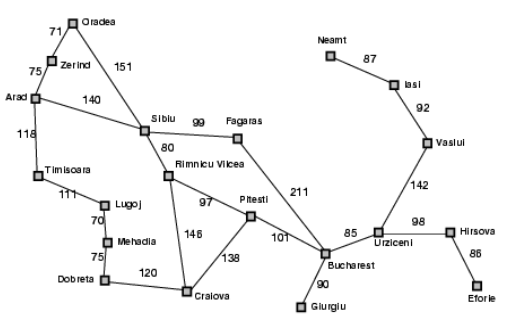
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vashui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Search A*

- Initial State: Arad; Objetive: Bucharest; f(n) = g(n) + h(n);
- g(n) = cost until n; h(n) = straight line distance to the goal



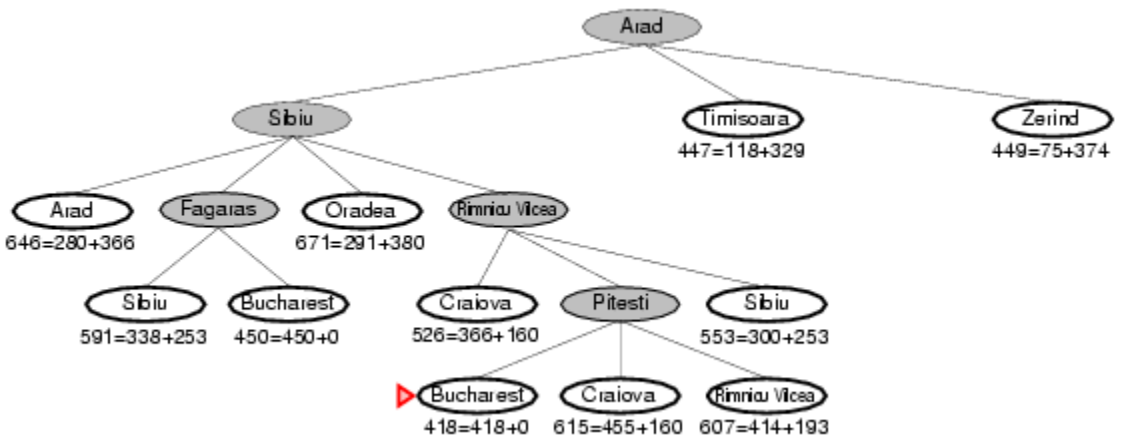


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Search A*

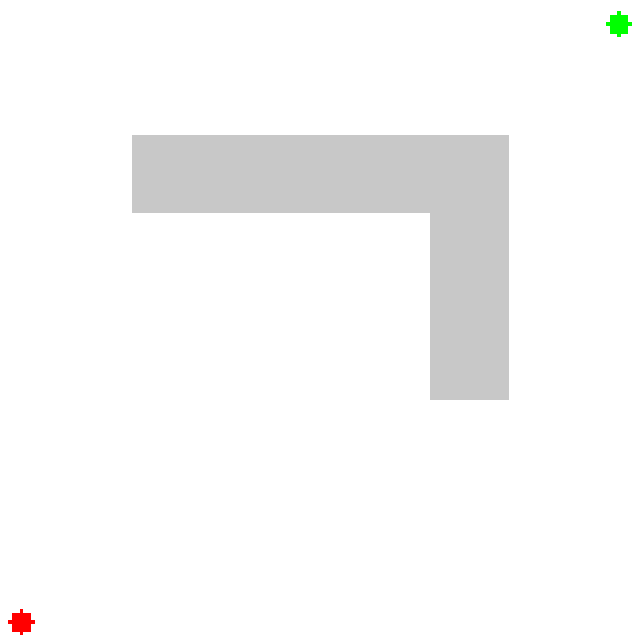
- Initial State: Arad; Objetive: Bucharest; f(n) = g(n) + h(n);
- g(n) = cost until n; h(n) = straight line distance to the goal



Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

ISLab
Synthetic Intelligence Lab

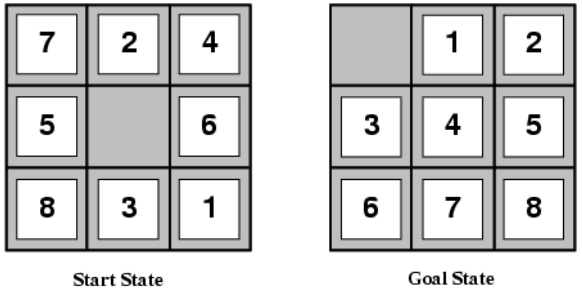
Search A*
Example (2)



Source:
https://en.wikipedia.org/wiki/A*_search_algorithm

Heuristic Functions

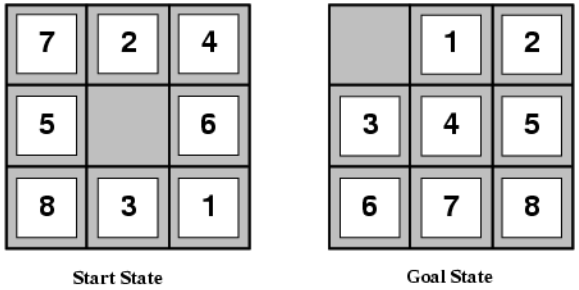
- **Heuristics - 8 Puzzle**
  - Typical solution in 20 steps with an average branching factor: 3
  - Number of states: 320 = 3.5*109
  - No. of states (without repeated states) = 9! = 362880
  - Heuristics:
    - $H_1(n)$ = No. of parts out of place
    - $H_2(n)$ = Sum of the distances between the parts and their correct positions



Source: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

Heuristic Functions

- **Heuristics - 8 Puzzle (2)**
  - Problem Relaxation as a way of inventing heuristics :
    - A piece can be moved from A to B if A is adjacent to B and B is empty
    - a) Piece can move from A to B if A is adjacent to B
    - b) A piece can move from A to B if B is empty
    - c) Piece can move from A to B



Source: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

Search Strategies

- **Limited Memory Search - IDA*/SMA***
- IDA* - Iterative Deepening Search
  - Strategy: Use of a limit cost in each iteration and iterative depth-first search
  - Problems in some real problems with cost functions with many values
- SMA* - Simplified Memory Bounded A*
  - IDA* from one iteration to the next only remembers one value (the limit cost)
  - SMA* uses all available memory, avoiding repeated states
  - Strategy: When it needs to generate a successor and has no memory, it forgets the node in the queue that appears to be unpromising (with a high cost).

Source: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

Summary

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Algorithm	Complete	Optimal	Time complexity	Space compelxity
First in Width (BFS)	Yes	If all escalated costs are equal	$O(b^d)$	$O(b^d)$
First in Depth (DFS)	No	No	$O(b^m)$	$O(bm)$
Iterative	Yes	If all escalated costs are equal	$O(b^d)$	$O(bd)$
Uniform Cost	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
Greedy	No	No	In the worst scenario : $O(b^m)$ Best case scenario : $O(bd)$	
A*	Yes	Yes (if the heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	

ISLab
Synthetic Intelligence Lab

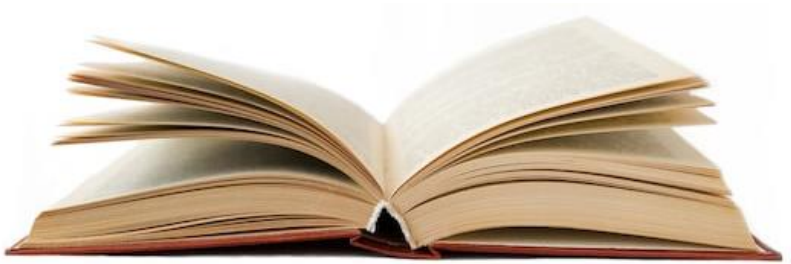
References

Recommended Bibliography

- Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594.
- E.Costa, A.Simões (2008), Inteligência Artificial-Fundamentos e Aplicações, FCA, ISBN: 978-972-722-340-4, 2008.

Other material

- Svetlana Lazebnik, Lecture notes Fall 2017 Artificial Intelligence, University of Illinois.
- Luís Paulo Reis, Lecture notes Artificial Intelligence (2019), Universidade do Porto



Universidade do Minho
Escola de Engenharia
Departamento de Informática

PROBLEM-SOLVING AND SEARCH METHODS

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial
2025/26