The Shortest Path Problem In Artificial Intelligence and Its Solutions

Yana Garipova

April 2018

Abstract

This project is about the shortest path problem in artificial intelligence and its solutions. The project's ultimate goal is to achieve a firm understanding of the subject by researching academic literature and running experiments. The project consists of the written research and the experiment sections. In the research part, the text provides an overview of different solutions to the shortest path problem. Since there are many solutions existing for various special cases, the text mostly concentrates on the mainstream ones: Dijkstra, A*, and D* branches of algorithms. The experiment part of the project deals with Dijkstra's and Priority Queue Dijkstra's algorithms. The experiment checks a claim that implementation of priority queue does not make much difference in terms of usage and runtime when dealing with real-world navigation. The algorithms' pseudocode and test results are summarized in this paper.

Introduction

Shortest path problem is one of the most attractive questions in the field of Artificial Intelligence. This subject is very practical since people use it every day via smartphone navigation. Once upon a time the novice of digital navigation has drastically changed our world and its economy in terms of mass logistics and people's daily routine. Nowadays, people mostly rely on the digital map when traveling or driving (sometimes even more than own heads). Then, we want to know the shortest path from a place to a place, and not only by distance but also in terms of time. We want to avoid traffic jams and reroute as quick

as possible after a missed turn. An algorithm is very important here: it must be accurate, it must be fast and light enough to be run over and over by a smartphone, and it must be suitable for a changing environment. There is a lot of different algorithms for finding the shortest path and a lot of literature sources on them.
Now, let us read some of them together and try to understand the algorithms and their classification.

Solutions to the Shortest Path Problem

There are numerous solutions for Shortest Path Finding problem. Most of them are improved versions of each other, but several individual branches of algorithms can be identified, like D* and A*. In this part of the paper, let us use the book [1] as a guide to go over the common algorithms and their different versions.

Dijkstra Based Algorithms

Dijkstra is the first algorithm, written by Edsger Wybe Dijkstra. It is over 50 years old but it is the one that runs the Google Maps [3]. Dijkstra is a uniform cost search that checks every node and then chooses the shortest path to the goal node. It works for both directed and undirected graphs[2]. Its time complexity is O(N^2), where n is the number of nodes. Dijkstra algorithm uses the concept of path weights (non-negative) that allows it to find the shortest path not only by distance, but also in terms of time.
Of course, some changes have been introduced to the algorithm since 1959. In 1984 computer scientists Fredman M.L. and Tarjan R.E. implemented the concept of priority queue into the original algorithm[4]. Using priority queue significantly improves the time complexity of Dijkstra's algorithm to O($E + N * log(N)$), where E is the number of edges in a graph [4]. This version is especially great for graphs with a small ratio of edges per node [1]. According to the same book [1], this implementation does not work better for some types of directed graphs, such as acyclic graphs or when path weights are small integers. Plus, the book[1] claims that priority queue does not make much difference if we deal with real-world large road networks and there are better approaches to improve Dijkstra's time complexity.
Firstly, the book introduces Bidirectional search, which is two Dijkstra algo-

rithms running from the start node and a goal node and meeting in the middle. This shortens the original algorithm's time complexity by 2.
Even further, the book [1] introduces the hierarchical query algorithm, which is bidirectional Dijkstra with two priority queues and hierarchical approach implemented. Hierarchical approach (graph hierarchy) divides all the graph nodes into levels by analyzing the connection between the nodes' neighborhoods. (It uses the knowledge of neighborhoods and the fact that the straightest connection leads to the shortest way to find the shortest path faster.) This division into levels helps when there is a tie between nodes or when optimizing algorithms further in terms of rearranging nodes, precomputing distances, and changing abort criteria [1]. Algorithms like Contraction Hierarchies, Transit Node Routing, and Hub-based Labeling Algorithm are Dijkstra based and optimized in accordance with the graph hierarchy principles[1].
There are also non-hierarchical approaches for optimizing Dijkstra. One of the most interesting examples is Edge Flags algorithm [1]. It implements the realistic idea of road signs indicating directions and runs Dijkstra backward search. Of course, this is not all that there is to it. Another modification of Dijkstra is Reach algorithm, which implements node pruning. The A* is also a generalization of Dijkstra's algorithm that prunes the subgraph if additional information is available that updates a heuristic value[1].
Dijkstra is the standard algorithm which gave birth to or inspired many other efficient solutions for shortest path problem, including the A*. However, the A* algorithm turned out to be a lot different, so it is being recognized as independent. Thus, let us explore the A* family of algorithms.

A* algorithms

The A* is an informed search presented by Peter Hart, Nils Nilsson and Bertram Raphael in 1968. A* utilizes chooses a path by a cost function. The cost function is distance from current node to the next node (n) + the heuristic value of n. The heuristic values are estimation of the distance from each node to the goal. The A* requires that the heuristic values are admissible (not overestimated) and consistent (heuristic value of a node n $\leq$ the heuristic value of n's neighbor + the distance to the neighbor) to return the optimal solution. The classic A* simply follows the path of the smaller values of the cost function.[5]
Dijkstra's algorithm uses distances from the source to reached nodes as the only criteria to decide which node will be selected next [1]. Using heuristics allows

A* to find the shortest path faster. However, if A*'s heuristic function is not consistent, it does not guarantee to return the actual shortest path. Moreover, A* uses a lot of memory [5]. For a larger map it is recommended to run an Iterative Deepening A* (IDA)[5].
The A* could also be bidirectional, but it raises the question of terminating conditions[1].
The book [1] also mentions Landmark A* (ALT) as a non-hierarchical solution to the shortest path routing problem. First, the algorithm computes the set of landmarks - nodes with known distances to other nodes (pre-processing stage) and then uses the triangle inequality rules to find the direction of the shortest path [1].
There is also a Repeated A* that runs the classic A* search and follows the path to the goal. If the cost function values have changed on the way, it re-runs the A*[6].
Generalized Adaptive A* (GAA*) is upgraded Repeated A*. GAA* updates the heuristic values of the explored set nodes after every search is performed and uses them in the new search. It also updates the h values of some states if the cost has decreased. GAA* stays consistent and therefore, admissible, what guarantees optimality of solution[6]. However, it does not use the previously found optimal paths [6]. That is why MPGAA* has been introduced.
Multipath Generalized Adaptive A* (MPGAA*), written by Carlos Hernandez, Roberto Asin, and Baier Jorge in 2015[6]. MPGAA* utilizes the previously found paths for fast navigation in dynamic terrain [6]. It also stops the search early if a certain linear algorithm returns true. A couple of years later the same authors proposed to improve the MPGAA* for extended visibility ranges, but it is not very useful for the real world navigation [7].
Lifelong Planning A* (LPA) is an incremental version of A*, written by Sven Koenig, Maxim Likhachev, and David Furcy in 2004[8]. LPA repeatedly finds shortest paths from a given start vertex to a given goal vertex while the edge costs of a graph change or vertices are added or deleted. Plus, it reuses the results from previous A* searches[8]. The LPA was a base for a new algorithm called D*Lite, also written by Koenig Sven and Maxim Likhachev[11]. The authors of the article [6] called D* Lite algorithm the "state of the art" of fast navigation in the dynamic terrain. Let us research about the D*Lite algorithm and where it is coming from.

D* algorithms

Turns out, D* is not a simpler name for Dijkstra. Actually, D* is an informed incremental search algorithm, which is based on A* but is dynamic (i.e. path cost parameters can change during the problem solving process)[9]. D* was written by Anthony Stentz in 1993. D* is able to plan paths in unknown, partially known, and changing environments[9]. There is also Focussed D*, also written by Anthony Stentz. This algorithm extends D* even more by calculating heuristics for some nodes, just like A* does[10]. The D*Lite algorithm combines these two and the Lifelong Planning A*[11]. D* Lite is recognized as the most successful version of D* family[6], mostly because of its easy implementation. Even Stentz Laboratory uses the D*Lite. Interestingly enough, from [9], [10], and [11] we can see that D* algorithms are usually used for automotive robots, such as drones.

Which algorithms to choose for the experiment?

Dijkstra, A*, and D* branches of algorithms are the most often used ones, but of course they are not everything there is to the shortest path problem. There are other good solutions, for example, Reach algorithm. It is based on both Dijkstra and A*[12] and could be used for directed and undirected graphs[13]. Reach algorithm allows paths such that edges in opposite direction have a negative length. There are plenty of other solutions for the shortest path problem. The reader is encouraged to take a look at the source[1]. All the algorithms are extremely interesting to study, but due to time constraints for this project, only a couple of them shall be chosen for the experiment part.
The facts that Dijkstra algorithm is the grandfather of the other shortest path searches and it is being used by Google maps create a high interest to test it and compare to another algorithm.
Recall that standard Dijkstra's complexity is O(N^2), where n is the number of nodes, when Dijkstra with implementation of priority queue runs O($E + N * log(N)$), where E is the number of edges in a graph [4]. Also, as mentioned before, the book [1] claims, that there is no much difference between these two when dealing with real world road networks. I would like to check this claim in this project's experiment.

Dijkstra and Priority Queue Dijkstra

Dijkstra is an old algorithm with a high credibility. It does not have precomputing stages like hierarchical algorithms and its result is always guaranteed to be correct, because it does not depend on heuristic values like A* does. Dijkstra's algorithm's generality and simplicity of implementation and understanding make it a popular and reliable solution.
First, let us take a look at the standard Dijkstra's pseudocode[2]:

Set of visited vertices is empty
Distance to start vertex = 0
Vertices queue/list has all the vertices
For all other vertices:
    distance = 1000000 //some large number equivalent to infinity;

While the Vertices queue is not empty:
    select vertex with the min distance
    add it to the visited set
    if new shortest path is found:
     set the new value of shortest path
Return distance

The pseudocode of priority queue dijkstra is as follows[4]:

Create an empty distance to nodes and the path sets
Create an empty set of closed nodes
Initialize visited set as a priority queue
Initialize the counter
List all vertices
Set distance to start vertex = 0
Path to Start Vertex = None
For every (not closed) and (not starting) vertex:
    distance = 1000000
    path = None
Other nodes:
    add to priority queue
While visited priority queue is not empty:

get information from the next node on the priority queue
select min node
add explored edge of min node
increment count
If the min vertex is None or goal:
break
For every edge of min vertex that is not None nor closed:
update the weight of the path (current weight + edge length)
if it is a better path (new weight¡ best found distance):
——remove the previous vertex from priority queue
——update the best found distance
Return path, distance, count

Experiment: PQ Dijkstra vs Non PQ Dijkstra

Standard Dijkstra and Dijkstra with priority queue implemented are to be tested in this part of the project.

Then, they will be compared in terms of actual run-time for different problem sizes (time complexity).

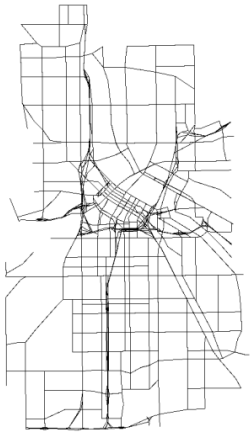


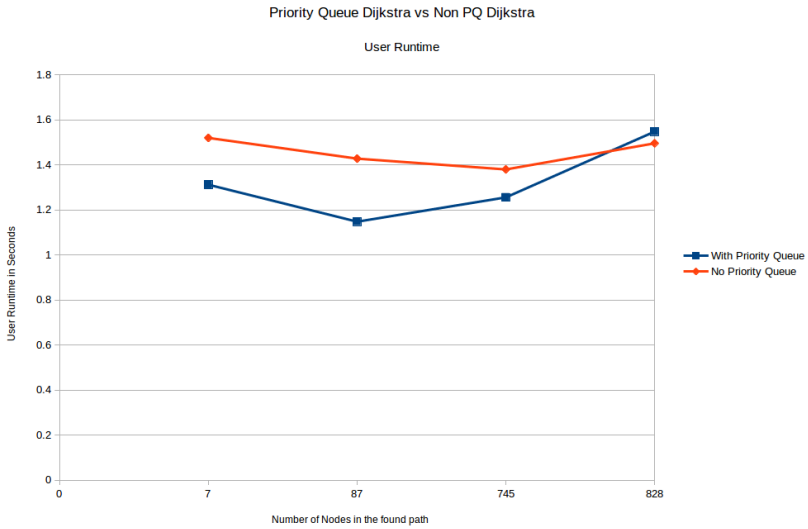
Figure 1: Minneapolis Map

The algorithms are to be tested using a map of Minneapolis in lisp format. The map is formatted as follows: each line contains information about a segment of a road. Each line includes 5 numbers in parentheses. The first number is 1 to indicate a one way road or 2 to indicate a two-way road. The next two integer numbers are the x and y coordinates of the start of the segment, the next two numbers are the x and y coordinates of the end of the segment, For instance, (1 1121 7568 1042 7545) indicates a one way road starting at point [1121, 7568] and ending at [1042, 7545]. There are 1357 lines in the file, each corresponding to a road segment. As an image, the map looks like Figure 1.

This experiment's programming part includes the text file with the map data, the map file, queue/priority queue helping file, and the main file. The map file defines the Map, Edge, and Vertex classes and has code for algorithms and some other functions. The algorithms and all the other code is written and run in Python 3.
The two algorithms were run for different path sizes. The standard Dijkstra checks every node before it chooses the shortest path, thus it always gives the correct result. Implementation of the priority queue is supposed to make the algorithm find it faster since some paths will be prioritized over the other ones. The starting and goal vertices were chosen at random, creating the ultimate shortest paths of different sizes. The results are summarized in the following next.

Results

Running the same algorithms for the same path produces different results in tenths of a second, but the general trend looks like this:



Most of the time the Priority Queue Dijkstra works faster, indeed. However, sometimes the standard Dijkstra returns slightly quicker. In theory, the algorithms have significantly different time complexities, but in reality, both algorithms return within 1.6 seconds. Thus, the book's "The Shortest-Path Problem: Comparison and Analysis of Methods" statement is confirmed.

Future work

Here, the runtime of Dijkstra and Dijkstra with priority queue have been tested. This project has several opportunities for future development. One of them is to take Edge Flags version of Dijkstra and compare it to the standard and PQ Dijkstra. This algorithm has a very interesting idea. It works similar to the real world driving with no map but relying on the road signs. Moreover, it would be exciting to check all of these algorithms in real time, in terms of ability to work efficiently in the dynamic environment and compare them to the "state of the art" of dynamic environments, D*Lite algorithm. Unfortunately, this idea is too ambitious for this project but it is a great way to develop it further.

Conclusion

The purpose of this project was to understand and represent well the shortest path problem in artificial intelligence and its solutions. The goal was achieved by studying related literature, programming, and running an experiment. The most popular application of the shortest path problem is the road network navigation. Numerous algorithms based on Dijkstra, A*, and D* allow us to find the shortest path in general and in various special cases. The standard Dijkstra and Dijkstra with priority queue implemented are the oldest and the most reliable algorithms. Many sources claim that Dijkstra with priority queue has a much better time complexity than the standard Dijkstra. Even if it is so in theory, one of the sources claims that there is no practical difference between the two, and this is exactly what this project's simple experiment has confirmed.

Bibliography

[1] Ortega-Arranz, Hector., Diego R. Llanos, and Arturo. Gonzalez-Escribano. *The Shortest-path Problem : Analysis and Comparison of Methods.* Synthesis Digital Library of Engineering and Computer Science. San Rafael, California (1537 Fourth Street, San Rafael, CA 94901 USA): Morgan Claypool, 2015.

[2] Yan, M. Dijkstra's Algorithm. *MIT Lectures*, 2018.

[3] Lanning, D., Harrell, G., Wang, J. Dijkstra's algorithm and Google maps. *Proceedings of the 2014 ACM Southeast Regional Conference*, 1-3. Kennesaw, Georgia, 2014.

[4] Fredman, Michael Lawrence; Tarjan, Robert E. (1984). Fibonacci heaps and their uses in improved network optimization algorithms. *25th Annual Symposium on Foundations of Computer Science*. IEEE. pp. 338–346.

[5] Russel, S.J, Norvig, P. *Artificial Intelligence A Modern Approach*. 3d ed. Pearson Education, Inc. 2010.

[6] Hernandez C., Asin, R., Baier J. A. Reusing Previously Found A* Paths for Fast Goal-Directed Navigation in Dynamic Terrain. *Association for the Advancement of Artificial Intelligence* (www.aaai.org), 2015.

[7] Hernandez, C., Baier, J. A. Improving MPGAA for extended visibility ranges. *En Proceedings of the 27th International Conference on Automated Planning and Scheduling*, ICAPS 2017 (pp. 149-153). AAAI press, 2017.

[8] Koenig, S., Likhachev, M., Furcy, D., Lifelong Planning A*. *Artificial Intelligence Journal*, 2004

[9] Stentz, Anthony, Optimal and Efficient Path Planning for Partially-Known Environments. *Proceedings of the International Conference on Robotics and Automation*: 3310–3317, 1993.

[10]Stentz, A., The Focussed D* Algorithm for Real-Time Replanning. *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[11]Koenig, S., Likhachev, M., D* Lite. *American Association for Artificial Intelligence*, 2002.

[12] Goldberg, V.A., Kaplan, H., Werneck, R., Reach for A*: Point-to-Point Shortest Path Algorithm. *MIT Lectures*, Silicon Valley, 2009.