

Assignment 2

Subject

Topics of this session :

1. Uninformed Search.
2. Formalise a problem as a graph search problem.
3. Start-Goal search problems.

This assignment is graded and must be submitted (individually) on Moodle before next week's class.

For each exercise, detail your reflexion steps :

- We are mostly interested in your actual thinking process.
- Even if you are unable to solve an exercise, write out what were your reflexion steps.
- For each attempted exercise, a written feedback will be provided.

Reference material : [Artificial Intelligence, A Modern Approach, Chapter 3.1-3.5.](#)

For coding :

- [Noto](#) (Online Jupyter NoteBook).
- Any other python coding environment you prefer using.

Exercise 1

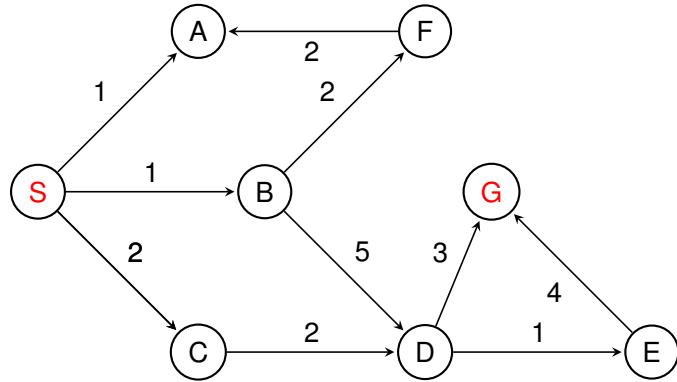


FIGURE 1 – A weighted directed graph.

We want to determine a path between the starting node **S** and the goal node **G**, using the graph shown in Figure 1. Each edge has a weight representing its cost.

1. For each of the following search algorithms, list :
 - The order in which nodes are visited.
 - The final path found from **S** to **G**.

- The total cost of this path.

Apply the following algorithms :

- (a) **Depth First Search (DFS)**
- (b) **Breadth First Search (BFS)**
- (c) **Uniform Cost Search (UCS)**

Important : If there is a need to break ties between nodes, always expand them in **alphabetical order**.

For example, if the algorithm reaches nodes **A**, **B**, and **C**, it should expand **A** first.

2. Analyse the algorithms :

- Are these algorithms **optimal**? Why or why not ?
- Are they **efficient**? Explain your reasoning.

Exercise 2

Based on the type of problem given in Exercise 1, suggest an alternative search algorithm.

Implement your proposed algorithm using Python. (An example code is given in the last section of the pdf).

Exercise 3

Imagine you are in Neuchâtel and want to travel to Geneva Airport by train. The journey involves passing through train stations in the cantons of Neuchâtel, Vaud, and Geneva.

1. Formalise the problem as a **start-goal search problem** :

- What are the **states** in this problem ?
- What is the **initial state** ?
- What is the **goal state** ?
- What are the possible **actions** available at each state ?
- What are the **costs** associated with moving between states ?

2. Model the problem as a weighted graph.

3. Implement the problem as a graph in Python and use one of the search algorithms from the previous exercises to find an optimal path from Neuchâtel to Geneva.

Example implementation of BFS

Below is an implementation of Breadth-First Search (BFS) to use as a starting point :

```

# We represent a weighted graph as a dictionary where:
# keys are nodes
# values are the children of their key node.
graph = {
    # Below means node S can lead to both A (with a cost of 3)
    # and B (with a cost of 2)
    'S': [ ('A', 3), ('B', 2)],

    'A': [ ('C', 1)],
    'B': [ ('D', 4)],
    'C': [ ('G', 2)],
    'D': [ ('G', 1)],
    'G': []
}

def bfs(graph, start, goal):
    # the queue contains all the paths we can expand upon.
    # here we store for each element of the queue:
    # - the current node to expand from
    # - the full path
    # - the cumulated path cost
    queue = [(start, [start], 0)]
    visited = set() # set to track visited nodes

    while queue:
        # we want to pick the node which comes first in alphabetic order
        queue.sort()
        node, path, cost = queue.pop(0) # get and remove the first element

        # we found the goal, return the path
        if node == goal:
            return path, cost

        if node not in visited:
            visited.add(node)
            for neighbor, weight in graph.get(node, []):
                # create a new path including the neighbor
                new_path = path + [neighbor]
                # enqueue new path with updated cost and new expandable node
                queue.append((neighbor, new_path, cost + weight))
    return None # No path found

print(bfs(graph, start='S', goal='G')) # prints the path found by BFS

```