MiniProyect 1 – Heuristic function

Alejandro Cirugeda K-5087

Juan Carlos Gómez Quintana K-5101

Summary

The objective of this assignment is to design a program that could find the solution of the "Binario" Game using two search space algorithms in a designing tree, and compare their performances on the given puzzle. The two algorithms that we are going to use are Breadth-first algorithm and A* algorithm and a heuristic function.

A. Rules of Binario

Binario is played on a square board with no specific size but has to have the same number of columns and rows. When the game starts, some of the cells will be already filled with "0" or "1" tokens.

The objective of this game is to fill the whole board with these tokens, but following some rules:

- Each cell must only contain one token, "0" or "1".
- There can't be more than two tokens of the same value next to or below each other.
- All rows and columns must contain the same number of "0" or "1" tokens.
- Each row and column must be different from the other ones.

Each binario puzzle does only have one solution.

B. Searching and puzzle space

In our project, the decision graph's nodes are 6x6 matrices. Each matrix can contain 3 numbers:

- -1 : This represents that a cell is empty.
- 0 : It represents that the cell has a "0" token.
- 1 : It represents that the cell has a "1" token.

This is our puzzle space.

The "initial_state" matrix corresponds to the graph's root node, which has the initial configuration of the board. While the "solution" matrix corresponds to terminal node that we want to reach with our algorithm and has the configuration of the final board.

Our searching space is formed by paths through the decision graph, which start from the root node.

C. Heuristic function

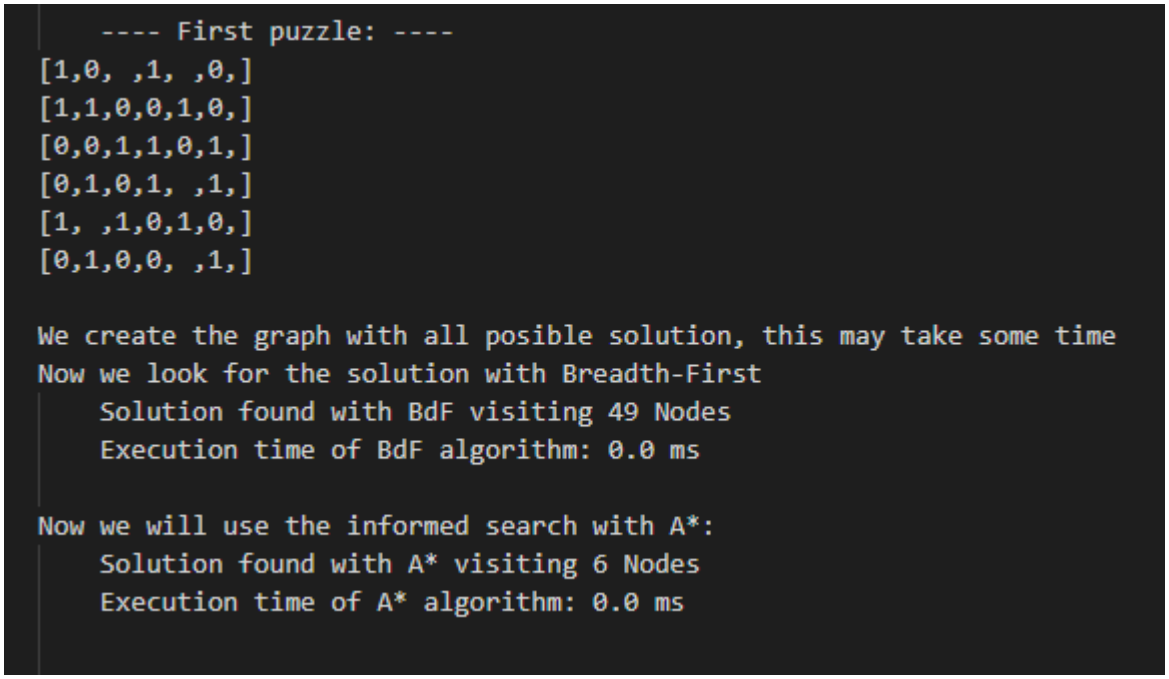
In our project we have defined g(x) (the cost of getting from the initial state to the current state) as the depth of the current node in the graph. Moreover, we have defined our h(x), heuristic function, as the number of cells which are filled with a wrong token or are empty.

In our case, our heuristic function is admissible because the objective function of every node, whose formula is f(x) = g(x) + h(x), is always smaller or equal to the value of g(x) in the node which has the desired board (terminal node).

Furthermore, we can observe that the heuristic function is also monotonous as the value of the objective function never increases while it approaches to the terminal node.

D. Results

After seeing the results of both algorithms, we can conclude that A* is better and faster than Breadth-First because the heuristic function is admissible and monotonic. This means that A* will find the solution after a finite number of visited nodes which is no greater than the cost of the function of Breadth-First. We can see these differences in the following pictures:



On the first puzzle we have 5 blank spaces at the beginning of the game, the more blank spaces, the more nodes the algorithm has to visit. Here, we can appreciate the difference between both algorithms: while Breadth-First algorithm visits 49 nodes, A* visits only 6.

```
    ---- Second puzzle: ----
[0, ,1, ,0,1,]
[1, , ,0,1,0,]
[ ,1,1, ,1,0,]
[ , ,0,1, ,1,]
[0, ,1, ,0,1,]
[1, ,0,1, ,0,]

We create the graph with all posible solution, this may take some time
Now we look for the solution with Breadth-First
    Solution found with BdF visiting 11337 Nodes
    Execution time of BdF algorithm: 28.984 ms

Now we will use the informed search with A*:
    Solution found with A* visiting 14 Nodes
    Execution time of A* algorithm: 0.0 ms
```

In the second puzzle we start with 13 blank spaces, so the nodes visited by both algorithms increase.
In this performance we can see more clearly the difference between them.

```
    ---- Third puzzle: ----
[0, ,1,0, ,0,]
[0,1,1,0, ,1,]
[1,0, ,1, , ,]
[0, ,0, , , ,]
[1, ,1, ,0,1,]
[ , , , , ,0,]

We create the graph with all posible solution, this may take some time
Now we look for the solution with Breadth-First
    Solution found with BdF visiting 235282 Nodes
    Execution time of BdF algorithm: 5202.069 ms

Now we will use the informed search with A*:
    Solution found with A* visiting 18 Nodes
    Execution time of A* algorithm: 0.0 ms
```

In the third puzzle we start with 17 blank spaces.
Again, the difference is huge.