

benchmarking report

Fundamentals of AI



May 29, 2022

AAIT

Addis Ababa

*Name: Yoseph Habtamu*

*Section: 2*

*Id: UGR/9364/12*

**BFS (breadth first search):**

This graph searching algorithm finds the destination node by traversing all its child node and traversing their children one node at a time.

To implement BFS algorithm I used a list as a queue to keep track of the other nodes. Queue stores data in FIFO order so, when a node is traversed the child nodes are stored in order. Then the next step is popping the first item and appending its untraversed children nodes to the queue until there is no untraversed node in the queue

**DFS (depth first search):**

This graph searching algorithm finds the destination node by traversing one child node and traversing one of its child one node one by one until it reaches the end of the graph or a node that is already traversed.

To implement DFS algorithm I used a list as a stack to keep track of the other nodes. Stack stores data in LIFO order so, when a node is traversed the child nodes are stored in order so that popped node is child of the previous node. And this continues until there is no node left in the stack.

**Dijkstra’s shortest path algorithm:**

This algorithm works for weighted graphs (graphs where the value of the weights is not equal and the edges taken between the nodes matter). Dijkstra is a greedy algorithm that takes the shortest path among the list of edges connected to the node.

To implement this algorithm, I used a dictionary to keep track of the nodes name as a key and smallest amount of weight that a user takes to reach the node from the initial as a value and another dictionary to store the node and its immediate parent through which the node reached the destination and a heap library from python called heapq to sort find the smallest among the edges of the node.

**A star search:**

A variant of dijkstra’s algorithm, a star tries to find shortest path between two nodes in a graph using a heuristic function. Heuristic function is a guide that tells us how much we are getting close to the destination. So, when traversing through the graph, unlike dijkstra which takes solely the weight we take into account the heuristic.

Enough with the intro and let’s deep dive to the analysis of each algorithms using different benchmarks.

1. **Time:**

**BFS vs DFS**

Both BFS and DFS have a time complexity of O (v+e) where v is vertex and e is edge but does this means they take the same time to traverse through a given graph?

My answer is no. whether BFS or DFS is faster depends on the given graph i.e. the way the algorithm traverse through the node and the chosen destination.

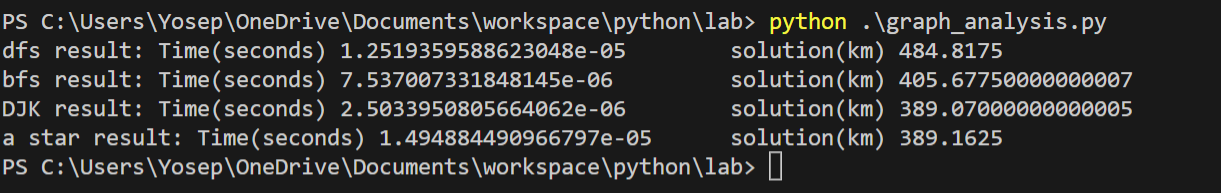
For example:

* If DFS algorithm starts traversing from left to right and the destination is immediate child of the initial node, then the algorithm traverses a lot of unnecessary nodes which makes it slower if we compare it to BFS which finds it way faster
* Another scenario, what if the there are a lot of children start and end node and we chose BFS the we have to go through all its children node but if we chose DFS we could have get a lot faster

But in our assignment case, the result found between the BFS and DFS is in favor of BFS for that there can be reasons worth mentioning.

* The nature of the graph: the graph is not highly interconnected considering the number of nodes of the graph.
* Implementation of the algorithms

Statistical comparison:



The above image shows the average time in second for the graph found in page 82 of the text and solution length in kilo meter (we will discuss this in detail)

This data represents the average time a node takes to go to another node in different algorithms

So on average why is BFS faster for this graph?

The main reason behind the result is that the graph is not deep enough for the BFS algorithm to traverse and the child of each node is very small (the maximum is 4).

**Djikstra vs A star:**

When it comes to which one is faster among the two there is no clear answer. But in theory a star should be faster. Wait what does that mean?

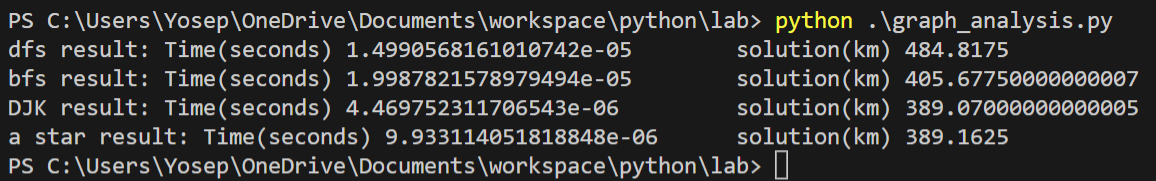
The path a star algorithm chooses is the one with smaller heuristic, unlike dijkstra which chooses the smallest weight. But what to choose as a heuristic function and its implementation depends on the programmer and this opens up margins for error. Sometimes. The heuristics might be biased which makes either a star slower or give path that is not the shortest. how ever, a good heuristic not only shortens the search time but also gives the path close to its corresponding dijkstra algorithm.

On the other end however, dikstra always guarantees the shortest path between any two nodes. But, The main drawback is it gets slower after some threshold. That is because dijkstra, to find the shortest path between two nodes, it has to traverse through the whole graph and that is not cheap computation and as the number of nodes increases the time for computation almost becomes squared (my algorithm runs around O() ) so after a while it even could take minutes if the node are large enough.

The main thing to keep in mind is, to choose between a star and dijkstra really depend on the application area. For example, if performance and speed is need over accuracy then using a star is preferred. Best examples in video games. However, in other aspects where the accuracy of the data crucial over performane then dijkstra is the better candidate.

For our assignment the average time for dijkstra and a star using my implementation is shown below.

The heuristic function I used for this assignment is the haversine formula which is used to find the distance between two points in a sphere accurately



To test the result please go to graph\_analysis.py and call the method analyze with graph and heuristic as a parameter

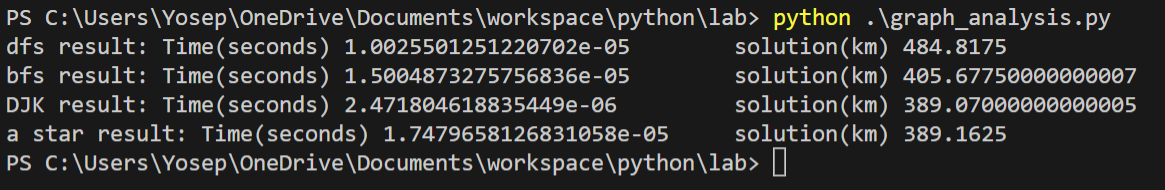
My dijkstra and a star search time result is almost equal that is because of the small number of nodes of the graph and most importantly small number of edges I will clearly demo the difference in the Bonus section below.

1. **Solution Length:**

Solution length is the average distance the algorithm finds for the connected nodes. That means the smaller solution length means the algorithm connects the two nodes with smallest distance coverage possible. So Which one of the four algorithm is the best? why?

**BFS and DFS:**

These algorithms, in my opinion are not best suited for this case. The main reason is they don’t consider the distance between the edges as relevant. They treat all the edges as equal (unweighted graph) so the solution found by these algorithms to be the smallest is purely dependent on luck or if the graph has some kind of pattern that these algorithms can follow. That is why the result in my implementation (shown below) is far off from the ideal (djikstras algorithm)

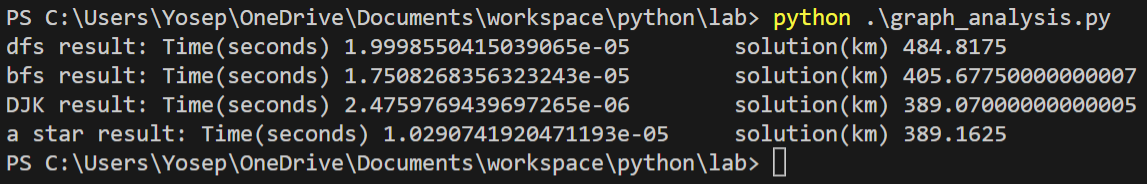


Clearly the algorithms are not close to the ideal.

**Djikstra vs a star:**

When it comes to solution length the most ideal algorithm is djkstra. Because the algorithm chooses the smallest distance that brought it to the current node as ‘ancestor’ and continues to choose the smallest among its child nodes. So it is always guaranteed to find the shortest path from a weighted graph.

In a star search algorithm, it changes a little bit because of the heuristic, we are not guaranteed to find the shortest path however if we have pretty good heuristic function we get a result pretty close to the djikstra’s result.



The above image shows there is only slight difference between dijkstra and a star solution length which shows my heuristic function is pretty good.

What happens to the average time and solution length when the node and edge of the graph increase??

In this part of the report I will discuss what happens to the algorithms when data set increases for these I choose the following parameters

15 nodes and 30 edges

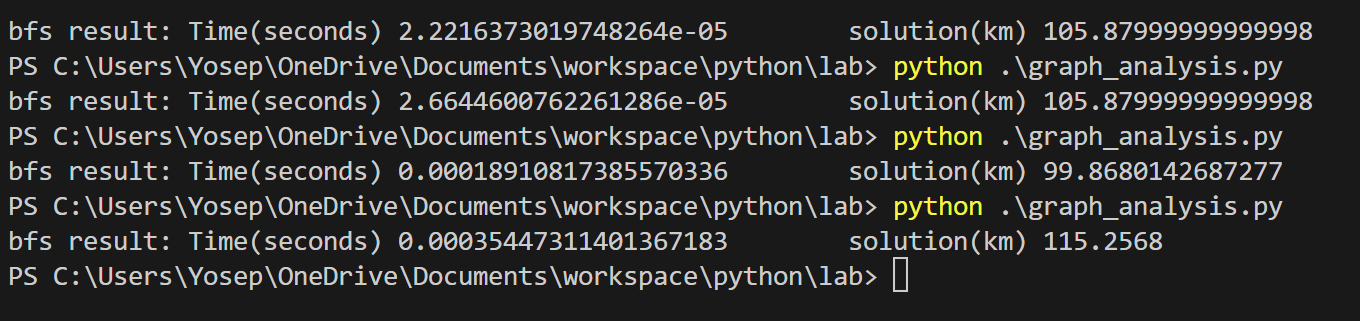
20 nodes and 70 edges

30 nodes and 150 edges

50 nodes and 250 edges

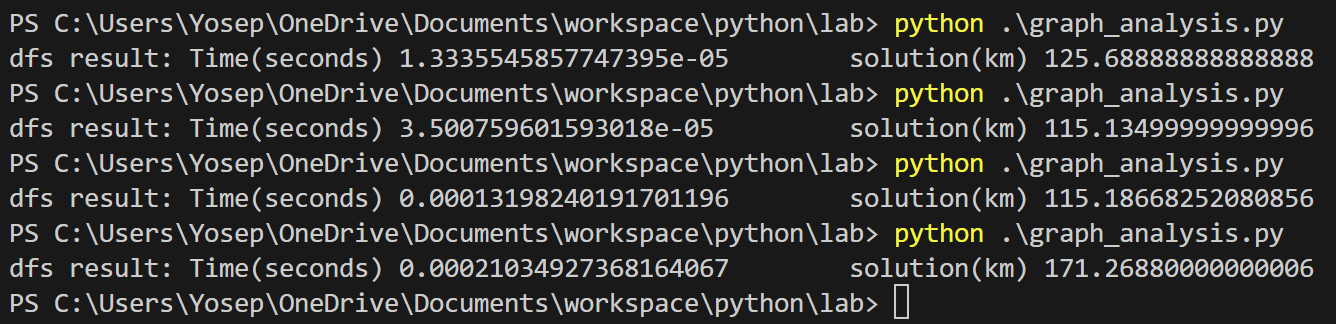
With the above datasets I will bench mark each of the 4 algorithms.

1. BFS\_search:



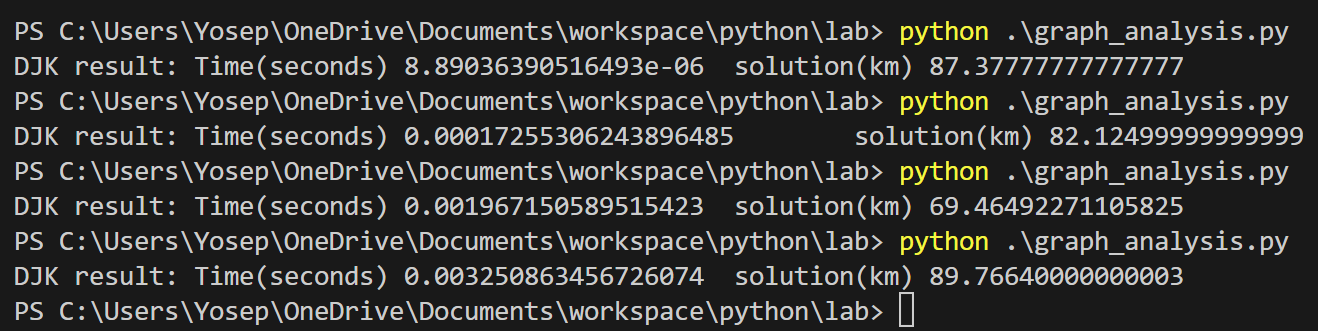
The above image shows how the average time to traverse a graph behaves when the number of nodes and edges keep increasing. From this I concluded that for the same algorithm If the number of nodes and edges increase then the time increases

1. DFS\_search:



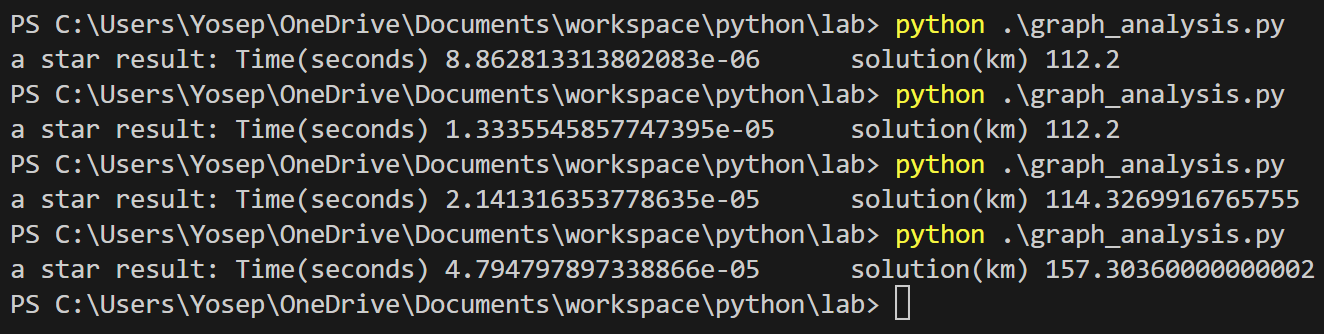
The above images shows as the number of nodes and edges increase the time to find the end node using Bfs algorithm also increases

1. Dijkstra\_algorithm



The above image shows as the number of nodes and edge increase the time to traverse and find the shortest time becomes very difficult for dijkstras algorithms that is why its time very large but it has the smallest solution length among all the other algorithms.

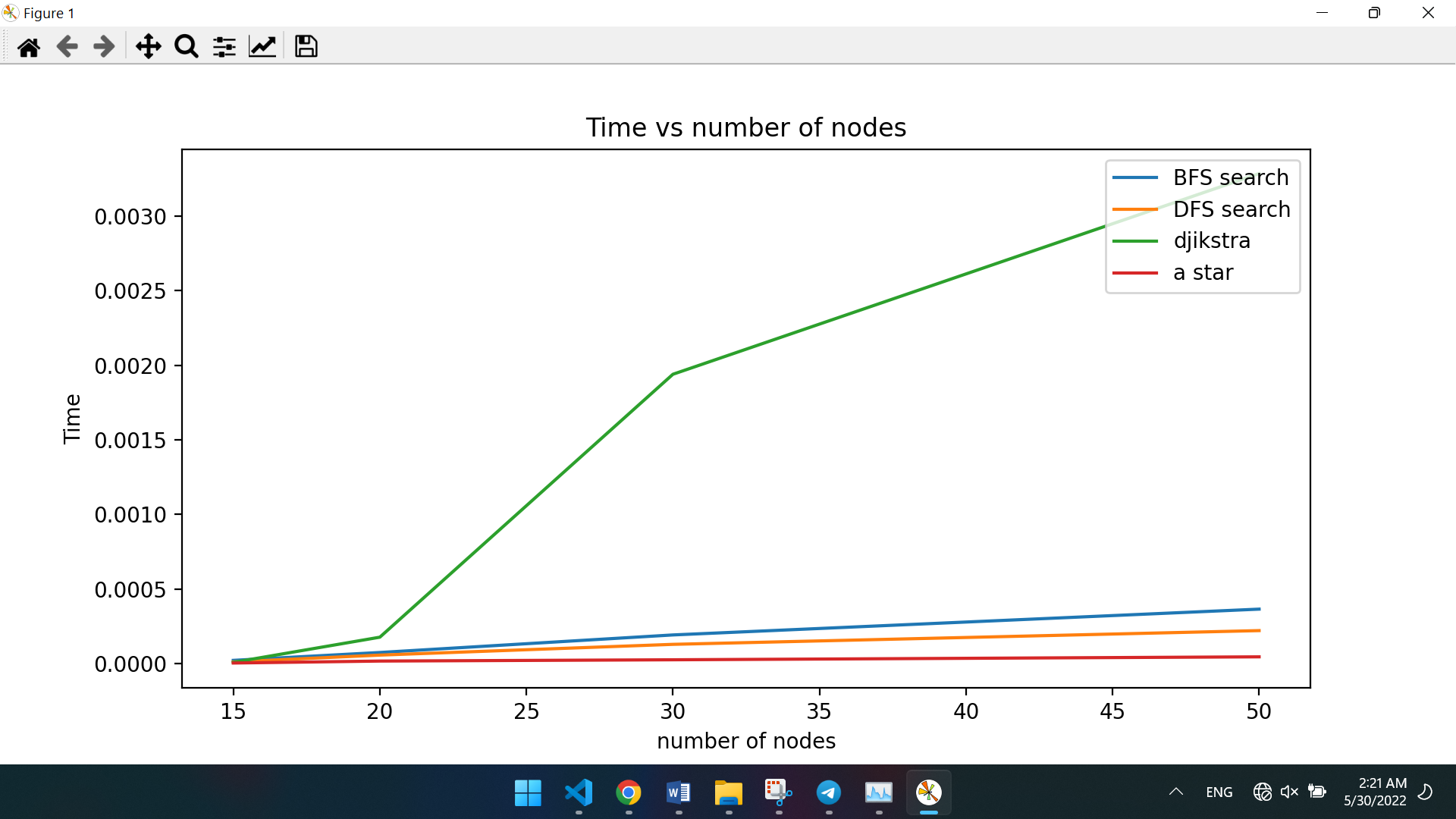
1. A\_star\_search:



According to the data above a star is by far the fastest among all other algorithms. But to achieve this speed the algorithm sacrificed the accuracy of the shortest path but as I mentioned earlier this is a common trade off when using a star search

To summarize all the data and visualize it I used matplotlib to plot the graph of

1. Average time vs number of node



1. Average solution length vs number of node

