International Journal of
Science and Engineering Investigations vol. 8, issue 92, September 2019
Received on September 6, 2019 ISSN: 2251-8843

Tailored Dijkstra and Astar Algorithms for Shortest Path Softbot Roadmap in 2D Grid in a Sequence of Tuples

Erenis Ramadani[1], Festim Halili[2], Florim Idrizi[3]
[1,2,3]Department of Informatics, Faculty of Math and Natural Sciences, University of Tetova, Tetovo, North Macedonia
([1]erenisramadani@gmail.com, [2]festim.halili@unite.edu.mk, [3]florim.idrizi@unite.edu.mk)

Abstract- Artificial Intelligence conveying the Data Mining and Machine Learning are the top trends recently that have been researched, not only in the world of IT. Shortest path problem is one of the most commonly used algorithms in several solutions, and through this paper, we are trying to provide additional information about the technologies, methods and their accuracy in implementing shortest path algorithms, in this case – Dijkstra and Astar algorithm for finding the shortest path. Both algorithms, Dijkstra and Astar, are used in this paper, trying to provide a comparison between their performance and accuracy in solving the shortest path between multiple nodes in a given 2D grid, where nodes are represented by tuples, which in fact are a given location in the grid

Keywords- *Dijkstra, A*, Astar, Algorithms, Shortest Path*

I. INTRODUCTION

The data flow era we are facing lately, is bringing a lot of challenges amongst all of its advantages. Managing, storing or studying these data are just some of the challenges enterprises are facing. According to [1], 90% of the total amount of the data in the world has been generated in the last two years. And we are a live proof of how fast this is still growing. But, luckily, most of these data are used for improving human life in order to make it easier by trying to provide everything a human being needs in everyday life. As a result, for example, we recently have autonomous cars: all it takes is the driver to set the destination, and the car, in a much safer and faster way, will send him there. Although this is still in "testing" mode, and most of us are not yet comfortable trusting a machine doing "what we have to do", in a very short period of time, this will become a reality. Considering these improvements, we can easily state that the requirements for efficient production and living have also improved, and thus, the shortest path problem is becoming a hot topic to study and research.

The shortest path problem is a very well-known and even better treated by many researchers, and in this paper, we will try to use the same in order to provide a better notion of what in fact shortest path problem includes, how it is treated and try to provide exemplary solution for the new researchers of the field.

In this paper, we have treated a very interesting problem, which, in fact, came due to a professional challenge, which generally required us to build a pizza bot, which, in a given 2D grid, and a sequence of tuples (pairs) representing locations in the grid, would deliver pizza in each location given through the sequence of tuples. The solution is done using Python as programming language, and Dijkstra and Astar as algorithms.

There are several reasons why we chose Python instead of any other language, and one of them is because it is the most common language used for problems of this field, there are multiple ready-to-use libraries for AI problems, which make working in these kind of problems easier. Dijkstra and A* also have their reasons: Dijkstra is one of the first algorithms in the field of shortest path problem, is the most common algorithm used for research purposes, and A* is very similar to Dijkstra, in fact, it is derived from Dijkstra, with additional properties added during the calculation.

Following in this paper, you will be able to read more about each algorithm separately and the programming language, which is represented in the second section of the paper.

II. PRINCIPLES OF DIJKSTRA ALGORITHM

Dijkstra's Algorithm is an algorithm for finding the shortest paths between nodes in a graph. It was conceived by computer scientist Edsger W. Dijkstra in 1956. The algorithm exists in many variants, but the original one finds the shortest path between two nodes. [2]

Originally, Dijkstra's algorithm does not use the priority queue. Since all nodes have the same priority, it runs in O (V2) time complexity, where V is the number of the nodes. [3]

In a given chain of nodes, as shown in Figure 1, let A be the initial node, where searching for the shortest path should begin. Let F be the destination, where searching for the shortest path should end. Dijkstra's algorithm assigns a tentative distance value to each node of the chain: it sets 0 for the initial node, and infinity for the rest of the nodes. The infinite value does not imply that there is a possible infinite distance, but to make sure that later on, we know which nodes have been checked, and which haven't (there are some variant implementations that does not label the value at all). There should be a set that will

keep track of the nodes that are already checked, which initially contains only the initial node and another one of those which are unchecked yet. In the next step, for each unvisited neighbor of the current node is calculated the tentative distance value, which consists of the (distance to current node + distance from current node to the neighbor). If the result is less than their current tentative distance, the new distance is set as tentative distance. When all of the neighbors are considered and evaluated, the current node is removed from the unchecked set of nodes to the checked set. The algorithm finishes when the destination node is marked as checked.

Let us use the chain of nodes represented on Figure 1 as an example to illustrate how the algorithm works. Let us use Table 1 to track the path, which consists of three columns: nodes, their distance from the initial node, in this case – node A, and the previous node. After the algorithm is finished, Table 2 will be generated.

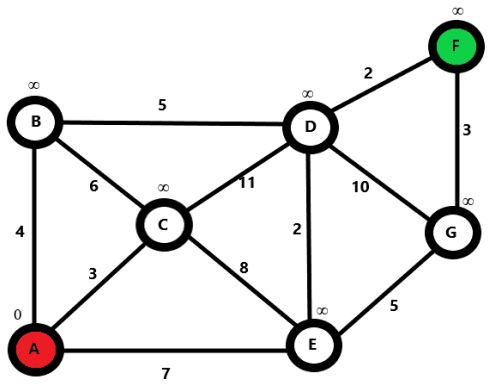


Figure 1. Sample Graph of Nodes

TABLE I. THE TABLE BEFORE CALCULATING THE DISTANCE

Nodes	Shortest distance from node A	Previous node
A	0	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/

As seen in the illustration, A has three neighbors: B, C and E. Initially, the distance to each of them is calculated, and the smallest value is considered to be their tentative distance, until next iteration. In this case, the next values for B will be 4, C will be 3 and E is 7. Since C is the smallest value, C is added in the checked set, and the same step is repeated here: C has three neighbors: B, D and E. The distance from C to B is 9, but since the previous was 4, this is disregarded. The distance from C to D is 14, and since the previous value for D is infinite, the new distance to D is 14. E is disregarded also, since the distance from C is higher than the previous. The next smallest distance that is not added in the checked set is B. There is only one option from B: to D, since C is already marked as checked. The distance from B to D is 9, and thus, the new distance for D is 9 since the previous is 14. The same steps are repeated until the destination node is reached and added in the checked set.

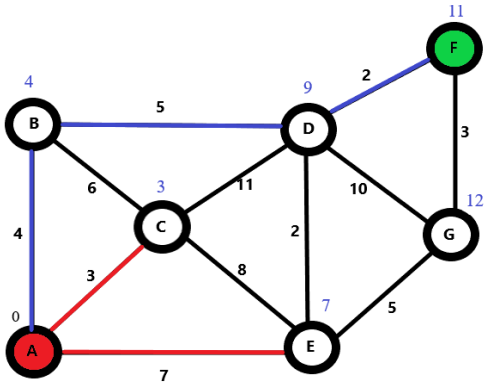


Figure 2. The path that algorithm follows during execution, where the red lines represent every try, and the purple line is the shorteset path.

In the end, the following table is generated, containing all the distances and their predecessor node.

TABLE II. GENERATED VALUES AFTER DIJKSTRA ALGORITHM IS EXECUTED

Nodes	Shortest distance from node A	Previous node
A	0	/
B	4	A
C	3	A
D	9	B
E	7	A
F	11	D
G	12	E

III. PRINCIPLES OF A* ALOGORITHM

A* pronounced as “A star” is an algorithm widely used in the process of finding the shortest path between two nodes. It was first published by Peter Hart, Nils Nilsson and Bertram Raphael in 1968. A* achieves better performance by using heuristics to guide its search. It’s a best first search algorithm formulated in terms of weighted graphs, starting from a specific starting node to find the destination node having the smallest cost (shortest time).

The heuristics is the sum of two functions: G which represents the exact cost of the path from the initial node to the current one and H which is an admissible but not overestimated cost to reach the goal node. [4][5]

Selecting the heuristic function is a very important factor that affects the performance of A*. If the cost of H is equal to the cost necessary to reach the target node, we say that H in this case is ideal, and it would always follow the perfect path.

In another case, when H is overestimated, it may happen that the path to the target node is found faster, but the cost is not optimal. And is some particular cases, it may come to a scenario that, despite that the path exists; the algorithm fails to find one. And when the chosen H is underestimated, the algorithm will find the best possible path to the target node. An implication of this would be that, the smaller the value of H, the longer will take to find the path: in worst case scenario, for H = 0, the algorithm will provide the same results as Dijkstra's algorithm. [2]

The time complexity of A* depends on the heuristics too. In the worst case scenario, when the algorithm is dealing with an unbounded search space, the expansion of the nodes is exponential in the depth of the solution d: O (bd), where b is the branching factor (the average number of the successors per state). In this case is assumed that a goal node exists and is reachable from the start node. Otherwise, the algorithm will not terminate. [6]

The A* algorithm creates and maintains two lists, one open and another closed. In the open list is the priority queue also, since it contains and keeps track of the nodes that are still to be examined, and the closed list keeps track of the nodes that have already been examined. [7]

The following snippet contains the code for A* that was used during the development of the model we represent in this paper. The code is written in Python.

```
def a_star_search(graph, goal):
    frontier = PriorityQueue()
    start = (0, 0)
    frontier.put(start, 0)
   came_from = {},  cost_so_far = {}
   came_from[start] = None
   cost_so_far[start] = 0
   goal = deque(goal)
   while not len(goal) == 0:
     current_goal = goal.popleft()
     while not frontier.empty():
       current = frontier.get()
       for next in graph.neighbors(current):
       new_cost=cost_so_far[current]+graph.cost(current,next)
       if next not in cost_so_far or new_cost<cost_so_far[next]:
            cost_so_far[next] = new_cost
            priority = new_cost + heuristic(current_goal, next)
           frontier.put(next, priority)
           came_from[next] = current
   return came_from, cost_so_far
```

IV. Problem Description

This paper aims to provide a comparison in the performance of Dijkstra and A* algorithms. The idea to this problem came during a challenge to create a bot that would deliver pizzas depending on the input, which would be a list of tuples with coordinates representing points of a 2D grid. The initial solution was provided with Dijkstra's algorithm only, but aiming to achieve a better performance, mostly in time, A* came as an option. The results from the simulations are quite descriptive and understanding, and will be elaborated in the next subsections.

A. Simulation

Since the challenge was to deliver a proof of concept, all the simulations have been executed using a PC, and the results are only for research purposes, thus they can't be used with production purposes, without further, deeper and real tests.

All the simulations have been executed multiple times, with different input parameters, in different environments with different sizes. In the following subsection, the achieved results will be represented in different figures.

B. Simulation Results

In the previous sections, we explained the basic principles of both algorithms, Dijkstra and A*, on how way that they operate. To provide an even better comparison between them, we will try to represent the results of the simulations for both of them.

To do so, we will start will smaller group of tuples, in smaller grids, and will keep incrementing both of them.

Figure 3 contains the results of the simulations, which also contains the code to visually illustrate the path which the algorithm follows to reach all of the given points. Figure 4, instead, represents the results of the simulations without drawing the path. The results are much faster, which is understandable due to the less calculations the machine has to process.

Regarding the results, the horizontal axis represents the grid size, the first bar represents the results when the tuples were provided to Dijkstra algorithm, and the value represents the time the algorithm needed to find the shortest path, and the second bar represents the results about the time A* algorithm needed to find the shortest path for the provided tuples. During the comparison, we can see that big impact on time execution have other features of the whole project, in this case, path drawing. It is obvious that the results from figure 4 are about 80 - 90% better compared to figure 3.

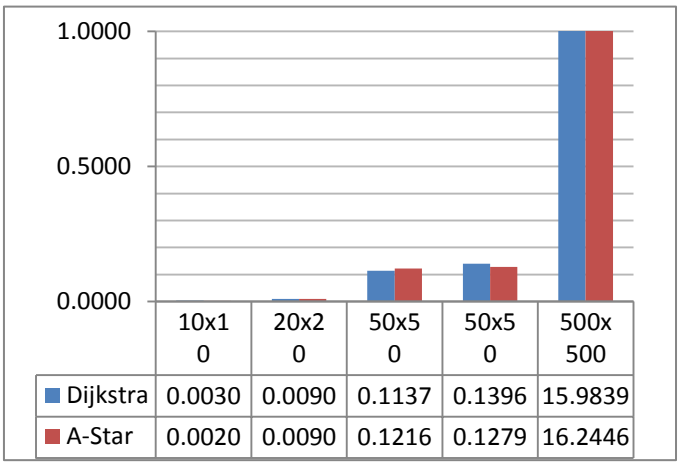
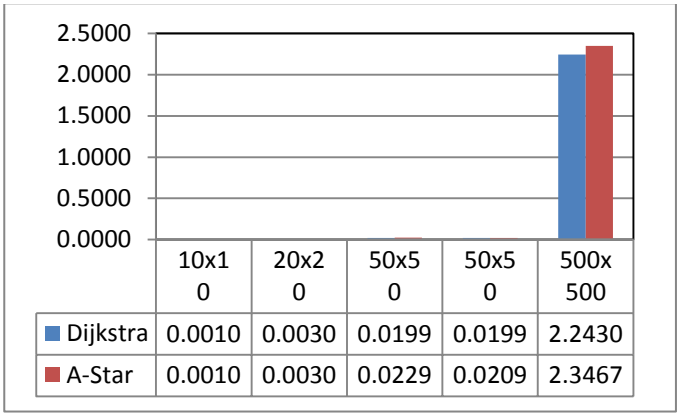


Figure 3. Simulation Results when the algorithm draws the path.



	10x10	20x20	50x50	50x50	500x500
Dijkstra	0.0010	0.0030	0.0199	0.0199	2.2430
A-Star	0.0010	0.0030	0.0229	0.0209	2.3467

Figure 4. Simulation Results when the algorithm does not have to draw the

V. CONCLUSION

In this work, a performance perspective of Dijkstra and A* algorithms is evaluated. The evaluation is based on the size of the grid and the input provided. We can conclude that both algorithm's performance rate is linear, which means that the bigger the grid, and the input, the more time will be needed to find the shortest path. What we failed to conclude is that A* performs better compared to Dijkstra. This might be for several reasons, one of which must be the environment where the simulations were executed.

As it is obvious from Table 3 and Table 4, in different conditions, the algorithms behavior is different. Thus, we don't have a clear "winner" between them, although in most cases, A* seems to have the advantages and be the best algorithm to use.

Since the algorithms were developed only for proof of concept purposes, they are not maximally optimized, and we think that with some more improvements, the algorithms can be much more efficient, which will also be in our focus for future work. It can also be improved to run on much bigger environments, for example, in a 50000x50000 grid.

REFERENCES

[1] B. Marr, "How much data do we create every day? The mind-blowing stats everyone should read." [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#fbc976460ba9.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs". Numerische Mathematik. 1959.

[3] M. Leyzorek, R. S. Gary, A. A. Johnson, W. C. Ladew, S. R. Meaker Jr, R. M. Petry, R. N. Seitz, "Investigation of Model Techniques – First Annual Report – 6 June 1956 – 1 July 1957 – A Study of Model Techniques for Communications Systems." Cleveland, Ohio, 1957, Case Institute of Technology.

[4] I. Zarembo, S. Kodors, "Pathfinding Algorithm Efficiency Analysis in 2D Grid", Environment, Technology, Resources. Proceedings of the 9th International Scientific and Practical Conference, Volume II. 2013

[5] H. Wang, J. Zhou, G. Zheng, Y. Liang, "HAS: Hierarchical A-Star algorithm for big map navigation in special areas", 2014 International Conference on Digital Home, 2014.

[6] S. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach (2nd Edition)", Prentice Hall, pp. 97 – 104, 2003.

[7] X. Ji, L. Liu, P. Zhao, D. Wang, "A-Star Algorithm Based On-Demand Routing Protocol for Hierarchical LEO/MEO Satellite Networks", 2015 IEEE International Conference on Big Data (Big Data), 2015

How to Cite this Article:

Ramadani, E., Halili, F. & Idrizi, F. (2019) Tailored Dijkstra and Astar Algorithms for Shortest Path Softbot Roadmap in 2D Grid in a Sequence of Tuples. International Journal of Science and Engineering Investigations (IJSEI), 8(92), 56-59. http://www.ijsei.com/papers/ijsei-89219-06.pdf