Comparing the Performance of Depth First Search and Breadth First Search for a Robot with Energy Constraints Exploring an Unknown Environment

Shyam Rajendren
School of Computing
University of North Florida
Jacksonville, Florida
Email: n01185608@unf.edu

***Abstract*—This paper identifies a performance metric in which to compare algorithms that are used to search an unknown environment when a robot has an energy constraint. The two algorithms focused on in this paper are Depth First Search (DFS) and Breadth First Search (BFS). This paper hopes to contribute a standardised evaluation metric to compare algorithms so that most optimal ones can be used in real-life situations.**

I. Introduction

In many scenarios we must rely on robots to explore environments which humans cannot. A couple of examples would be deep ocean floor mapping and terrestrial object explorations (such as Mars). Robots are also helpful in solving problems closer to home as searching for missing persons in disaster zones. However, considering the expenses necessary to carry out such operations, we must make sure that the robots search their environments in the most efficient and cost effective manner possible.

There are many algorithms that can be used to help a robot traverse an unknown environment but when selecting an algorithm for a robot to use in big operations we would like to pick the one which performs the best. In order to do this there must be a standard evaluation metric which we should use to compare the performance between different algorithms. This paper hopes to do just that by comparing Depth First Search (DFS) and Breadth First Search (BFS) using a standard performance metric.

Comparing and creating new algorithms to search an unknown environment is not new but hardly has it been done with a robot's energy capacity kept in mind. However, in a real world environment, a robot would not have an infinite energy supply so it is prudent for us to consider this constraint when comparing algorithms.There have been a few studies where researchers evaluate the performance of their algorithms when a robot is given this energy constraint, but their performance metrics vary which makes it difficult to asses which algorithm actually performs better.

In this paper we will discuss previous work done by fellow researchers in the field as well as discuss our own work. We will then outline the methodology we used as well as the evaluation metric used to compare the two algorithms. Finally, we will discuss our results as well as any future work.

II. Previous Work

While few in number there has been research done by other scholars in this topic. For instance, Strimel and Veloso used a boustrophedon decomposition to cover an unknown environment [1]. In this method, the robot returns to the charging station when its energy level is too low to continue the coverage. Mishra *et al.* designed a coverage planning algorithm which uses multiple robots to traverse the unknown environment [2]. This method involves splitting the robots into two groups, workers and helpers. When a worker needs to recharge its battery, an associated helper will continue the worker's coverage. Though these studies added useful information to the topic, neither of them formally analyzed their algorithms and instead only looked to prove that their methods correctly covered every point in the environment. Through this paper, we have set a precedent of comparing algorithms based on the same parameters. This will help decide which algorithms are superior so that they can be used practically.

III. Methodology

We will be using a novel method in order to compare the DFS and BFS algorithms. The goal of this project is for a robot to traverse every point in an unknown environment while having to return to a charging station (which will be placed at the robot's starting point) whenever its battery runs low. Thus the starting point and end point will be the same as the robot must return to its charging station once its traversal is complete. The input for this project is a grid environment with a random distribution of obstacles which will be traversed using DFS and BFS. The grid environment here is simply a graph with nodes and edges that resembles a square grid.

Depth First Search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root of the tree and traverses as far as possible down the tree's branch before backtracking. The time complexity for this algorithm is $O(b^d)$ where b is the branching factor, in this case

two, and d is the depth of the tree. Breath First Search (BFS) is similar to DFS except it traverses all the nodes at the present depth of the tree before going to the next depth level. The time complexity for this algorithm is also $O(b^d)$. The environment can be traversed in all four cardinal directions and each node in the grid has a maximum of two parents and two children. The output here will be the path/paths taken by the robot to traverse the entire unknown environment.

Due to how the DFS and BFS algorithms work, some points will be missed and this will increase with the number of obstacles present in the environment.In order to effectively search an unknown environment, the robot must start at or close to the center of the environment. The energy of the robot is equivalent to the path length. For example, if the robot moves from (0,0) to (0,1) (an euclidean distance of 1), the robot's energy will decrease by 1. As such the energy constraint placed on the robot must be greater then twice the length/width of the grid environment. That way the robot can reach the extremes of the environment (the maximum depth in case of DFS and maximum breadth in case of BFS). In our simulations, we kept the energy constraint equivalent to three times the length/width of the environment. For example, in a 10x10 grid environment, the energy constraint = 10 x 3 = 30.

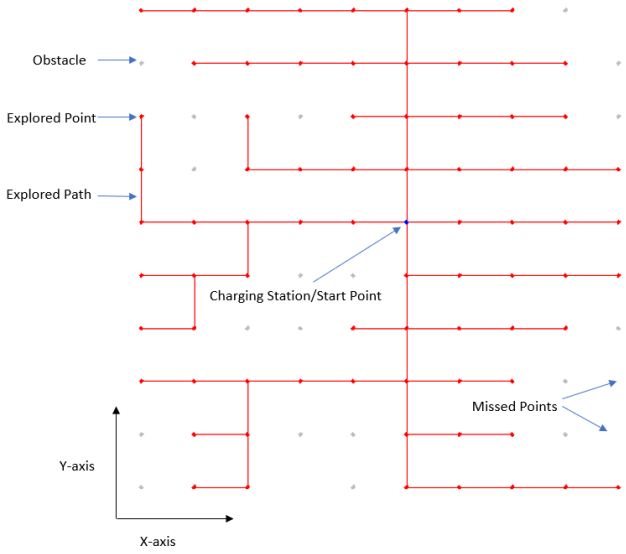

Fig. 1. Labelled example of an output using Depth First Search in a 10x10 Environment

Both DFS and BFS require a normal tree's depth and breadth in order to function. However, in a grid environment such as ours, there is both a positive and negative depth (y-axis) and breadth (x-axis). Therefore, in order for DFS and BFS to explore the entire environment, we must split the environment into 4 quadrants and perform DFS and BFS separately on each one of them. We use the x-axis and y-axis of our grid as a makeshift depth and breadth in order to implement the DFS and BFS algorithms. The main axes in this environment will be centered on the charging station/start point of the robot. For example, if the robot starts at (5,5), then the main axes would be x=5 and y=5. As the effectiveness of DFS and BFS in our simulated environment revolves around the main axes in the environment, any obstacles present on one of these axes will severely hurt the performance. As such in the environment we create, we will ensure that there are no obstacles present on the main axes of the environment. This is obviously extremely unlikely in a real-world scenario but we can do this in our case as we are not trying to prove the proficiency of BFS and DFS but rather we are trying to compare the two algorithms.

IV. Evaluation

A simulation approach will be used to solve this problem. Three environments were created: a 10x10 environment, a 50x50 environment, and a 100x100 environment. Each environment had a random obstacle distribution where the obstacle percentage can be either 10%, 20%, or 30%. The environments were created using Graph Stream, an open source Java graph handling package available online [3]. The data sets we will collect are time taken to search the environment, number of trips, energy consumed (equivalent to the sum of all the path lengths), and the percentage of points unexplored.. This data will help serve as a simple evaluation metric to compare the two searching algorithms.

V. Results

The following three tables show the data that was collected. Note the values shown are the average of 5 separate test runs. This was done in order to get more accurate results. Energy consumed and total number of trips have been rounded to the nearest integer. The yellow highlight indicates the better result for a particular metric. The blue highlight indicates a tie in the two results.

10x10 Environment			
		Depth First Search	Breadth First Search
10% obstacle percentage	Time Taken (Microseconds)	108.6	88.4
	Total No. of Trips	11	11
	Energy Consumed	231	277
	Percentage Unexplored	0.4	0.2
20% obstacle percentage	Time Taken (Microseconds)	86.8	166.8
	Total No. of Trips	11	10
	Energy Consumed	262	258
	Percentage Unexplored	1.6	1.6
30% obstacle percentage	Time Taken (Microseconds)	81.4	58.6
	Total No. of Trips	9	9
	Energy Consumed	224	220
	Percentage Unexplored	5	4.6

Fig. 2. Results obtained from the 10x10 Environment

50x50 Environment			
		Depth First Search	Breadth First Search
10% obstacle percentage	Time Taken (Microseconds)	1007.8	996.8
	Total No. of Trips	48	48
	Energy Consumed	6910	6830
	Percentage Unexplored	0.976	0.976
20% obstacle percentage	Time Taken (Microseconds)	899.6	981
	Total No. of Trips	41	41
	Energy Consumed	4704	5848
	Percentage Unexplored	4.632	4.632
30% obstacle percentage	Time Taken (Microseconds)	808.6	797.4
	Total No. of Trips	32	32
	Energy Consumed	4495.6	4517.6
	Percentage Unexplored	15.488	15.488

Fig. 3. Results obtained from the 50x50 Environment

100x100 Environment			
		Depth First Search	Breadth First Search
10% obstacle percentage	Time Taken (Microseconds)	3961	3939.8
	Total No. of Trips	94	94
	Energy Consumed	27669	27511
	Percentage Unexplored	1.144	1.144
20% obstacle percentage	Time Taken (Microseconds)	3687.4	3918.4
	Total No. of Trips	80	83
	Energy Consumed	23543	23624
	Percentage Unexplored	5.422	5.422
30% obstacle percentage	Time Taken (Microseconds)	2192.8	3521.8
	Total No. of Trips	61	62
	Energy Consumed	17603	18228
	Percentage Unexplored	16.004	16.004

Fig. 4. Results obtained from the 100x100 Environment

VI. Conclusion

From our results, we can see that DFS and BFS essentially perform the same when trying to traverse and unknown environment. In fact, in almost all the test runs we find that both algorithms miss the exact same points. Though there are scenarios in which one performs marginally better than the other, it is inconsistent and any variations in results is probably due to the particular environment that was produced and no inherent superiority in either algorithm. In fact, it is very likely if that a far greater number of tests were run, we would find that any differences in the results in each performance metric would all but vanish.

These results are not surprising though in fact this is actually what one would expect. The DFS and BFS search algorithms are meant to search a tree/graph and how well they perform relies almost solely on where the data in question has been placed in the tree/graph. However, in our case, we require the algorithms to search every single point in the environment which would be equivalent to the worst-case scenario for both algorithms. The worst-case scenario for DBS and BFS are the same as they both have to search every point.

In conclusion, we can say the DFS and BFS are not suitable for exploring an unknown environment as is made quite clear by the percentage of unexplored points in every scenario that was tested. We can also say the both algorithms perform almost equally and any difference in performance results is due to where the obstacles are present in the environment.

VII. Future Work

In the future we would like to test theses algorithms in a real-world environment so that we may get a better gauge of an algorithm's proficiency. We would also like to compare more search algorithms using the standardised performance metric which was devised in this paper.

References

[1] Grant P. Strimel and Manuela M. Veloso. 2014. Coverage planning with finite resources. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), 2950–2956

[2] Saurabh Mishra, Samuel Rodríguez, Marco Morales, and Nancy M. Amato. 2016. Battery-constrained coverage. In *CASE*. IEEE, 695–700.

[3] GraphStream - A Dynamic Graph Library.[online] Available at: https://graphstream-project.org/