Multi-output maze solving using Prolog

Department of Computer Science: University of Turin

Autori: Alessandro Saracco, Leonardo Magliolo , Mattia Marra

Authors: Alessandro Saracco, Leonardo Magliolo , Mattia Marra

Artificial intelligence and laboratory

A.Y. 2021/2022

Maze generation

Maze generation was performed by a script in Python considering the following inputs:

- Number of rows
- Number of columns
- Initial position
- Final positions

Walls are generated randomly; the probability of a wall being placed within a specific box (except for boxes assigned to initial and final positions) has been set equal to a constant between 0% and 50%.

Choice algorithm

A* was chosen as the algorithm to be implemented in Prolog considering the following characteristics:

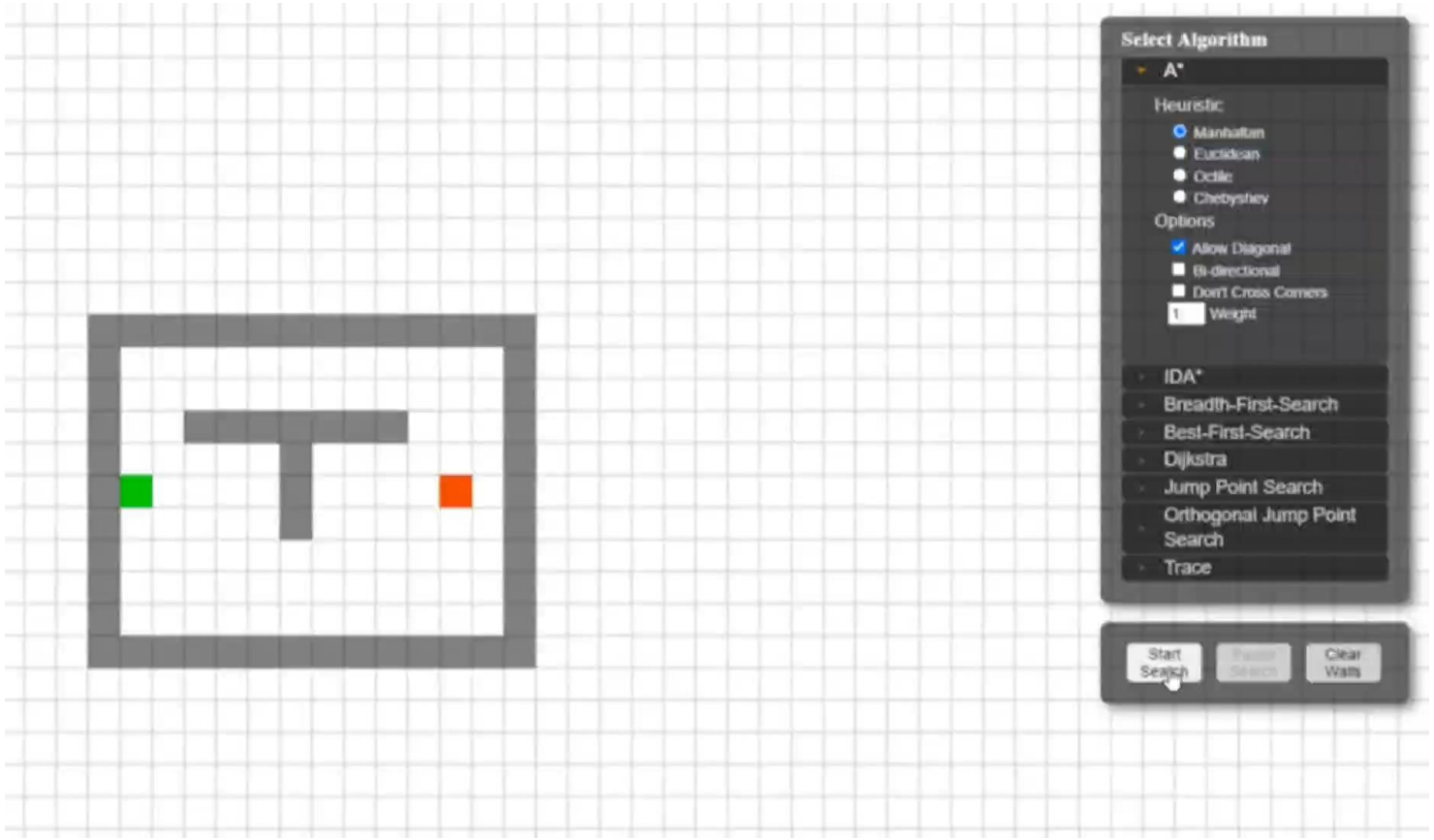
- Conduct informed research
- Complete in the domain under consideration
- If the heuristics used for informed search is monotonic then it is optimal
- There is no other algorithm that guarantees fewer nodes expanded to find an optimal solution

Choice algorithm

Prior to the implementation of A*, research was conducted about effectiveness of the Possible algorithms applied to the reference problem.
The following site was particularly helpful in order to compare A* and IDA*.

Given the machine on which the algorithm was run, the size of the problem that was chosen to be addressed (no more than 1000x1000 mazes) and possible timing due to the testing phase, it was chosen to implement A* rather than IDA*.

IDA* vs. A* example with Manhattan heuristics



Choice of heuristics

As a result of the research, it was chosen to implement two versions of A*,
one Manhattan heuristics and one with Euclidean distance.
They were implemented because they have been widely studied in the literature relation to solving the multi-output maze problem.

The chosen heuristics are not monotonic in the considered search space (since it is possible to move diagonally with only one action), so the calculated solutions
from the implemented algorithm turn out to be suboptimal.

Performance analysis: Maze types

For execution of the performance analysis performed, it was chosen to run the algorithm On square mazes having side lengths: 25, 50, 100, 500 and 1000.

Three maps were generated for each side listed above:

- one without walls with solutions (referred to as NM)
- one with walls with solutions (denoted as MS); the probability that a wall is entered within a given box is 50%.
- One with characteristics identical to the one indicated in the previous point but without solutions (referred to as NS)

Performance Analysis: Hardware Specifications

Hardware specifications:

- CPU: AMD Ryzen 7 5800X
- RAM: Corsair VENGEANCELPX16GB DDR4
- SSD: Samsung MZ-V8V1T0 980

Performance analysis: Data collected

A* with Manhattan heuristics

Labyrinth	Time	Branching Factor	Depth	Cost	Out of Stack
25 x 25 MS	< 1 s	29.94	35	1048	False
25 x 25 NM	< 1 s	13.66	24	328	False
25 x 25 NS	< 10 s	/	/	/	False
50x50 MS	< 1 s	8.65	55	476	False
50x50 NM	< 1 s	13.83	49	678	False
50x50 NS	< 3 m	/	/	/	True 10 GB
100x100 MS	< 1 s	10.77	103	1110	False
100x100 NM	< 1 s	13.91	99	1378	False
500x500 MS	< 2 s	14.03	516	7240	False
500x500 NM	< 1 s	13.98	499	6978	False
1000x1000 MS	< 40 s	26.34	1065	28060	True 1GB, False 8 GB
1000x1000 NM	< 4 s	13.99	999	13978	False

A* with Euclidean distance heuristics

Labyrinth	Time	Branching Factor	Depth	Cost	Out of Stack
25 x 25 MS	< 0.5 s	19.5	34	663	False
25 x 25 NM	< 0.5 s	12.91	24	310	False
25 x 25 NS	< 2 s	/	/	/	False
50x50 MS	< 1 s	14.8	51	755	False
50x50 NM	< 0.5 s	13.1	49	642	False
50x50 NS	> 5 m	/	/	/	True 10 GB
100x100 MS	< 1 s	12.43	99	1231	False
100x100 NM	< 0.5 s	13.16	99	1304	False
500x500 MS	< 2 s	17.43	500	8716	False
500x500 NM	< 1 s	13.22	499	6604	False
1000x1000 MS	< 4 s	11.66	999	11649	False
1000x1000 NM	< 2.5 s	13.24	999	13229	False

Performance analysis: Considerations on results

It can be seen from the data collected that the Euclidean distance heuristic is not significantly more advantageous than the Manhattan distance heuristic for most the reduced-sided maps considered in testing. It turned out to be particularly advantageous, however, in the the search inside in mazes whose side amounts to 1000, both in terms of timing and memory usage.
In the case of the 1000x1000 MS maze, a solution is found in less than four seconds by drawing on less than one GB of RAM; Manhattan heuristics instead, the solution computation took about 40 seconds and 8 GB of RAM at peak acquisition.

Performance analysis: Considerations on results

Although the A* algorithm performed reasonably well about the timing with which the solutions were computed, it is evident that in mazes where the solution was not present the amount of RAM used to
node expasion was inadequate with respect to the machine that performed it, even for rather small maps.

In cases where a given maze causes stack overflows with A*, to determine whether a solution, exists it would make more sense to use algorithms that reduce the expanded nodes present in the frontier, such as IDA*.

Performance analysis: Considerations on results

Setting the side of the map equal to l then the depth the deepest branch explored would be <= of l^2.

By setting the branching factor equal to 8 (number of moves that can be made in the case worst), then the maximum number of nodes retained in memory by A* would be $\in \mathcal{O}(8^l)$ while in the case of IDA* the same amount would amount to at . $\mathcal{O}(l^2)$