Student: Dăscălescu Dana
Group: 507 - Artificial Intelligence
Date: April 20, 2023

# Informed Search

**A***

## 1 Project Overview

The objective of this project is to create an application that utilises an Informed Search algorithm. To provide a basis for comparison, we have chosen to use the same application from Project 1 and evaluate the performance of the search algorithms.

## 2 Problem description

In classroom 308, students taking the Artificial Intelligence exam are seated in rows of two-person benches arranged in three columns. Some spots are free, meaning there are no students occupying those seats. Carina wants to help her friend Oscar cheat on the exam by sending him the answers through a message. However, there are restrictions on how messages can be passed. Students can only pass the message to the bank colleague behind or in front of them, or the one right beside them, but not diagonally. Additionally, passing messages between rows is challenging since the teacher can easily notice it. Only the penultimate and final benches on each row can be used for message transfer. To ensure she does not get lost in the classroom, Carina wants to include on the note the path it needs to take from one colleague to the next. To make matters worse, some students are upset with each other or disagree with cheating. This may make them unwilling to participate in the message transfer, or they will report it to the TAs if they notice it happening. Therefore, Carina needs to be cautious about selecting trustworthy colleagues who are willing to help her.

## 3 Solution

### 3.1 Problem formulation

The current configuration of the classroom, in which students are arranged in a maze pattern with two-person benches arranged in three columns, and the current limitations make it difficult for students to communicate with one another. This problem can be viewed as a graph, with each student representing a node and the edges representing the possible message paths between them. However, the constraints of the problem limit the validity of certain edges. Specifically, only the last and second-to-last benches in each row may be used for message transmission, and passing messages between rows presents a significant obstacle due to the vigilant supervision of the teacher. Furthermore, some students' reluctance to participate in message transfer may make it difficult, and it's not always possible to transfer messages within the columns.

To solve this problem, we can formulate it as an informed search problem, which consists of an **initial state**, a set of **actions**, a **transition model**, a **goal test**, and a **path cost** function. The initial state represents Carina's location, while the set of actions represents the transfer of the message from one student to another. The transition model takes into account the constraints of the problem, such as the limitations on available neighbours and the presence of uncooperative students. The goal test checks if the current state represents Oscar's location. Finally, the path cost function represents the number of steps required to transfer the message from Carina to Oscar.

To find the shortest path, we can apply an informed search algorithm such as `A*` to navigate through the search space efficiently. By using an informed search algorithm, we can reduce the time and computational resources required to find the optimal solution.

## 3.2 Search Strategy

### 3.2.1 A*

The A* algorithm is an informed search algorithm that efficiently finds the shortest path in graph search problems. It uses a heuristic function that estimates the cost of the cheapest path from the current state to the goal state in addition to the actual cost of the path from the initial state to the current state. In our problem of finding the shortest path for transferring a message in a classroom, the A* algorithm can be used to navigate the search space and find the optimal solution more efficiently.

In the context of our problem, the A* algorithm will be used to find the shortest path from Carina's location to Oscar's location, while taking into account the constraints and limitations imposed by the classroom's configuration. The heuristic function used in this case can be the Manhattan distance or Euclidean distance between two nodes in the graph, which provides an estimate of the distance to the goal. The actual cost of reaching a node can be calculated by counting the number of steps required to transfer the message from one student to another.

By using the A* algorithm, we can navigate through the search space more efficiently, as it considers both the actual cost of reaching a node and the estimated cost to reach the goal and prunes unpromising nodes early. As a result, we can find the optimal solution with less computational resources and time compared to other uninformed search algorithms. The effectiveness of the A* algorithm in solving our problem will be evaluated in subsequent sections.

### 3.2.2 Heuristic function(s)

To ensure that the A* algorithm finds the minimum-cost path to a goal node, several conditions must be met, including having a finite number of successors for any node in the graph, edges with weights greater than a positive quantity µ, and an admissible heuristic function. Admissibility means that the heuristic function can never overestimate the cost of reaching the goal.

Admissibility is a requirement that ensures that the A* algorithm will always return an optimal solution. However, consistency is a stricter requirement than admissibility. A heuristic function is consistent if the estimated cost of reaching a goal node from a given node is less than or equal to the sum of the cost of reaching a neighbouring node and the estimated cost of reaching the goal node from that neighbouring node. In other words, if $h(n)$ is the estimated cost of reaching the goal from node $n$ and $c(n, m)$ is the cost of travelling from node $n$ to neighbouring node $m$, then a heuristic function $h$ is consistent if $h(n) <= c(n, m) + h(m)$ for all nodes $n$ and $m$.

This section focuses on the heuristic functions that can be used in our problem and their properties. We will explore three admissible heuristics and justify our selection of the Manhattan distance as the most suitable for the current scenario:

All three of the heuristic functions given as examples - *Manhattan distance*, *diagonal distance*, and *Euclidean distance* - are admissible. Moreover, they are also consistent, which means that they satisfy the stricter requirement for optimality. Therefore, using any of these heuristic functions in the A* algorithm will ensure that it always returns the minimum-cost path to a goal node. But considering all the aspects, we choose the Manhattan distance as our admissible heuristic since it is relevant to the movement constraints in our problem and provides an accurate estimation of the distance to the goal node.

## 4 Implementation

We will solve this problem using an informed search algorithm: A*. The implementation follows the guidelines described in Balcan and Hristea [1] and Russell [2]:

1. The A* algorithm starts by initializing an empty list called `CLOSED` and adding the start node to a list called `OPEN`.

2. It sets a boolean variable called `scope_node_found` to *False*

3. While the `OPEN` list is not empty, the algorithm performs the following steps:

   3.1 Extract the first node, `current_node` from the OPEN list and add it to the CLOSED list.

   3.2 If the `current_node` is the goal node, set `scope_node_found` to *True* and stop the search.

   3.3 Expand the `current_node` by obtaining its successors in the graph. Exclude successors that are already on the path from the start node to the `current_node`.

   3.4 For each successor node, calculate its $g$ and $f$ cost values.

   3.5 For successors that are not in the `CLOSED` or `OPEN` sets, create a new node, set its parent to `current_node`, and calculate its $g$ and $f$ cost values. Insert it into the `OPEN` list in a way that it remains ordered in ascending order based on $f$. If there are two nodes with the same $f$ value, the node with the greater $g$ value is placed first.

   3.6 If a node was inserted into the `OPEN` list in the previous step, repeat the sorting process for the OPEN list.

4. When the search is finished, write the results to a file, including the path from the start node to the goal node, the cost of the path, and the execution time.

The code provided defines classes for the implementation of an A* algorithm to solve a problem of message passing between students sitting in a classroom.

The *‘**Configuration**‘* class defines the state of the problem, which includes the arrangement of children in a seating matrix, the position of the child with the message to transmit, and methods to compute the Manhattan distance and equality of two configurations.

The *‘**Node**‘* class represents a node in the search graph for the A* algorithm. Each node contains a configuration object and a heuristic function h, which estimates the cost of reaching the goal state from the current state.

The *‘**Edge**‘* class represents a directed edge between two nodes, which contains the starting and ending nodes and the cost of the edge, which is 1 in this case.

The *‘**Problem**‘* class defines the problem to be solved, which includes a list of nodes, a list of edges, the starting node, and the goal node. The problem class also provides methods to search for a node by configuration and to add a node to the list of nodes.

The *‘**NodeTraversal**‘* class contains information associated with a node in the open and closed lists used by the A* algorithm. It includes a reference to the node in the graph, properties specific to the A* algorithm, such as the heuristic evaluation functions f and g, and an estimate of the depth of a node in the graph.

***Output:***

```
———————————————————————————————— Conclusion ————————————————————————————————
Minimum cost path:
[ oscar  > diana  v tasha  v simona  v felix  v nikita  >> charlotte  ˆ tana
    > patricia  v victor  carina  ]
Execution time: 0.0025745000000000073
```

## 5 Performance analysis

When considering search algorithms, it is important to analyse their performance in terms of memory usage and execution time. In this context, A* search, breadth-first search (BFS), and depth-first search (DFS) are three common algorithms used for problem solving.

In terms of time execution, our experiments indicate that A* search is the fastest algorithm, with an execution time of 0.0020 seconds, followed by BFS and DFS, which took 0.8292 seconds and 0.8277 seconds, respectively. This finding is consistent with the fact that A* search is an informed search algorithm that utilises heuristics to guide the search towards the goal state, which can result in faster execution times compared to BFS and DFS.

Regarding memory usage, all three algorithms have similar memory requirements, with BFS and DFS using slightly more memory (40.86 bytes and 40.87 bytes, respectively) compared to A* search (40.46 bytes). This is expected as BFS and DFS require the maintenance of a queue and a stack, respectively, which can result in higher memory usage compared to A* search.

In conclusion, based on our experiments, A* search is the most efficient algorithm in terms of time execution, while all three algorithms have similar memory requirements. However, it is important to note that the performance of each algorithm can vary depending on the specific problem being solved and the characteristics of the search space.

## References

[1] Maria Florina Balcan and Florentina Hristea. *Aspecte ale Cautarii si Reprezentarii Cunostintelor in Inteligenta Artificiala*. Editura Universitatii din Bucuresti, 2004.

[2] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.