Shortest Path Comparison between the A\* and Dijkstra’s Algorithm

By

Steven Gantz

I. Introduction to History of A\* and Dijkstra’s Algorithms

In June of 1959, Edgar Dijkstra approached the problem of finding an algorithm that could find the shortest path through a graph in polynomial time. He initially approached this problem to demonstrate the capabilities of a new computer called ARMAC. Dijkstra finally formulated the algorithm one day while attempted to generate a transportation map of 64 cities in the Netherlands. [2]

He noted his findings in a paper titled “A Note on Two Problems in Connexion with Graphs”. The paper suggested that the aforementioned algorithm was valid only when at least one path exists between any two nodes in a graph. The paper describes exactly two problems the algorithm is able to solve.

The first is the following:

*Construct the tree of minimum total length between the n nodes. (A tree is a graph with one and only one path between every two nodes) [1]*

The second is the following:

*Find the path of minimum total length between two given nodes P and Q[1]*

The algorithm Dijkstra developed first solved the first problem. Once a tree of minimum length between nodes existed, Dijkstra found it was trivial to solve the second problem. This algorithm was more effective than the previous algorithm which Dijkstra noted was, *“The solution given above is to be preferred to the solution by L. R. Ford as described by C. Berge for, irrespective of the number of branches, we need not store the data for all branches simultaneously but only those for the branches in sets I and II, and this number is always less than n. Furthermore, the amount of work to be done seems to be considerably less.”*[1]

This was the algorithm used primarily by computer scientists for relevant applications until a requirement arose for a faster algorithm with a smaller space and time complexity. This manifested itself in a path planning algorithm.

In 1968, Nils Nilsson was attempting to improve the path planning done by Shakey the Robot. An early prototype robot that could navigate through an obstacle filled room. The algorithm he implemented, was titled A1. Soon after, Bertram Raphael suggested significant improvement upon the algorithm, with the pair retitling it to A2. Finally, Peter Hart made some final changes to A2 before testing and concluding that it was the best possible algorithm for finding shortest paths.

Later that year, in June of 1968, the trio published a paper titled “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. The paper attempts to bring together two different ways of approaching the shortest path problem. They note the methods as; the mathematical approach, and the heuristic approach [3]. They proceed to define the algorithm with examples and proofs. The algorithm has since been classically called the A\* algorithm. It enjoys widespread use throughout computer systems and continues to be used in many applications around the world.

II. Overview of Computations

The A\* algorithm is remarkably similar to Dijkstra’s original algorithm. The main difference in the process is the inclusion of a heuristic to each path calculation. Where Dijkstra’s Algorithm keeps track only of the current path traveled thus far, comparing each node to find the shortest path forward, A\* deviates by also adding in a heuristic analysis to this comparison. Therefore there is a more specific weight given to each path that is more likely to represent to paths distance so far as the heuristic should favor nodes that are literally closer to the final goal node. It can be said that A\* places preference on nodes literally closer to the goal node. Effectively reducing Dijkstra’s Algorithms extra expansions on nodes that may further away from the goal node than other more favorable nodes. This is shown visually in Figure 1 below:

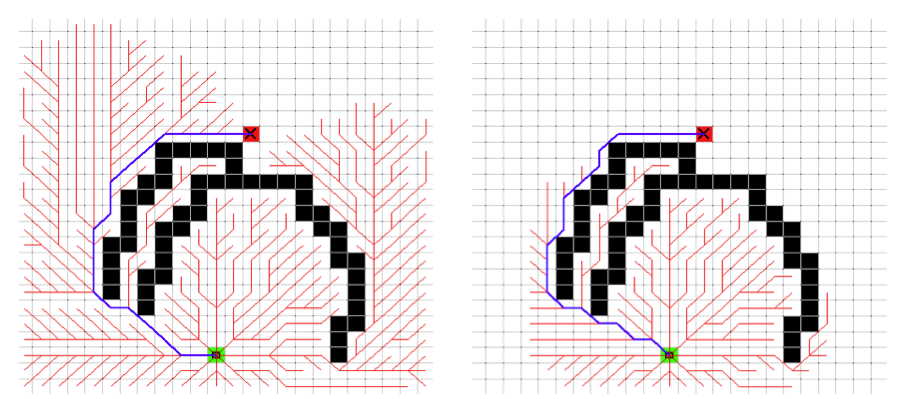
Fig. 1.1

Fig 1.2

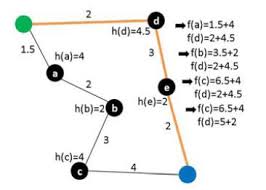
[6]

Dijkstra’s Algorithm is located on left, denoted as Fig 1.1, which travels in equal distance in every direction until it eventually finds the source node. Comparatively the A\* algorithm (Fig 1.2) attempts to travel only closer to the goal node using the heuristic to estimate the remaining distance.

This heuristic estimates the distance to the goal node from each checked node, giving a different final weight than would be found by Dijkstra’s Algorithm.

Fig. 2

[5]



As seen in Figure 2 above, where Dijkstra’s Algorithm would only count the length of the current path to build its tree, the heuristic adds an extra level of detail that sets the algorithm on a more direct course. Under certain well-defined conditions, the A\* algorithm will always be the optimal method for traversing through the shortest path in the graph.

The heuristic being the primary defining factor between Dijkstra’s Algorithm and it’s A\* counterpart, it becomes obvious that though Dijkstra’s was developed nearly ten years before the A\* algorithm was first implemented, it is now commonly accepted that most general variants of Dijkstra’s Algorithm are in fact a special case of A\* where the heuristic has a zero value for all nodes.

III. Space and Time Complexity Comparison

The running time of Dijkstra’s Algorithm using big O can vary based on algorithm implementation. The general upper bound of big-O for the algorithm is

*O(|V|2).*

This is the upper bound, and there exists multiple implementations of the algorithm that work faster than the upper bound. However, none of these implementations are any faster than an implementation on a sparse graph G. With a sparse graph, using an efficient data structure, one may have a worst case runtime of

*O(|E|log|V|)*

The time complexity of Dijkstra’s Algorithm is always a worst case of

*O(|V|2).*

In some very special cases, or using finely tuned data structures, one is able to decrease the worst case to

*O(|E|log|V|)*

However, generally the cost is *O(|V|)* for each insertion operation, and *O(|V|)* for every time finding the minimum. This in addition to the initial |V| operations to initialize the nodes, results in a runtime of

*|V| + |V|2* = *O(|V|2)*

The time complexity of the A\* algorithm is exponential in regards to the complexity of the heuristic. Generally, while using an optimum heuristic, A\* will be *O(V)* in space complexity as only the required nodes will be stored in the current path. However, if the heuristic itself is polynomial or exponential, then the space complexity will represent that in space complexity.

IV. A\* and Admissibility

The A\* algorithm depends heavily on the concept of admissibility. The algorithm only works when the heuristic estimate is considered admissible. Whether the heuristic is admissible is dependent on whether the heuristic ever overestimates the actual cost. If an overestimation occurs, the current length of the path can be overestimated, and therefore not lead to an optimal path. The equation below displays this feature:

*n is a node*

*h is a heuristic*

*h(n) is the cost by h to reach the goal node from the node n*

*h\*(n) is the actual cost to reach goal from node n*

*f(n) is the resultant amount and the current length of the path*

*g(n) is the total cost from the starting node to the current node n*

*h(n) is the estimated cost heuristic from the current node to the goal*

*f(n) = g(n) + h(n)*

*if h(n) > h\*(n), then f(n) may be greater than the optimum path, and a legitimate path may not be found.*

Dijkstra’s Algorithm has no such issues, and is therefore is easier to use when a heuristic is inadmissible or difficult/impossible to calculate.

V. Web Application Project

To demonstrate the visual differences, positives, and negatives along with using Dijkstra’s vs the A\* Algorithms, I decided to implement a web application that treats a map of Kutztown University’s north campus as a graph mapped to a 2d plane.

The project is available at

<http://stevengantz.com/GraphTheoryProject2016/>

The image for the application was retrieved from Google images and is displayed in proportion to the user’s screen height and width. On the image there are nodes displayed that are selectable by the user. When two nodes are selected, one has the option of selecting either the A\* algorithm or Dijkstra’s Algorithm. The path is then computed through the graphs and displayed to the user with the optimal path.

The project was implemented in the JavaScript programming language using an in-progress library called p5.js. The library was designed to allow for easy drawing and image manipulation in a client browser environment. Due to limitations in the JavaScript programming language, I was unable to time the algorithm’s runtime within the browser. The actual implementations of algorithms also run at several levels of magnitude longer than the standard worst case for each. This is because JavaScript holds references to objects loosely, and therefore I routinely have to search through lists of vertices to find a specific value or update a specific field.

Due to difficulty in mapping out points on the Kutztown campus, as well as the lack of a 1:1 realistic campus map, the current application is loosely accurate to actual distances.

VI. Conclusion

Dijkstra’s Algorithm is still commonly used today in many applications where the graph being investigated is extremely large or unable to be accurately estimated. The algorithm is popularly implemented in mapping and graph traversal software. A\* on the other hand, is commonly used where an accurate heuristic can be generated within the mapping. It is an extremely common algorithm in AI and video games for this purpose. Many modified versions of A\* will tend toward the optimal solution, though not distinctively search for it, and it is this version that is most commonly used as it allows for nearly optimal solutions with a small memory and time footprint.

In regards to developing an application for traversing a graph of Kutztown’s campus, due to the campus’ small size and high number of nodes and edges, I think the A\* algorithm would be an effective tool in finding the shortest path between two points. While Dijkstra’s Algorithm would calculate the shortest path to each end node, A\* would run in a much faster timeframe. The heuristic is easily admissible as the distance calculated from node to node would be in direct lines using the distance formula, which should never over-estimate the shortest path.

**References**

[1] Dijktra, E. W.: A Note on Two Problems in Connexion with Graphs. Pg 1 (1959)

[2] Frana, Phil: “An Interview with Edsger W. Dijkstra” (<http://dl.acm.org/citation.cfm?doid=1787234.1787249>)

[3] Hart Peter E., Nilsson Nils, J, Raphael Bertram: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. Pg 1 (1968)

[4] <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>

[5] [http://digilib.its.ac.id/public/ITS-Undergraduate-19142-Paper-519446.pdf Pg 2](http://digilib.its.ac.id/public/ITS-Undergraduate-19142-Paper-519446.pdf%20Pg%202)

[6] <http://gamedev.stackexchange.com/questions/61850/in-a-star-how-does-the-heuristic-help-determine-your-path> - User: <http://gamedev.stackexchange.com/users/998/bobobobo>

<http://matwbn.icm.edu.pl/ksiazki/cc/cc35/cc3536.pdf>