Instituto Superior de Engenharia de Lisboa

Artificial Intelligence for Autonomous Systems

André Fonseca

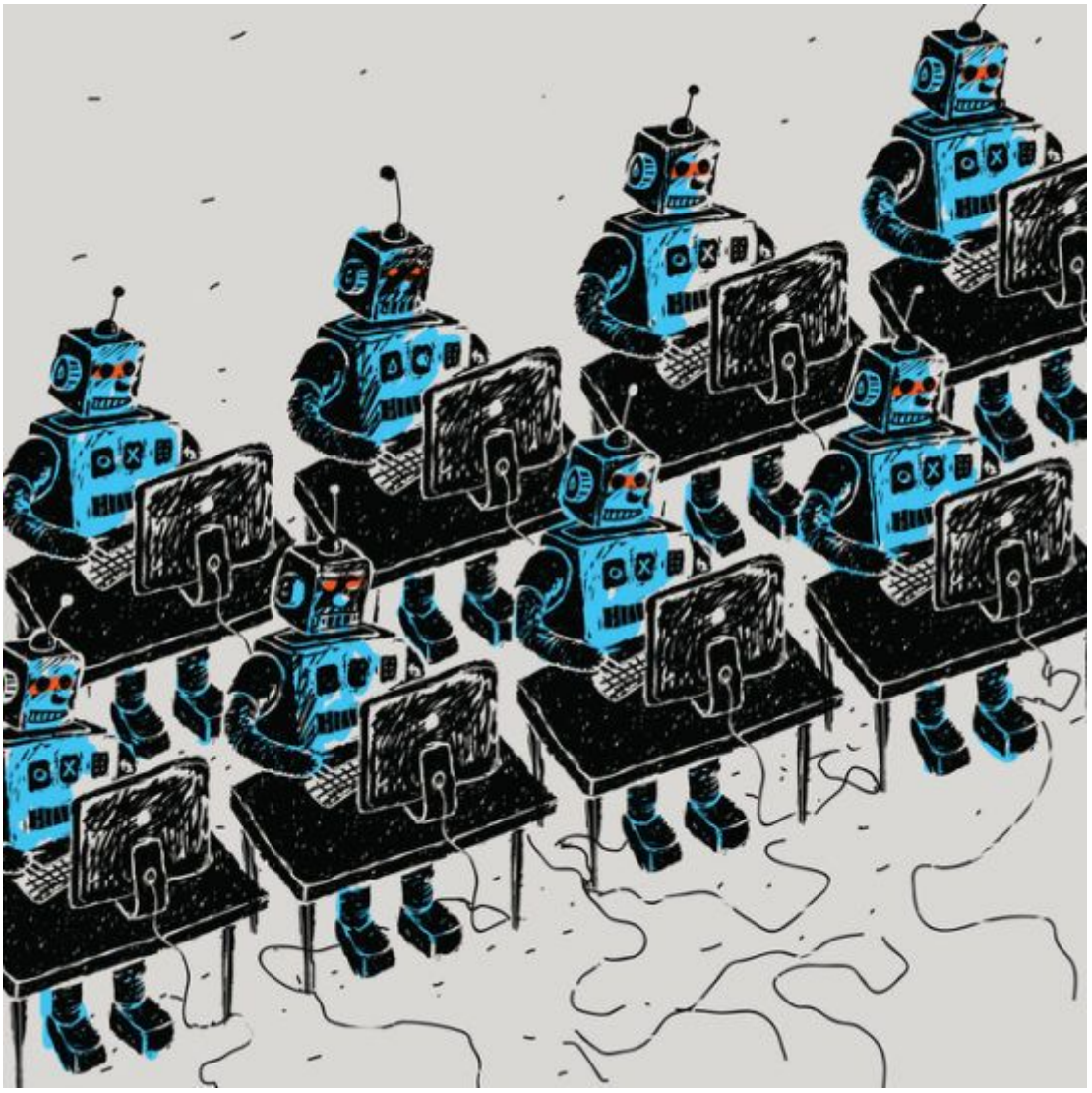
A39758@alunos.isel.pt

Professor Eng. Paulo Vieira

pjvieira@deetc.isel.pt

July 2017

Contents



"We tend to overestimate the effect of a technology in the short run and underestimate the effect in the long run."

Roy Amara

Introduction

This final report aims to describe and explain all the subjects studied in the Artificial Intelligence for Autonomous Systems class. During the semester three projects were developed with the provided architectural system models.

The first project simulates the interactions between an agent and an environment which the agent belongs to. When the simulation starts the gatekeeper goes into patrol mode. In case of hearing a noise the gatekeeper inspects the area, searching for the noise's origin. If there is no longer any noise, then he continues to patrol. At any time the gatekeeper may encounter an enemy and in that case he must protect the area and warn the foe to retreat. In this situation, if the enemy backs off, he will remain inspecting the area for any source of noise. In case the enemy persists, the gatekeeper will have to defend himself and fight the threat. In combat situation two actions may occur. The gatekeeper can be victorious and defeat the enemy and, then, he goes back to patrol; or in case of defeat the simulation is restarted. The interaction, at any level, is made by text inputs and outputs.

The second project implement an autonomous resolution system of the 8 puzzle problem. The initial problem presents two puzzle configurations to be solved, but the main objective is to develop a solution that is able to solve any kind of puzzle configuration or, even further, any kind of problem that fits the developed problem model. The puzzle consists of an area divided into a grid of 3 by 3. On each grid square is a tile, except for one square which remains empty. Thus, there are eight tiles. A tile that is next to the empty grid square can be moved into the empty space, leaving the previous position empty in turn. Tiles are numbered, 1 to 8, so that each tile can be uniquely identified. The puzzle is solved using multiple search algorithms such as bread-first search, depth-first search and other variants.

The third and last project (...)

Before going into the projects analysis and resolutions (...)

https://github.com/andrewfonseca/IASA

Artificial intelligence overview

AI has always intrigued people, from TV shows, movies and marketing using robots with pseudo-intelligence. It is the perfect topic to craft thrillers, adventures and dramas, just like the well known Terminator movie or the brilliant Westworld TV show.

The most basic definition of intelligence is: the ability to learn and solve problems[1]. This is regardless of being human intelligence or machine intelligence. So artificial intelligence could be described as the intelligence exhibited by machines or software [2]. To be more specific it's the study and design of intelligent agents - a system that perceives its environment and takes actions that maximizes its chances of success. In general an intelligent agent should be able to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize and adapt to new situations[3].

Difference between Human and Machine intelligence

What are the differences between Human and Machine Intelligence? - The answer to this question may be vast and may change in the future, when new technologies or biological processes are discovered. Right now the main differences are:

- Humans perceive by patterns whereas a machine perceive by a set of rules and data;
- Humans store and recall information by patterns, machines do it by searching algorithms;
- Humans can figure out the complete object even if some part of it is missing or distorted; whereas the machines can only do it by searching through out all possible solutions.

[1] Definition of intelligence - Merriam-Webster

[2] Definition of artificial intelligence, Wikipedia

[3] Preparing for the future of artificial intelligence - Executive Office of the President National Science and Technology Council Committee on Technology

Definition throughout history

As history goes on the definition of AI has changed and four approaches mastered their own definition.

Thinking humanly

The cognitive approach in which machines are designed with minds, machines that think in the full and literal sense. It requires scientific theories of internal activities of the brain and how humans think.

Acting humanly

Machines have to act and do things like humans. This was the definition proposed by Alan Turing in the well known Turing Test: where a computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or a computer. The major components suggested by Alan Turing were: knowledge, reasoning, language understanding and learning.

Thinking rationally

AI is made using math merged with logic. It uses some notations and rules of inference or derivations for thoughts to codify all knowledge in this representation and use it to derive new knowledge.

Acting rationally

Intelligent systems are designed to act and to maximize their goal. The intelligent agents are expected to maximize the goal achievement giving the available information about the environment, the background knowledge, past experiences and other useful information.

This last approach is the one adopted in the study objectives of IASA class.

Applications

AI is a very hyped-up topic for a reason, it has the ability to revolutionize all industries and in the edge, the world. AI is being used by virtual assistants, like Google Now, Siri and Amazon Echo, in speech recognition; To translate sentences or entire documents from one languages to another; In all types of robots, such as robotic surgery and navigation; Enhancement of medical imaging to analyze patient's exams and detect diseases; Autonomous vehicles are already in use, these type of vehicles can drive autonomously without the intervention of a human. The possibilities are endless.

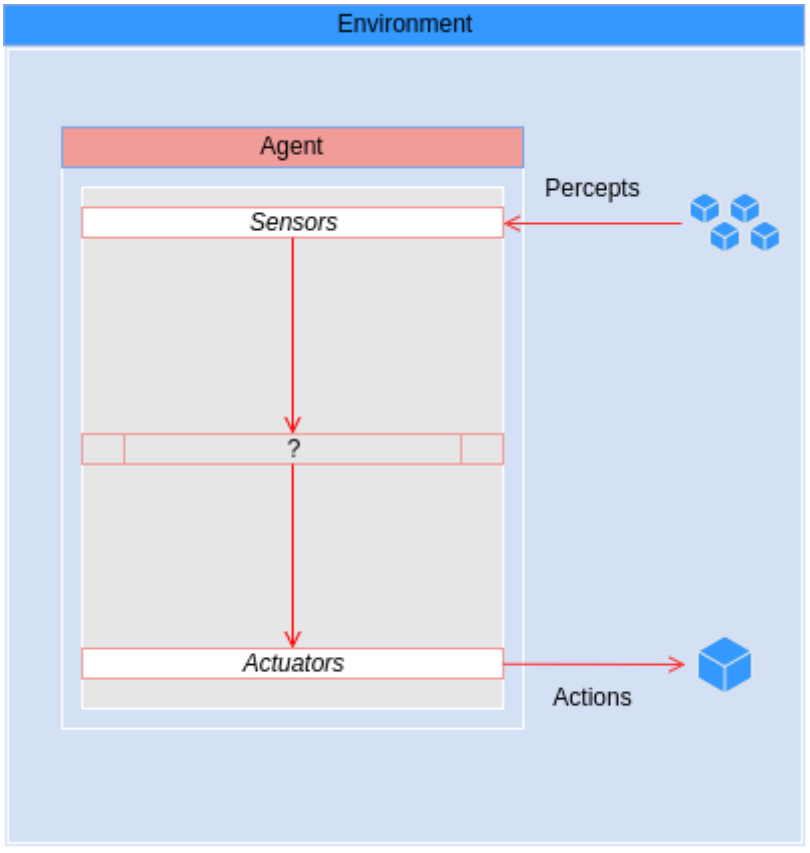
Intelligent agents

An **agent** is anything that can perceive its environment through sensors and act upon that environment through actuators. It can be seen as a function ‘f’ that goes from something that he perceives ‘P’ to a set of actions ‘A’.

$$f : P \rightarrow A$$

These generated actions are a byproduct of thinking and deliberating given on what its observed or sensed from the environment - the cycle or the loop of an agent.

A human agent has eyes and ears as sensors and hands, legs, mouth and other body parts as actuators while a robotic agent would have cameras and motion sensors and multiple motors as actuators. Examples of robotic agents: thermostats, smartphones, self-driving cars, vacuum cleaners.



Agent properties

The properties of a rational agent can be grouped under the acronym PEAS (Performance, Environment, Actuators, Sensor). These definitions are actually the problem specifications, for the task environment that the rational agent is meant to solve:

- Performance: Which qualities it should have?
- Environment: Where it should act?
- Actuators: How will it perform actions?
- Sensors: How will it perceive environment?

Considering the PEAS for a self-driving car: The performance could be the safety of the passengers, obedience of driving rules (stopping at red lights or driving under speed limit for example); the environment is all the roads, other cars, pedestrians, road signs; the actuators are everything that makes the car move, steering, brakes; the sensors include a camera, a sonar, a GPS, a speedometer.

Rationality

An agent should do its best given what it is observing and past experience and knowledge the agent has. A rational agents is an agent that does the right thing and rationality is relative to how to act in order to maximize a performance measure - factor that describes how well the agent is doing when performing a certain task to reach a certain goal. An agent must then figure out the actions or a set of actions that lead to the goal while considering the impact of its actions on future states.

> *"For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has."*

Nature of environments

Agents can operate in a multitude of environments. These environments need to be categorized in order to assess the agent's sensors and actuators.

- **Fully observable** (vs partially observable): An agent's sensors give it access to the complete state of the environment at each point in time;
- **Deterministic** (vs stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. On the opposite there is a non-deterministic environment, or stochastic, in which uncertainty about outcomes is modeled or quantified in terms of probabilities;
- **Episodic** (vs sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself;
- **Static** (vs dynamic): The environment is unchanged while an agent is deliberating. It can also be called semi-dynamic if the environment does not change with the passage of time but the agent's performance score does;
- **Discrete** (vs continuous): A limited number of distinct, clearly and well defined percepts and actions;
- **Single agent** (vs multiagent): An agent operating by itself in an environment;
- **Known** (vs unknown): The designer of the agent may have knowledge about the environment.

Search agents

There are four basic types of agents in order of increasing generality: simple reflex agents, mobile-based reflex agents, goal-based agents and utility-based agents. All of which can actually be generalized to learning agents that can improve their performance and generate better actions.

Simple reflex agents

Simple reflex agents choose their actions only based on the current percept. These type of agent lack rationality because the input percept is directly connected to an output action with a condition-action rules map. Their main disadvantages is that the environment must be completely observable and if they are deprived from one condition the agent might not know what to do unless it selects a default action.

Model based reflex agents

Model-based agents use a model of the world to choose their actions, keeping an internal state. This model is based on how the world evolves and how the agent's actions affect the world.

Goal-based agents

Goal-based agents choose their actions in order to achieve goals. This approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

Utility-based agents

Utility-based agents choose actions based on a preference (utility) for each state with regard to the production of a good outcome. Sometimes achieving the desired goal is not enough, so things like probability of success, the resources needed to execute the scenario, the importance of the goal to be achieved, the time it will take, might all be factored in to the utility function calculations.

Learning agents

This type of agent was purposed by Alan Turing[4] where an agent is assembled with learning capabilities so that it can learn on his own or with assistance. This method allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.

A learning agent can be divided into four conceptual components:

- Learning element: responsible for making improvements - learning with past experience;
- Performance element: responsible for selecting external actions;
- Critic: how well is the agent doing in comparison to a fixed performance standard;
- Problem generator: allows the agent to explore different possibilities and keep on learning.

[4] Computing machinery and intelligence (1950) - Alan Turing

Agent architecture

* Internal representation of reactive and deliberation architectures.

* BDI architecture

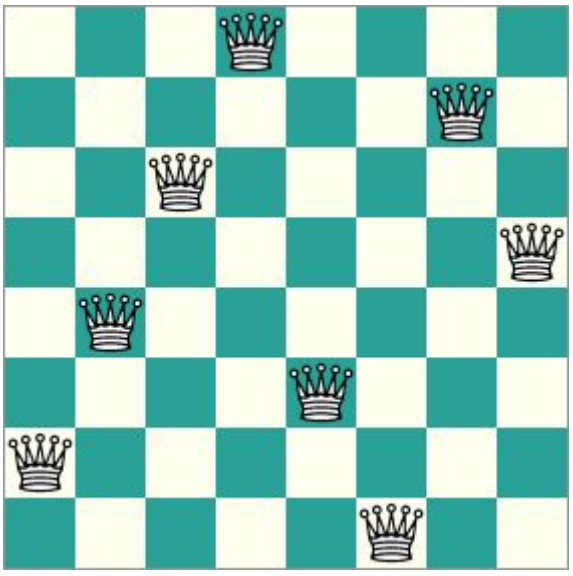
* http://transectscience.org/pdfs/vol1n1/1118_35.pdf

* http://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr008.pdf

* http://www.dca.fee.unicamp.br/~gudwin/courses/IA889/2014/IA889-05.pdf

Search strategies

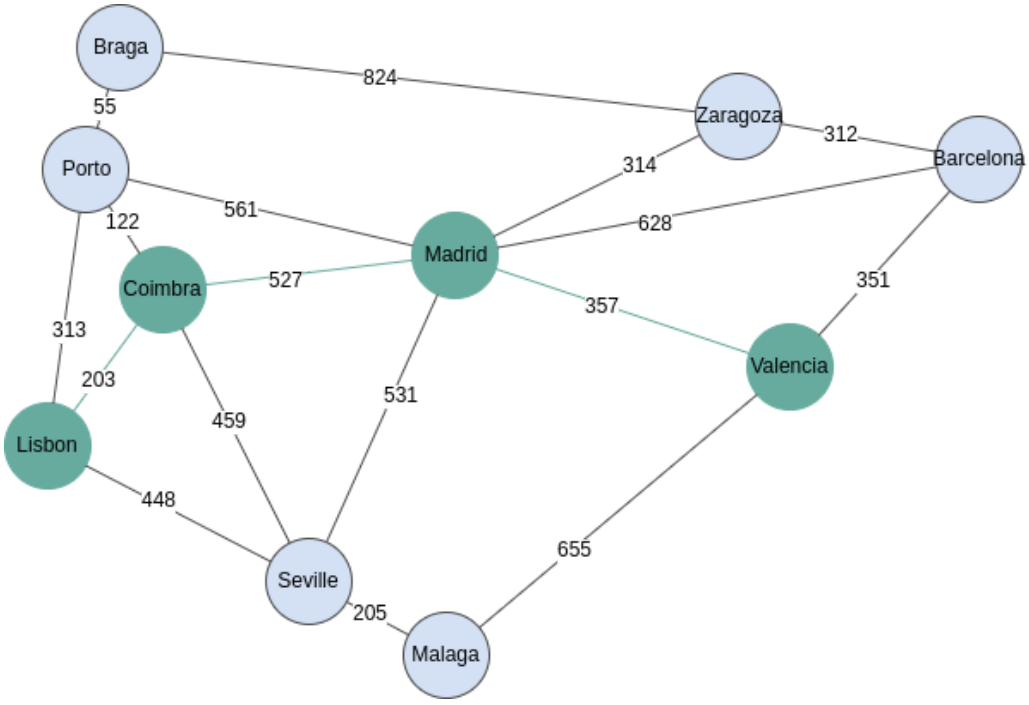
An action or set of actions can be called a path, that will lead to a certain goal and paths may come in different costs or depths. Paths are explored using uninformed search strategies or informed search strategies.



The 8 queen problem is a typical problem where search strategies are used to reach the goal of placing 8 queens, on a chess board, so that no queen is attacking any other horizontally, vertically or diagonally. The algorithm will search for possible configurations where no queen attacks another one. The number of possible sequences is:

$$64 \times 63 \times 62 \times ... \times 57 = 1.8 \times 10^{14}$$

Another practical use is route finding between two cities. For example to go from Lisbon to Valencia there are several routes that can be chosen, but which one is the optimal route? This problem can be analysed considering a map represented by a graph in which the nodes represent cities and the edges represent the roads.



There are multiple ways on how to perform a search to find the optimal route; once the optimal solution is found then the agent can execute the path: The optimal route is Lisbon → Coimbra → Madrid → Valencia.

Problem solving as search can be defined through:

1. What is the goal to achieve
2. Problem formulation

Problem solving is a two stage process:

1. Search: "mental" exploration of several possibilities
2. Execution of the solution found

Problem formulation

A problem can be broken down to multiple components:

- Initial state: the state in which the agent starts;
- States: all the states reachable from the initial state by any sequence of actions;
- Actions: possible actions available to the agent. At a state `s`, `actions(s)` returns the set of actions that can be executed in state `s`;
- Transition model: a description of what each action does `results(s, a)`;
- Goal test: determines if a given state is a goal state;
- Path cost: function that assigns a numeric cost to a path with respect to some performance measure. It doesn't always have to be used but whenever there is a performance measure associated with the agent this function can be applied to know how much a sequence of actions costs. This is useful to evaluate the cost of multiple solutions to the same problem.

State space vs search space

The state space is typically the physical configuration where the different state in which problem is evolving. In the 8 queen problem this would represent all possible boards. In contrast, a search space is an abstract configuration of the problem usually represented by a search tree or a graph.

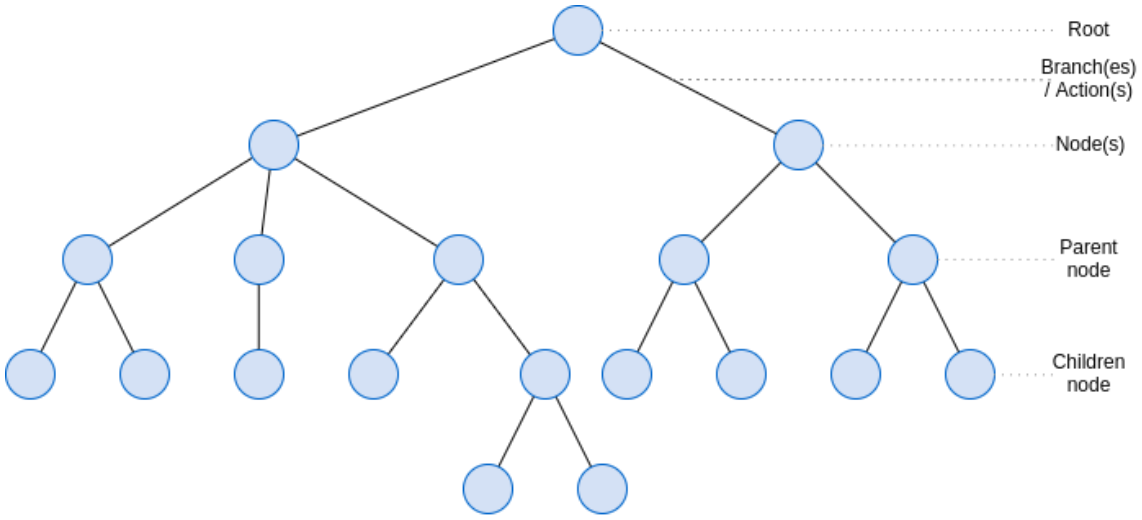
Search tree composition:

- Root: initial state;
- Branches: actions;
- Nodes: results from actions.

Node composition:

- Parent;
- Children;
- Depth;
- Path cost;
- Associated state in the state space;

Finally, a function called `expand` is defined. This function is responsible for creating all children nodes given a certain node.



Search algorithm

The search space is divided into three regions:

1. Explored: Represent all the nodes that have already been visited in the search tree;
2. Frontier: nodes that are about to be explored;
3. Unexplored: the remaining nodes yet to be explored.

The essence of search strategies is to decide in which order the nodes will move from Unexplored → Frontier → Explored.

```
function graph_search(initial_node, goal_test)

      frontier = Set(initial_node)
      explored = Set()

      while frontier is not empty:
            node = frontier.remove()
            explored.add(node)
            state = node.state()

            if goal_test(state):
                  return Success(state)

            for children in node.childrens():
                  if children not in frontier or explored:
                        frontier.add(children)

      return Failure()
```

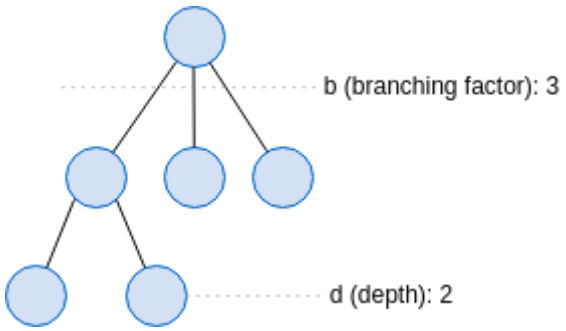
Dimensions

Search strategies are evaluated along the following dimensions:

- Complete: does it always find a solution if one exists?
- Time complexity: number of expanded nodes
- Space complexity: maximum number of nodes in memory
- Optimal: does it always find a least-cost (optimal) solution?

Time and space complexity will be measured in terms of:

- ‘b’: maximum branching factor of the search tree (actions per state);
- ‘d’: depth of the solution;
- ‘m’: maximum depth of the state space (can be infinite).



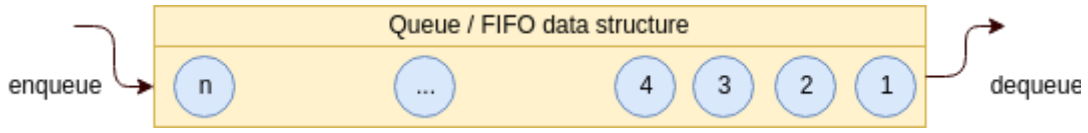
There are two kinds of search. An uninformed search strategy is when there is no information about the problem's domain. In informed search strategies there is information about the goal and the problem's domain in general.

Uninformed search

Breadth-first search

In breadth-first search (BFS) the shallowest node is expanded first. The shallowest node is the one on the top level-wise. The search goes level by level only when the current level is all done.

The BFS algorithm is similar to the graph search algorithm except that the frontier's data structure will be a queue or a FIFO (First In First Out).

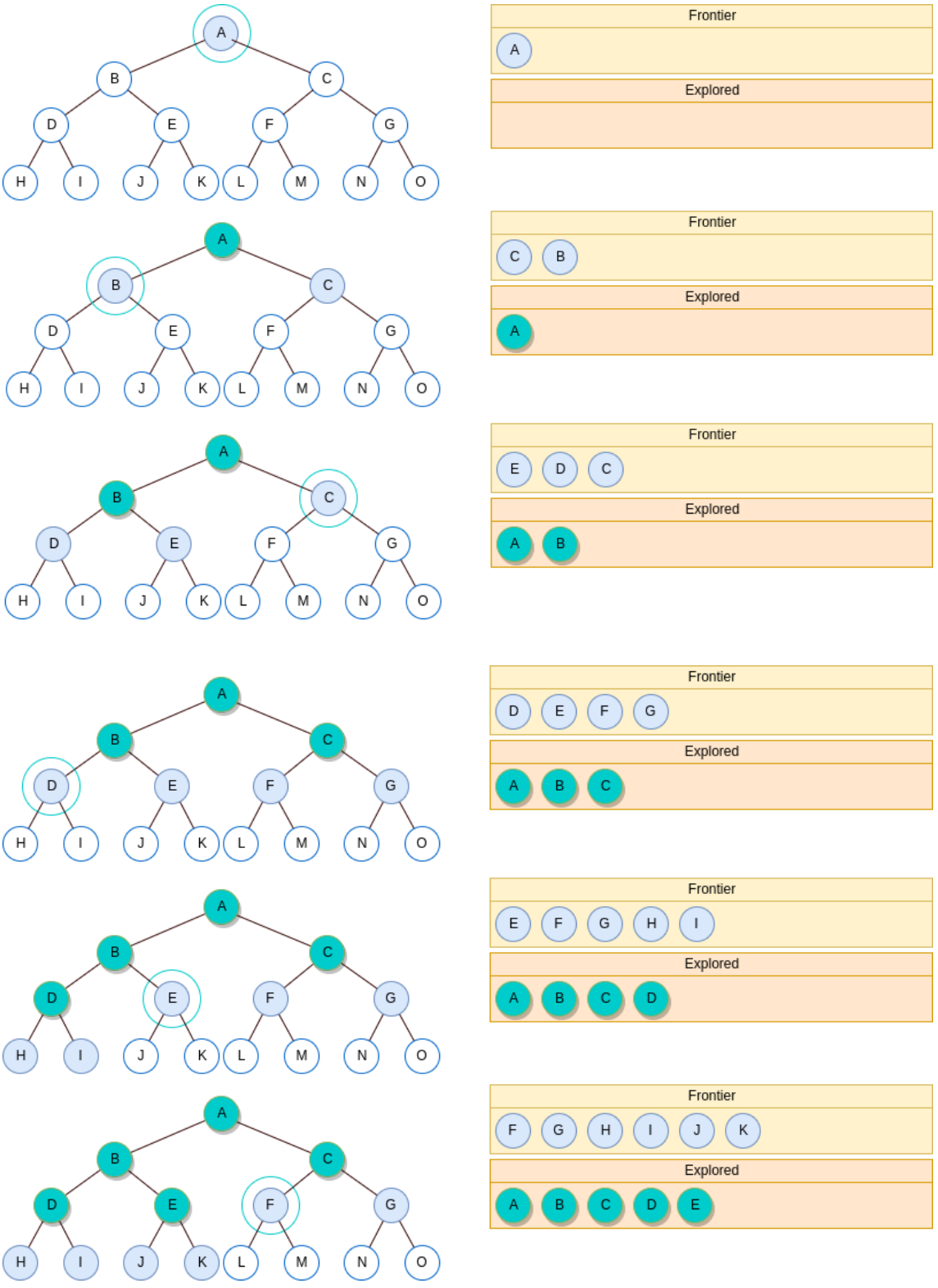


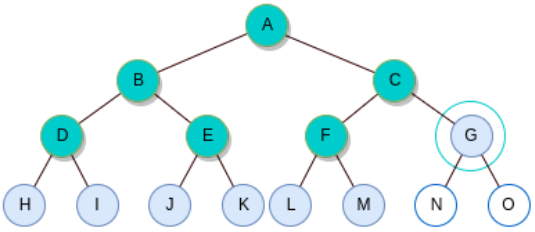

```
function breadth_first_search(initial_node, goal_test)
      frontier = Queue(initial_node)
      explored = Set()

      while frontier is not empty:
            node = frontier.dequeue()
            explored.add(state)
            state = node.state()

            if goal_state(state):
                  return Success(state)

            for children in node.childrens():
                  if children not in frontier or explored:
                        frontier.enqueue(children)

      return Failure()
```

Complete	Yes if b is finite.
Time	$1 + b + b^2 + b^3 + ... + b^n = O(b^d)$
Space	$O(b^d)$
Optimal	Yes if cost is 1 per step.

If time and space complexities are exponential, so why even use BFS? The answer to this question depends on the problem. For problems with domain knowledge and where the solution is shallow it is much better to use BFS, it will be found sooner and

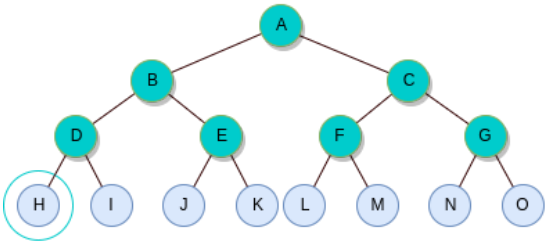
the exponential complexity will be much shorter. For other kind of problems BFS is not suitable. Memory requirement and exponential time complexity are the biggest handicaps.





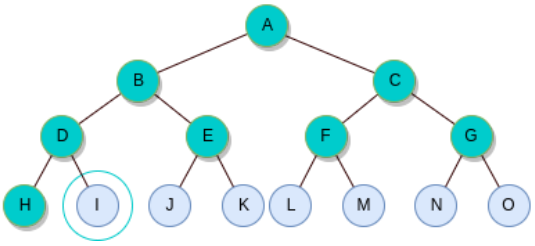
Frontier							
G	H	I	J	K	L	M	

Explored							
A	B	C	D	E	F		



Frontier							
H	I	J	K	L	M	N	O

Explored							
A	B	C	D	E	F	G	

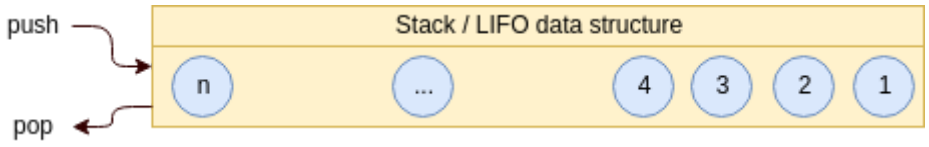


Frontier							
I	J	K	L	M	N	O	

Explored							
A	B	C	D	E	F	G	H

Depth-first search

Depth-first search (DFS) explores the deepest node first, as deep as possible along each branch before backtracking.

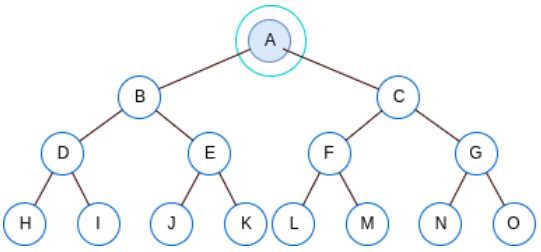



```
function depth_first_search(initial_node, goal_test)
      frontier = Stack(initial_node)
      explored = Set()

      while frontier is not empty:
            node = frontier.pop()
            explored.add(state)
            state = node.state()

            if goal_state(state):
                  return Success(state)

            for children in node.childrens():
                  if children not in frontier or explored:
                        frontier.push(children)

      return Failure()
```

Complete	No in infinite-depth spaces and yes in finite spaces, both need to avoid node loops.
Time	$1 + b + b^2 + b^3 + ... + b^m = O(b^m)$
Space	$O(b \cdot m)$ Linear space complexity because it needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path, hence the ‘m’ factor.
Optimal	No.

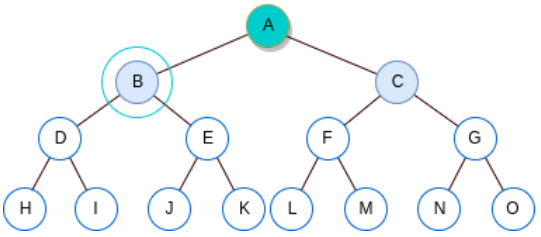
The time complexity is still exponential, just like regular BFS. The reason for that is because the search not only goes to the depth of the solution but to the depth of ‘m’, which is the maximum possible depth of the search space. In terms of memory

usage it will be much smaller compared to BFS, due to linear space complexity. Regular DFS should not be used when dealing with infinite search spaces.



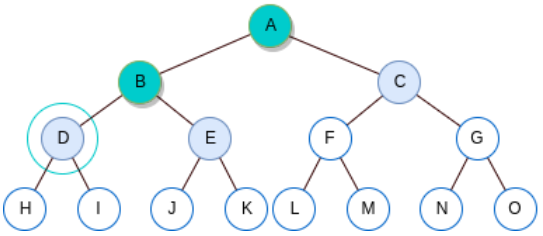
Frontier: A

Explored:



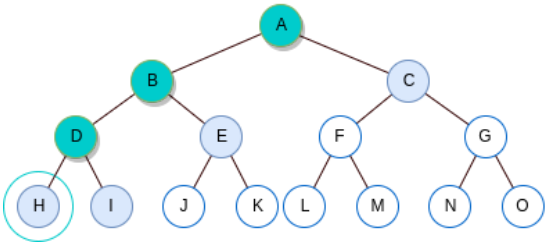
Frontier: B C

Explored: A



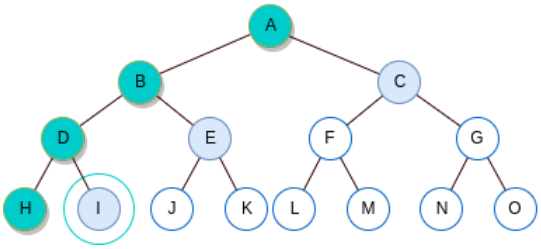
Frontier: D E C

Explored: A B



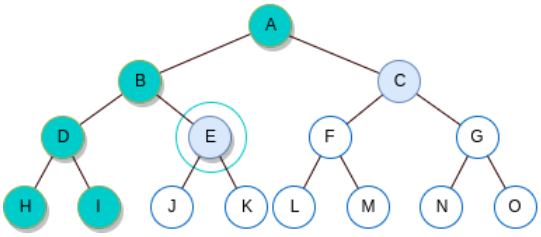
Frontier: H I E C

Explored: A B D



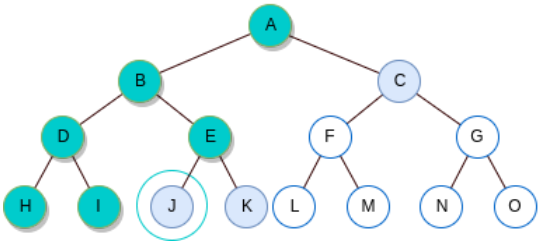
Frontier: I E C

Explored: A B D H



Frontier: E C

Explored: A B D H I

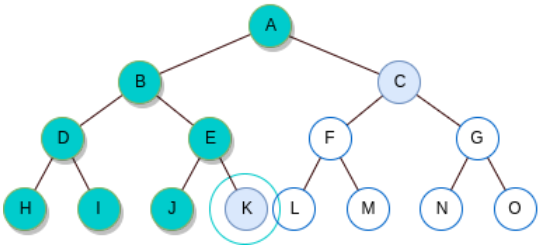


Frontier

J K C

Explored

A B D H I E

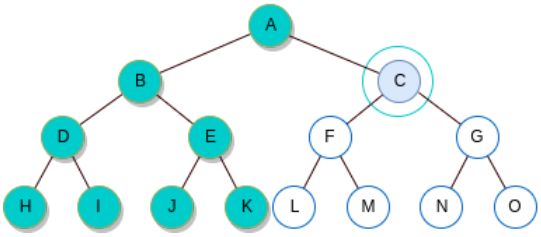


Frontier

K C

Explored

A B D H I E J



Frontier

C

Explored

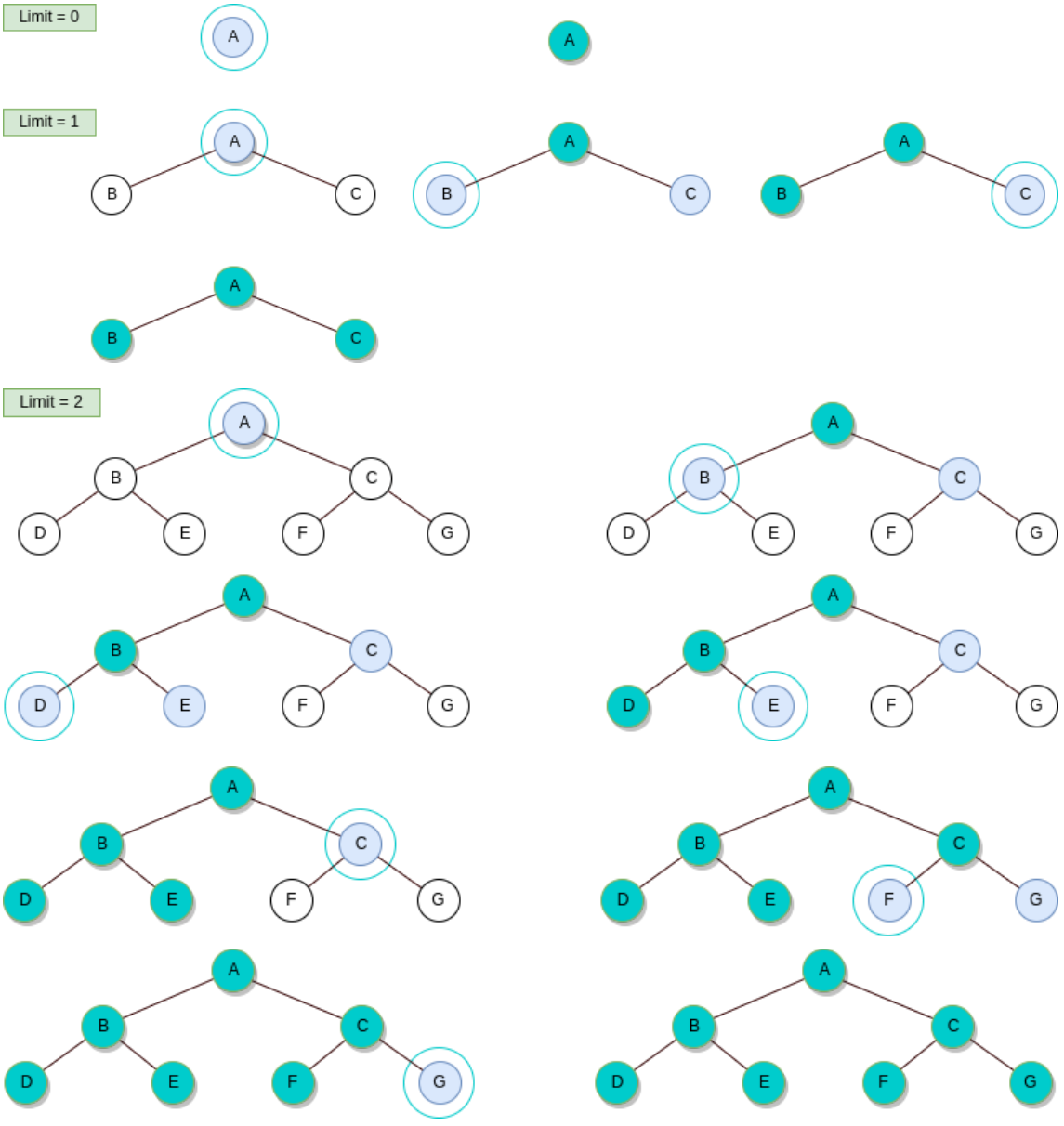
A B D H I E J K

Iterative deepening search

Iterative deepening is an extension of depth-limited search that actually combines the benefits of BFS and DFS.

The concept of this search is to iteratively increase the search limit until the depth of the shallowest solution `d` is reached or if the depth is within the max possible limit. Every time the limit is increase the search is restarted and this is not a big waste because most of the nodes are at the bottom of the search tree.

This type of search is particular useful when there is domain knowledge about the problem so that it's not needed go full depth of the search tree.



Uniform-cost search

Usually a graph search has some weight or cost between node connections. The uniform-cost search takes advantage of this information to find the cheapest solution, this is done by modifying BFS to prioritize by cost rather than prioritizing by depth. This means that nodes with the lowest path cost `g(n)` will be expanded first. The frontier data structure used is a heap - a priority queue.

```
function uniform_cost_search(initial_node, goal_test)
      frontier = Heap(initial_node)
      explored = Set()

      while frontier is not empty:
            // pick element with minimum cost
            node = frontier.delete_min()
            explored.add(state)
            state = node.state()

            if goal_state(state):
                  return Success(state)

            for children in node.childrens():
                  if children not in frontier or explored:
                        frontier.insert(children)
                  else if children in frontier:
                        // decrease the cost because a shorter path
to n has been found
                        frontier.decrease_key(children)

      return Failure()
```

Complete	Yes if solution has finite cost
Time	$O(b^{C^*/\varepsilon})$
Space	
Optimal	

Comparing uninformed search strategies

- Breadth-first search expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space and time complexities;
- Depth-first search expands the deepest node first. It is neither complete nor optimal, but has linear space complexity;
- Iterative deepening search calls DFS with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to BFS, and has linear complexity.
- Uniform-cost search expands the node with lowest path cost, `g(n)`, and is optimal for general step costs;

Informed search strategies

Greedy best-first search

A* search

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

Comparing informed search strategies

- Greedy best-first search expands node with minimal *'h(n)'*. It is optimal but is often efficient.
- A* search expands nodes with minimal *'f(n)'*. It is complete and optimal, provided that *'h(n)'* is admissible or consistent.

Markov decision process

Finding optimal policy

Reinforcement learning

In reinforcement learning agents learn from reinforcement or delayed reward.

Learning approaches for decision making in situation where outcome are stochastic, involves an agent that continued to plan and learn to effect its environment.

RL agents are driven by maximizing the reward on the long run.

SARSA algorithm

Q-Learning algorithm