Extended Essay

Computer Science

To what extent has the A* Pathfinding Algorithm become more optimal and efficient than Dijkstra's Algorithm and the Best-First-Search Algorithm through evolving from them?

Student name: Daniel Christian-Lau

Candidate number: xxx

Date of IB Exams: May 2016

Category: Computer Science (HL)

Supervisor's Name: xxx

Word Count: 3875

Abstract:

The research question being investigated: **To what extent has the A* Pathfinding Algorithm become more optimal and efficient than Dijkstra's Algorithm and the Best-First-Search Algorithm through evolving from them?** Focuses on exploring how the A* algorithm has adopted features from previous pathfinding algorithms, Dijkstra's algorithm and the BFS algorithm, to become highly efficient and effective for a range of different pathfinding problems. To answer this question a range of sources written by experts were considered to understand what differentiates the A* algorithm from other pathfinding algorithms and the characteristics which make it the most optimal and efficient algorithm choice in many cases.

The conclusion reached is that the A* pathfinding algorithm inherits elements from both Dijkstra's algorithm and the Best-First Search algorithm to become a stronger overall algorithm, which maintains the versatility required to always produce the most optimal/shortest path. A* includes two separate pathfinding functions in its algorithm (G (n) and H (n)) which were inherited from Dijkstra's algorithm and the BFS algorithm. These functions are merged in A*, enabling the algorithm to take into account the information currently available to it at any time and predictions made by the algorithm for the graph it is traversing when calculating its optimal path. The efficiency of the algorithm allows it to be used frequently without needing extreme amounts of processing power to compute, and the versatility allows it to be consistently effective with many graphs of varied characteristics. These perks mean the algorithm is a very popular option for pathfinding problems.

Table of contents:

1. Introduction

1.1 Pathfinding algorithms

Pathfinding or shortest path algorithms (the plotting, by a computer application, of the best route between two points) *(http://www.yourdictionary.com, n.d)*, such as the A* algorithm, are an important component of many systems today. They are used in video games to calculate routes around a virtual map *(En.cnki.com.cn, 2013)*, and in GPS (Global-positioning system)[1] *(Lee, n.d) (Heuristic shortest path algorithms for transportation applications: State of the art, 2005)* devices to guide the user to their destination. Like many advancements in modern technology or theory, their progression is reliant on adoption from previous versions. Learning from predecessors, like Dijkstra's algorithms and the Best-First-Search algorithm, allows them to learn from the past and become better at fulfilling their job. The quality of a pathfinding algorithm is based off of two things: how optimal the calculated path is and how much work was required to find that path (the efficiency of the algorithm) *(http://theory.stanford.edu, n.d)*.

1.2 The importance of investigating pathfinding algorithms

To what extent has the A* Pathfinding Algorithm become more optimal and efficient than Dijkstra's Algorithm and the Best-First-Search Algorithm through evolving from them?

To answer this question successfully an experimental comparison and analysis of the algorithms is required. It is also essential to understand that every country on the planet uses algorithms[2] *(TheFreeDictionary.com, 2011)* in some form, but not necessarily algorithms in computing.

---

[1] The Global Positioning System (GPS) is a satellite-based navigation system

[2] A finite set of unambiguous instructions that, given some set of initial conditions, can be perfor med in a prescribedsequence to achieve a certain goal and that has a recognizable set of end con ditions.

Algorithms can vary from generation of virtual landscapes to creation of complex chemical compounds. Even every day occurrences such as procedures for farming or teaching can be considered algorithms by definition. There remains paramount importance in learning how to maximise the effectiveness and efficiency of all of these algorithms, including pathfinding, as the world changes toward more computerised, automated systems which is ever more reliant on them. Learning how successful algorithms have been developed through studying their predecessors, and the features which can be adopted, can help to improve all future algorithms.

1.2 Global Implications

There is a plethora of uses for pathfinding algorithms, all of which would benefit greatly from improved pathfinding algorithms. Pathfinding algorithms can be used for much more than just game development and GPS. They can be used for finding the predicting how fungi will spread when growing, predicting animal migrations and for designing transport networks. All of which are only a handful of uses. Pathfinding algorithms are used frequently to design transport networks to reduce the travel time between points. This process is essential to reduce air and water pollution to the environment; ultimately algorithms such as the A* algorithm evolving will allow more efficient travel and result in a cleaner ecosystem[3] *(TheFreeDictionary.com, 2014)*. Furthermore, some pathfinding algorithms can mimic the movement of living organisms, allowing them to be useful in predicting movements or migration[4] *(Dictionary.com, n.d)*. Things which are useful for research purposes or for preventing endangerment and extinction[5] *(Lin, n.d)* of living creatures.

---

[3] An ecological community made up of plants, animals, and microorganisms together with their environment. A pond ora rain forest are each examples of complex ecosystems.

[4] To go from one country, region, or place to another.

[5] The extinction of an animal species occurs when the last individual member of that species dies.

2. The algorithms

In order to effectively investigate and compare algorithms, it is important to first understand their relative differences in methodology when trying to reach their goal. This will make it possible to see why an algorithm is 'better' than others in certain scenarios.

2.1 Dijkstra's algorithm

Dijkstra's Algorithm was designed to find the shortest path for a weighted graph[6] from one node (a point on the graph) to another. It is described as a Greedy Algorithm as it makes the most effective short-term decisions based on the information available at that time *(Muhammad, n.d)*. It can be used with either directed graphs (directional arrows as connections between nodes) or non-directed graphs, although the graph weights[7] must be non-negative and the graph must be connected[8] in some way. If either of these two conditions fail to be met, then the algorithm cannot be used *(Yan, n.d)*.

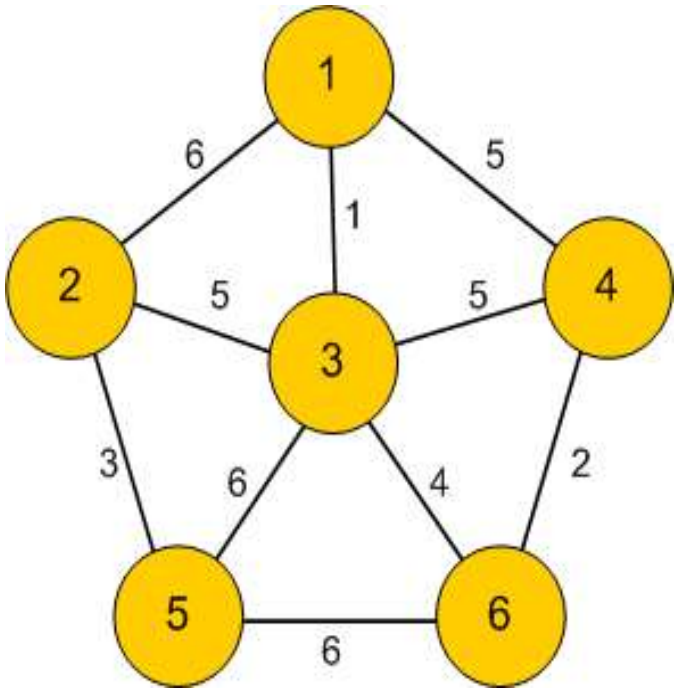


Figure 2.1.1 – A weighted graph *(Hadorn, n.d)*

[6] A type of labelled graph in which each branch is given a numerical weight and each label is a number *(Weisstein, n.d)*.
[7] The distance values between nodes *(Yan, n.d)*.
[8] Every node must be linked to at least one other node *(Yan, n.d)*.

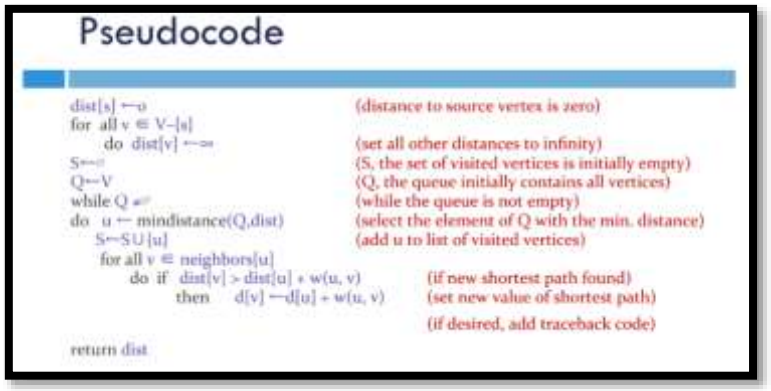


Figure 2.1.2 - Dijkstra's algorithm Pseudocode *(Yan, n.d)*

Figure 2.1.2 shows the Pseudocode for Dijkstra's Algorithm. The algorithm starts with an initial node and a goal node. The initial node is where the graph traversal begins, while the goal node is where it will end. The algorithm aims to find the least-cost path from the initial node to the goal node. A tentative distance value is assigned to every node (otherwise known as a Vertex in figure 2.1.2), which is 0 for the initial node and infinity for every other node. Next, the algorithm creates a set of unvisited and visited nodes to keep track of while running (*Fan, 2012).*

After these preliminary steps are complete, the algorithm starts at the initial node. The algorithm works around a 'current node', which changes after every loop of the algorithm. After the initial node, the current node will become the unvisited node with the smallest tentative value. All of the unvisited neighbours of the current node are considered (a neighbour node is any node connected to the current node). The distance to current node plus the weight of the edge between the current node and its unvisited neighbours is calculated. This value will replace the stored tentative distance to the current node if it is smaller than the current stored tentative distance to that node. After all the neighbours are considered, the current node is added to the visited set and removed from the unvisited set to keep track of where the algorithm 'has been'. The algorithm loops again after calculating a new current node and will only end when the goal node is added to the visited set and the shortest path is found *(Fan, 2012).* Dijkstra's Algorithm does not have to investigate all edges on a weighted graph to find the shortest path *(http://www.let.rug.nl, 2000).*

2.2 The Best-First Search algorithm

The Best-First-Search (BFS) algorithm uses heuristics, or Informed Searches, to exploit additional knowledge about the problem and calculate the most promising path to take *(http://www.cs.utexas.edu, n.d)*. Much like Dijkstra's Algorithm, the BFS Algorithm is a Greedy approach as it selects the path which appears locally best *(http://www.cs.ubc.ca, 2007)*. This prediction process is largely what defines the BFS Algorithm.

A heuristic function[9], h (n), provides an estimate of the cost of the path from a given node to the closest goal *(http://www.cs.utexas.edu, n.d)*; the algorithm will explore the node with the lowest cost first. There are many different types of heuristics available for use, making the BFS Algorithm adaptable for many different problems *(http://wiki.roblox.com, n.d)*.

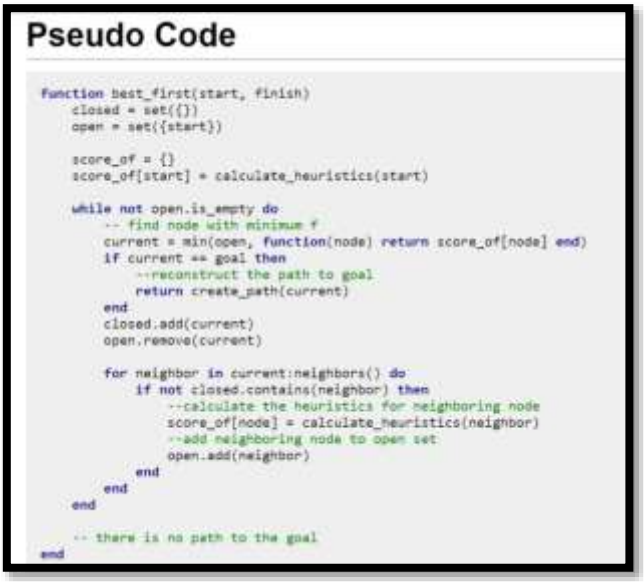


Figure 2.2.1 - BFS Algorithm Pseudocode *(http://wiki.roblox.com, n.d)*

[9] A relation from a set of inputs to a set of possible outputs where each input is related to exactly one output *(Nykamp, n.d)*.

The BFS Algorithm uses an initial goal and a goal node, similar to Dijkstra's Algorithm. It also has an open list and a closed list which can relate to the visited and unvisited list of Dijkstra's Algorithm. The primary difference between the two is that BFS uses heuristics to select optimal paths and hence does not need to backtrack to previous nodes. It is often used to find an optimal path on the first search, even if it is not the most optimal *(http://web.stanford.edu, n.d).*

As shown in Figure 2.2.1, the first step is the initial node being checked for goal conditions (conditions which define the goal node) *(http://web.stanford.edu, n.d).* If it is not the goal node, then it is removed from the open list and its child nodes[10] are put in the open list. The heuristic is applied to the child nodes, and the node that is estimated to be the most optimal is taken out of the open list and evaluated *(http://web.stanford.edu, n.d).* This node is checked for goal conditions and put in the closed list if none are found. If it is not the goal node, then its child nodes are put in the open list and the process continues. This algorithm will loop until the goal node is reached *(http://web.stanford.edu, n.d).*

The BFS Algorithm values efficiency over everything else, so it aims to complete it's Pathfinding in a single search, hence the name. It is efficient due to its linear nature in always taking the best available choice and has helped to created improved Pathfinding algorithms such as the A* Algorithm *(http://www.cs.ubc.ca, 2007).* A main limitation of the BFS Algorithm is that it does not try to find the most optimal path exclusively. The BFS Algorithm attempts to find an optimal path, but if it is not the most optimal, it does not matter. This may be a problem if the absolute quickest path is trying to be found to solve the problem at hand.

---

[10] Nodes connected to the current node (excluding the parent node) *(http://web.stanford.edu, n.d).*

2.3 The A* algorithm

The A* Algorithm is a more complex Best-First Search algorithm that relies on a similar open list and closed list to find a path that is both optimal and complete towards the goal. A* works by utilizing two separate path finding functions in its algorithm (G (n) and H (n)). These take into account the cost from the initial node to the current node: G (n) (taken from Dijkstra's algorithm) and estimates the path cost from the current node to the goal node: H (n) (taken from the Best-First-Search algorithm). The A* algorithm balances use of the two functions as it moves from the start node to the goal node. With each iteration through the main loop of the algorithm, it examines the current node n that has the lowest F (n) value when F (n) = G (n) + H (n) *(http://theory.stanford.edu, n.d).*

Dijkstra's Algorithm works well to find the shortest path, but it wastes time exploring in directions that aren't promising. Greedy Best First Search explores in promising directions but it may not find the shortest path. The A* algorithm uses both the actual distance from the start and the estimated distance to the goal to find the most optimal path in the least amount of work *(http://www.redblobgames.com, n.d).*

The clear main advantage of the A* algorithm is that it takes aspects from multiple different pathfinding algorithms in order be much more dynamic. The algorithm considering the best possible options at the time and for the future allows it to produce results to the same standard of Dijkstra's algorithm, while needing much less work, comparable to that of the Best-First-Search algorithm or better *(http://theory.stanford.edu, n.d).*

```
        // A*
1:      initialize the open list
2:      initialize the closed list
3:      put the starting node on the open list (you can leave its f at zero)
-
4:      while the open list is not empty
5:          find the node with the least f on the open list, call it "q"
6:          pop q off the open list
7:          generate q's 8 successors and set their parents to q
8:          for each successor
9:              if successor is the goal, stop the search
10:             successor.g = q.g + distance between successor and q
11:             successor.h = distance from goal to successor
12:             successor.f = successor.g + successor.h
-
13:             if a node with the same position as successor is in the OPEN list \
-                   which has a lower f than successor, skip this successor
14:             if a node with the same position as successor is in the CLOSED list \
-                   which has a lower f than successor, skip this successor
15:             otherwise, add the node to the open list
16:         end
17:         push q on the closed list
18:     end
```

Figure 2.3.1 – A* Algorithm Pseudocode *(Eranki, 2002)*

This process explained:

1. Create an open list and a closed list, both initially empty. Put the start node in the open list.

2. Loop until the goal is found or the open list becomes empty:

a. Find the node with the lowest F (n) cost in the open list and place it in the closed list. This node can be called q.

b. For the adjacent nodes to q:

i. Ignore them if they are already on the closed list.

ii. If they are not on the open list or the closed list, add them to the open list and store q as the parent for this adjacent node,

iii. Calculate the F (n), G (n) and H (n) costs for this adjacent node.

iv. If on the open list, compare the G (n) costs of this path to q and the old path to q. If the G cost of using the current node to get to q is the lower cost, change the parent node of the adjacent node to the current node (over q). Recalculate the F (n), G (n) and H (n) costs of q.

3. If open list is empty and the goal node has not been found, the algorithm fails *(http://web.stanford.edu, n.d)*.

3. An experimental comparison and analysis

3.1 The hypothesis - The A* algorithm will always calculate its path in a more efficient way than the other two algorithms and will also always produce the most optimal path, equal to that of Dijkstra's algorithm.

Why was this hypothesis made? This hypothesis is based off of the fact that the A* algorithm is much more dynamic than the other two algorithms, and therefore more suitable to navigate complex graphs. It takes into account factors which both Dijkstra's algorithm and the Best-First Search algorithm consider to calculate paths in much less work, while retaining the ability to find very optimal paths. Dijkstra's algorithm guarantees to find the most optimal path, which the A* algorithm should be able to simulate in many scenarios.

3.2 The method

The method will involve testing all three algorithms across different graphs and comparing how optimal the calculated paths are and how much work was required to calculate the paths. Each algorithm will be tested on a graph of 20x20 units[11] in size (400 units squared) a single time (3 per graph) and the work done (in squares) and length of the path (in units) will be recorded. A total of 6 unique graphs will be used in the experiment and each one will grow in complexity.

The simulation program being used for the experiment displays the work done as coloured squares (not green, red or grey), which also makes it possible to review where the work was

---

[11] A unit is the width or height of one square on the graph.

done by the algorithm. Aspects such as the areas where the algorithm has searched will become visible. Green, red and grey are excluded as they represent other points on the graph. Green squares are used to represent the start/root node, red squares are used to represent the goal node and grey squares are used to represent obstacles on the graph which the algorithms must work around.

Examples of the 6 graphs are shown below:

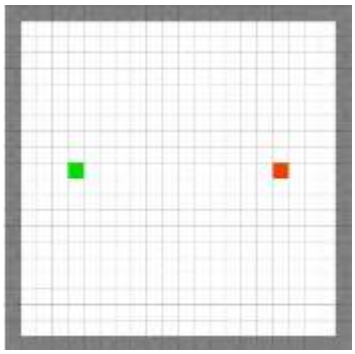


Figure 3.2.1 - The plain graph *(Xu, 2012)*

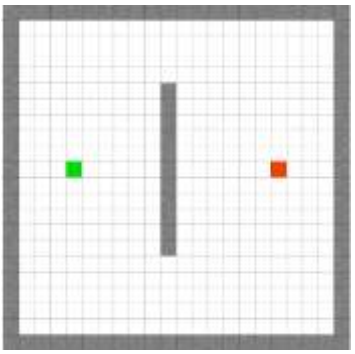


Figure 3.2.2 - The single wall graph *(Xu, 2012)*

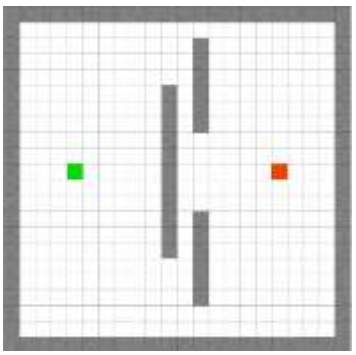


Figure 3.2.3 - The triple wall graph *(Xu, 2012)*

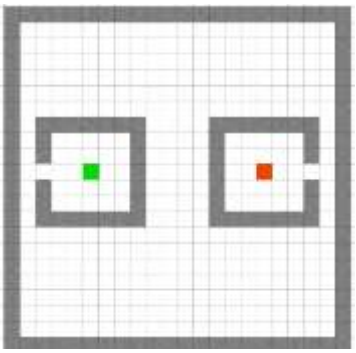


Figure 3.2.4 - Encapsulated goal and root nodes *(Xu, 2012)*

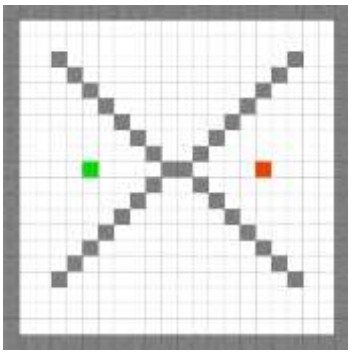


Figure 3.2.5 – The cross wall graph *(Xu, 2012)*

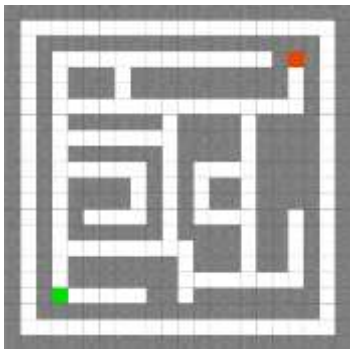


Figure 3.2.6 - The maze graph *(Xu, 2012)*

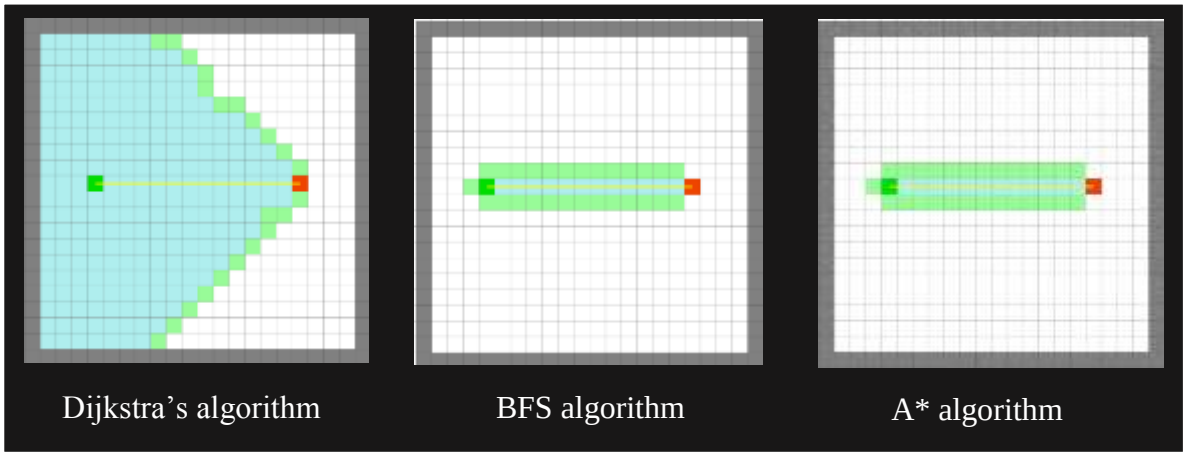
3.3 Controlled variables

In order for the experiment to remain fair, some variables need to be controlled – they must remain constant. The variables being controlled in this experiment are:

- Variable 1 – Use of the same simulation program: using different simulation programs may negatively distort results as different programs may use different versions of the same algorithm.
- Variable 2 – Use of the same graphs for each algorithm: using different graphs will lead to unfair comparisons being made. The algorithms must be tested the same for the comparisons to be fair.
- Variable 3 – Use of the same Heuristics: A* and the BFS algorithm are capable of using multiple different Heuristics which may vary results for large or more complex graphs. These two algorithms must use the same Heuristics for the entirety of the experiment for consistency and fairness.
- Variable 4 – The goal and root nodes retain the same position on each graph when the algorithm being used changes: changing the start and end points affects how the algorithm behave and the path which it decides to calculate. They must remain constant for a fair comparison.

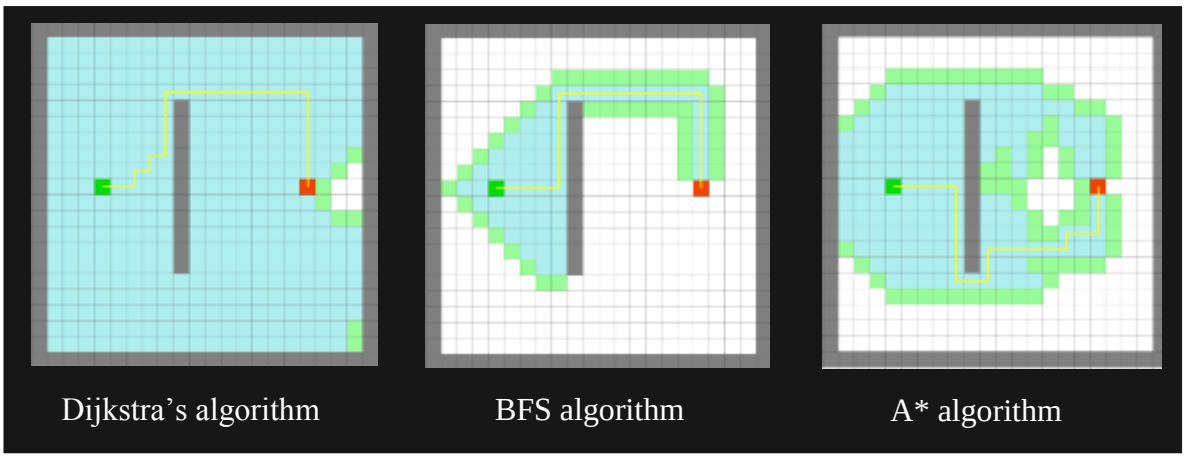
3.4 Analysis

Figure 3.4.1 - The plain graph test: *(Xu, 2012)*



For this graph, all of the algorithms found the most optimal path, although Dijkstra's algorithm was very inefficient in doing so. The BFS algorithm and the A* algorithm both had the same efficiency, which was far better than Dijkstra's algorithm as nearly all of the work done by the algorithms was toward the goal node.

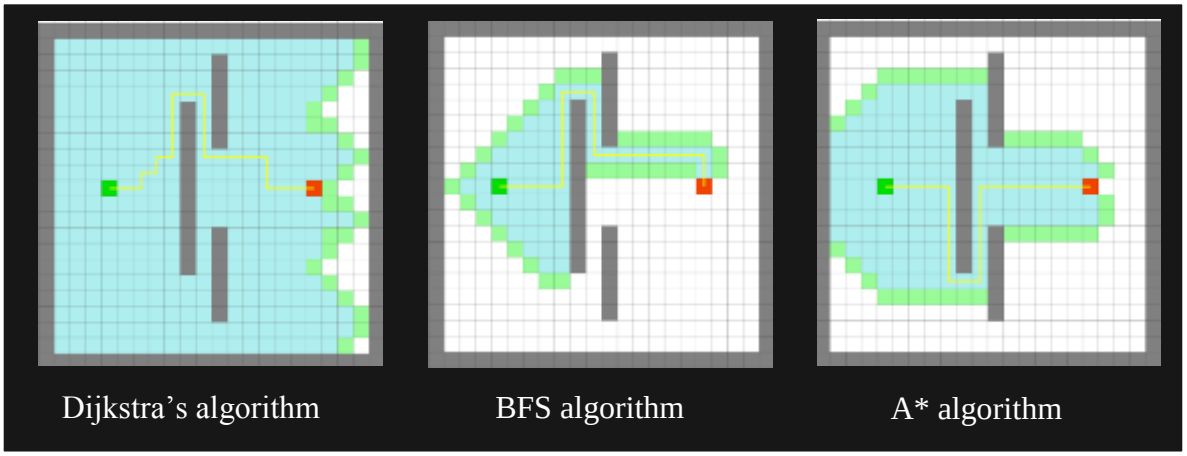
Figure 3.4.2 - The single wall graph test: *(Xu, 2012)*



The second graph required the algorithms to work around a very simple obstacle in the form of a wall. Like with the open graph, all algorithms found one of the most optimal paths (some graphs may have multiple solutions to the shortest-path problem) and Dijkstra's algorithm was far less

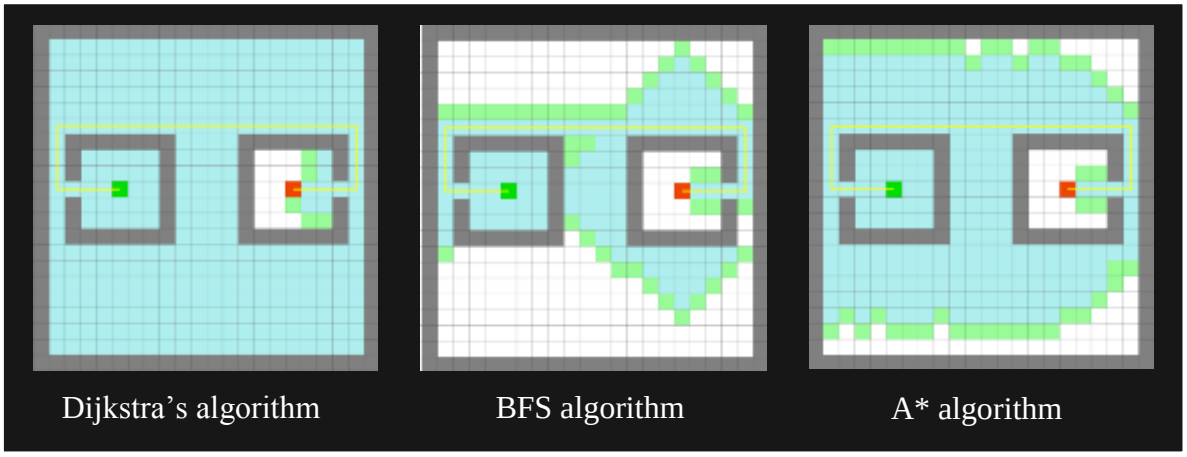
efficient than the others. A difference this time is that the BFS algorithm was more efficient than A*. BFSs work done did not deviate far from the calculated path, while A* had work done near the opposite side of the grid to its path.

Figure 3.4.3 - The triple wall graph test: *(Xu, 2012)*



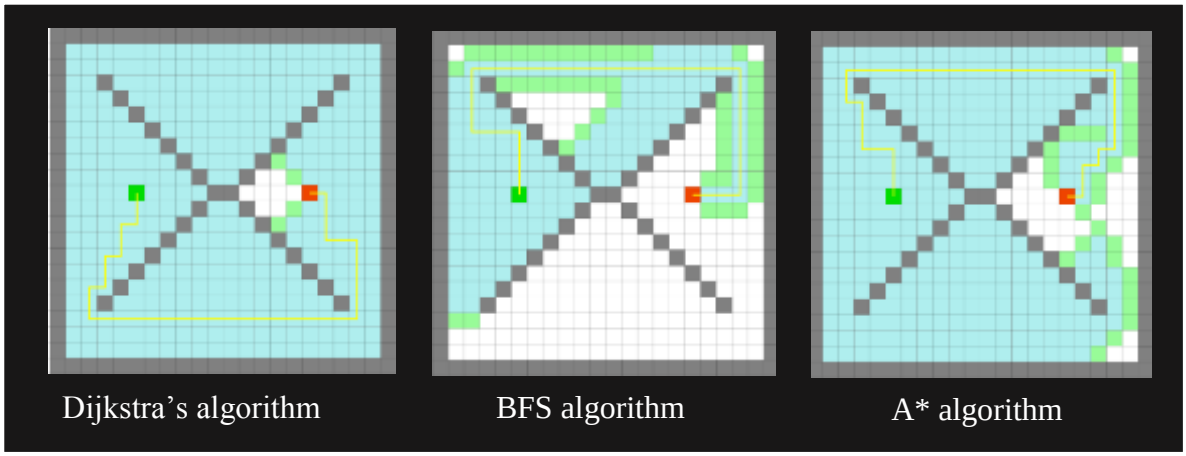
The third graph featured two smaller walls accompanying the original wall. It changed little from the second graph as the most optimal paths were the same size as before and found by all of the algorithms. As well as this, the efficiency of the algorithms changed very little.

Figure 3.4.4 - Encapsulated goal and root nodes graph test: *(Xu, 2012)*



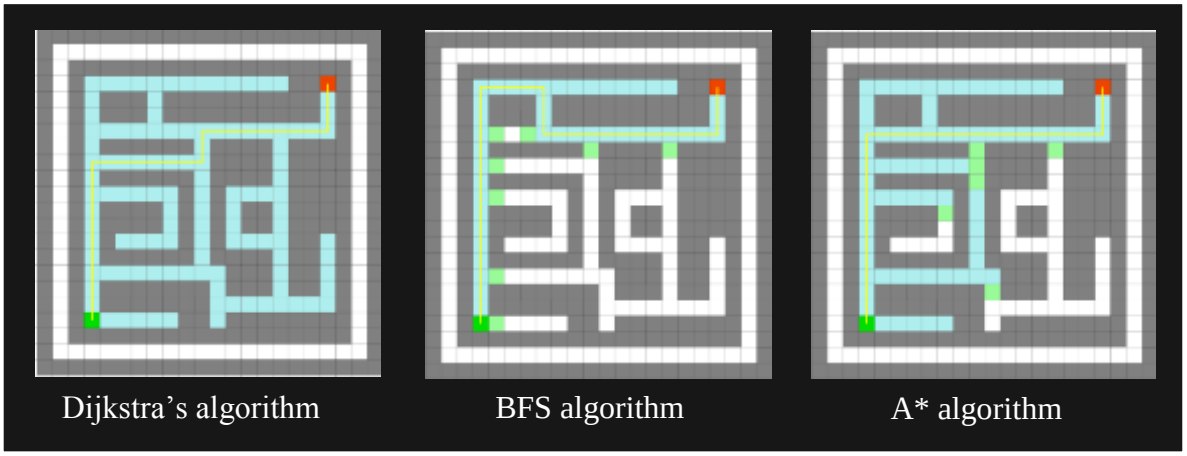
For the fourth graph, all of the algorithms were still able to calculate one of two most optimal paths. Dijkstra's algorithm remained the most inefficient algorithm with its work done covering almost the entire graph and BFS remained more efficient than A*. Furthermore, both the BFS algorithm and A* decreased in efficiency.

Figure 3.4.5 - The cross wall graph test: *(Xu, 2012)*



Increasing the complexity of the graph again results in A* and BFS reducing in efficiency. Figure 3.4.5 shows that A*s work done encompasses nearly the entire graph, similar to that of Dijkstra's work done, but the BFS algorithms work only covers around half of the graph. This jump in complexity did not have an effect on the BFS algorithms efficiency or the ability for all three of the algorithms to find one of the most optimal paths.

Figure 3.4.6 - The maze graph test: *(Xu, 2012)*



The final graph featured a much larger spike in complexity from the prior graph than any of the previous graphs did. Due to this graph being much more complex, it was the first graph where an algorithm did not calculate one of the most optimal paths. The BFS algorithm calculated a path which was not as optimal as it could have been, but the other two algorithms still manage to find one of the most optimal paths. Each algorithm also peaked in efficiency at this graph, although this may be due to the reduced number of options which the algorithm could choose at any time (due to more walls).

An algorithm will generate a path with 100% efficiency if work is only done where the path is calculated. For example, if there is 10 units of work done and the calculated path is 10 units in size, the algorithm was 100% efficient in calculating that path because only the necessary work was done.

The efficiency can be calculated using the equation: $\frac{\text{Path length}}{\text{Work done}} \times 100$

Figure 3.4.7 - Table of results:

Graph type	Algorithm used	Squares covered (work done) in 'units'	Path length in 'units'	Efficiency (nearest whole %)
Plain graph	Dijkstra's	267	13	5
	BFS	39	13	33
	A*	39	13	33
Single wall graph	Dijkstra's	383	26	7
	BFS	102	26	25
	A*	216	26	12
Triple wall graph	Dijkstra's	353	26	7
	BFS	93	26	28

	A*	175	26	15
Encapsulated goal and root nodes graph	Dijkstra's	340	36	11
	BFS	155	36	23
	A*	288	36	13
Cross wall graph	Dijkstra's	361	39	11
	BFS	191	39	20
	A*	334	39	12
Maze graph	Dijkstra's	117	30	26
	BFS	52	36	69
	A*	82	30	37

The data in figure 3.4.7 shows that the Best-First Search algorithm is consistently the most efficient algorithm, but also proves that it does not always find the possible shortest path for more complex problems, while the A* algorithm and Dijkstra's algorithm always find the most or one of the most optimal paths.

In addition, the data implies that Dijkstra's algorithm is especially inefficient, even for very small or simple problems. The efficiency of Dijkstra's algorithm actually increases with the complexity of the graph, while the efficiency of BFS and A* generally decreases with an increase in graph complexity. Dijkstra's algorithm begins at only 5% efficiency for graph 1 and ends at an improvement of 26%, while BFS and A* start at 33% efficiency each on graph 1 and drop to 28% and 15% respectively by graph 3.

4. Conclusion and evaluation

4.1 Experiment limitations and possible solutions

The experiment had two main limitations which may have resulted in less accurate results being achieved and a less reliable conclusion being drawn.

- Limitation 1 – Reliance on untested software: the simulation software used was not tested beforehand, which left the experiment vulnerable to possible systematic errors[12] that may have been caused by unknown errors in the program. Systematic errors in the data set could have negative effects such as making an algorithm seem less or more efficient than it actually is.

  Possible solution: testing the simulation program before the experiment or comparing it with other pathfinding simulation programs to ensure there are no errors with it.

- Limitation 2 – Graph size and complexity limits: since the graphs had to manually designed and analysed, there was a restriction to how complex and large they could be. This limitation inhibited the ability to fully test the algorithms as the graphs were either too small or simple to promote calculation of a larger variety of paths.

  Possible solution: use of graph templates or imported graphs to allow for large scale, complex graphs to be used. Graph analysis software may also be an option to enable these large, complex graphs to be properly and accurately examined.

4.2 Conclusion

In conclusion, the A* algorithm is a strong pathfinding algorithm which consistently produces a shortest path through a process that is reasonably efficient for small and simple problems or large and complex problems. Its solutions are to that of the same standard as Dijkstra's algorithm, although are produced far more efficiently, like those of the Best-First Search algorithm. It evolved from them by adopting features from both and merging them into a single pathfinding formula which is more dynamic and suitable for a larger range of problems.

---

[12] A systematic error is an error which affects an entire data set or variable.

Although, the experiment hypothesis was partially disproven (as the BFS algorithm was always more efficient), not a wide enough range and variance in graphs was tested to assume the BFS algorithm will always be more efficient than A*. A*s guarantee in calculating a shortest path is what often makes it the more optimal algorithm. Its dynamic nature is what gives it the possibility to be more efficient than BFS in some scenarios.

Bibliography:

Websites/Blogs/Online articles and presentations

Eranki, R. (2002). *Pathfinding using A* (A-Star).* [online] Web.mit.edu. Available at: http://web.mit.edu/eranki/www/tutorials/search/ [Accessed 23 Jan. 2015].

Hadorn, B. (n.d.). *Graph theorie using Zeus-Framework*. [online] Xatlantis.ch. Available at: http://www.xatlantis.ch/examples/graph_example.html [Accessed 20 Feb. 2016].

Lee, D. (n.d.). *What is GPS?*. [online] Www8.garmin.com. Available at: http://www8.garmin.com/aboutGPS/ [Accessed 23 Jan. 2016].

Lin, D. (n.d.). *Animal Extinction - What is an Animal Extinction?*. [online] About.com. Available at: http://animalrights.about.com/od/wildlife/g/Animal_Extinction.htm [Accessed 25 Jan. 2016].

Muhammad, R. (n.d). *Greedy Introduction*. [online] Personal.kent.edu. Available at: http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm [Accessed 17 Sep. 2015].

Nykamp, D. (n.d.). *Function definition*. [online] Mathinsight.org. Available at: http://mathinsight.org/definition/function [Accessed 20 Feb. 2016].

Weisstein, E. (n.d). *Weighted Graph*. [online] Mathworld.wolfram.com. Available at: http://mathworld.wolfram.com/WeightedGraph.html [Accessed 17 Sep. 2015].

Yan, M. (n.d). *DIJKSTRA'S ALGORITHM*. [online] http://math.mit.edu/. Available at: http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf [Accessed 17 Sep. 2015].

cs.ubc.ca, (2007). *Heuristic Search*. [online] Available at: http://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202006-7/Lectures/lect6.pdf [Accessed 18 Sep. 2015].

cs.utexas.edu, (n.d.). *Heuristic Search*. [online] Available at: http://www.cs.utexas.edu/~mooney/cs343/slide-handouts/heuristic-search.4.pdf [Accessed 18 Sep. 2015].

Dictionary.com, (n.d.). *migrate*. [online] Available at: http://dictionary.reference.com/browse/migrate [Accessed 25 Jan. 2016].

En.cnki.com.cn, (2013). *Research on algorithm of intelligent path finding in game development--《Computer Engineering and Design》2006年13期*. [online] Available at:

http://en.cnki.com.cn/Article_en/CJFDTOTAL-SJSJ200613008.htm [Accessed 23 Jan. 2016].

Let.rug.nl, (2000). *Comparison with Dijkstra's algorithm*. [online] Available at: http://www.let.rug.nl/~vannoord/papers/nle/node35.html [Accessed 17 Sep. 2015].

Redblobgames.com, (n.d.). *Introduction to A**. [online] Available at: http://www.redblobgames.com/pathfinding/a-star/introduction.html [Accessed 23 Jan. 2016].

TheFreeDictionary.com, (2011). *algorithm*. [online] Available at: http://www.thefreedictionary.com/algorithm [Accessed 25 Jan. 2016].

TheFreeDictionary.com, (2014). *ecosystem*. [online] Available at: http://www.thefreedictionary.com/ecosystem [Accessed 25 Jan. 2016].

Theory.stanford.edu, (n.d.). *Introduction to A**. [online] Available at: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html [Accessed 23 Jan. 2016].

Web.stanford.edu, (n.d.). *BEST - FIRST*. [online] Available at: http://web.stanford.edu/~msirota/soco/best.html [Accessed 18 Sep. 2015].

Wiki.roblox.com, (n.d.). *Best-first search*. [online] Available at: http://wiki.roblox.com/index.php?title=Best-first_search [Accessed 18 Sep. 2015].

Yourdictionary.com, (n.d.). *Pathfinding*. [online] Available at: http://www.yourdictionary.com/pathfinding [Accessed 18 Sep. 2015].

Books/E-books

Zhang, Y. (2012). *Path-Finding Algorithm Application for Route-Searching in Different Areas of Computer Graphics*. 1st ed. [ebook] Published online: InTech, pp.1-2. Available at: http://cdn.intechopen.com/pdfs-wm/29857.pdf [Accessed 23 Jan. 2016].

Heuristic shortest path algorithms for transportation applications: State of the art. (2005). 1st ed. [ebook] Unknown: Computers & Operations Research, pp.1-3. Available at: http://www.civil.uwaterloo.ca/itss/papers%5C2006-3%20(Review%20of%20heuristic%20SPA).pdf [Accessed 23 Jan. 2016].

Software

Xu, X. (2012). *Pathfinding.js*. MIT.

Videos

Fan, N. (2012). *Dijkstra's Algorithm*. [video] Available at: https://www.youtube.com/watch?v=gdmfOwyQlcI [Accessed 17 Sep. 2015].