Enhancement of Dijkstra Algorithm for Finding Optimal Path

Alec Zehst Tiong[1], Celeste June Panganiban[1], Mark Christopher Blanco[1], Richard Regala[1], Dan Michael Cortez[1]

[1]azmtiong2018@plm.edu.ph
[1]Computer Science Department, College of Engineering and Technology
Pamantasan ng Lungsod ng Maynila (University of the City of Manila)
Intramuros, Manila 1002, Philippines

Abstract

The Dijkstra algorithm is a common method used when solving shortest path problems. It is a graph-based method that compares node distances, selects the shortest subsequent node, and generates an ideal path. However, it is seen that the method struggles with memory utilization, particularly when working with substantial amounts of data or graphs. The purpose of this research work is to improve the algorithm to tackle the current problem. The enhancement is accomplished by providing an approach in which the two closest nodes are combined in each iteration. With this, the conclusion of the study is that the enhanced Dijkstra algorithm, with the application of node combination, was able to reduce the memory usage in comparison to the existing Dijkstra algorithm.

Keywords: Dijkstra Algorithm; Node Combination Algorithm; Memory Consumption

1. Introduction

People travel from one location to another to do a variety of activities. From each house, a person can go to the market to buy groceries, a student can go to school for academic activities, and an employee goes to a company to earn a living. People can memorize each path they go through for them to continually visit a place on multiple occasions. On the other hand, people tend to be lost while moving from one location to another due to the limitations of the human intellect. The development of mobile navigation tools such as Waze and Google Maps began, assisting users not only in locating their chosen locations, but also in providing accurate instructions on how to go to their destinations correctly.

The Dijkstra algorithm was created by Edsger Dijkstra, a Dutch computer scientist. It can be used for pathfinding between nodes in a graph, such as a road network or path. It needs parameters for the origin and destination locations. The Dijkstra algorithm is a well-known method for determining optimum pathways. According to Qing et al. (2017) it is a popular shortest route method. The Dijkstra algorithm is simple to develop, runs consistently, and adapts well to topology changes. In addition, Wayahdi et al. (2021) said that it is the most effective method for resolving the simple shortest path problem. Dijkstra's algorithm is one of the greedy strategies used to solve shortest path problems. The procedure, according to Gbadamosi et al. (2020), is used to determine the shortest paths to the vertices of a graph in the order of their distance from a specific source. Even though the traditional Dijkstra's Algorithm solves the shortest path problem, it may not be the best option in some situations due to a variety of factors.

2. Existing Dijkstra Algorithm

2.1. Problem of Existing Dijkstra Algorithm

When utilizing applications on mobile devices, consumers consider the size of the application itself to determine whether it requires a lot of storage. If the application's size is large and their devices' storage capacity is limited, the application's performance may suffer. As a result, the process is slowed. Memory utilization is another issue with the Dijkstra Algorithm. According to Fitro et al. (2018), calculations, which can be conducted in a period when checking one destination to another, are difficult to conduct when finding ideal output. That is the reason there is quite a challenge when dealing with the task of determining the shortest path, which is considered as an essential case study in computer science. In addition, Yalin et al. (2017) said that the algorithm has features which have larger time complexity and scope, but with lower efficiency of the searching process. People also calculate the amount of data produced by the program. People dislike it when programs generate a significant quantity of data, sacrificing the storage capacity of their devices. Fitro et al. (2018) said that the Dijkstra technique requires memory space as points keep being traversed with each iteration. It is considered a disadvantage when looking for the shortest path, especially with large networks. Hence, the researchers propose a modified version of the enhanced Dijkstra Algorithm of Qing et.al (2017) to solve the existing problem when it comes to memory usage.

2.2. Pseudocode of Dijkstra Algorithm

```
Load Data (Distances, points).
Initialization. Mark the starting node v and add it into S.
Traverse the nodes in V-S and select all the adjacent nodes as the candidate intermediate nodes.
Select the node i with the smallest number among the candidate intermediate nodes and add it into S set.
Regard i as the new intermediate node. Repeat (3) and (4) and choose the smallest numbered node j from
the adjacent nodes. Update the distance between the source node v and the node j.
        If DIST(j) > DIST(i) + C(i,j), modify DIST(j) to DIST(j) = DIST(i) + C(i,j), and add node j
        into S set.
Repeat (3)-(5) n-1 times. All the shortest paths from the source node to the goal node are stored in
DIST(X) when the search iteration traverses to the goal node.
```

3. Enhanced Dijkstra Algorithm

3.1. Enhancement of the Algorithm

The Dijkstra algorithm does not handle memory usage well. When looking for the shortest route from a large graph, Dijkstra's algorithm consumes a lot of memory to keep the points traversed over each iteration. For this matter, the researchers would introduce another method that lessens the memory usage while using Dijkstra Algorithm. The researchers intended to enhance the Dijkstra algorithm using the Node Combination Approach. According to Fitro et al. (2018), by the term itself, combination of two nearest nodes is done. These nodes are expressed by the vertex weights between the two. By merging the nodes with the least distance into a single node, this node combination technique saves memory on each iteration. It is supported by Amaliah et al. (2016), stating that deleting nodes after merging is an efficient way of memory utilization than complying with the traditional way of using Dijkstra Algorithm.

3.2. Pseudocode of Enhanced Algorithm

```
Initialization of graph, start and goal node
Initialization of toDelete, path, predecessor

while start has more than 1 child:
  place children of start to a temporary graph
  for each child and weight of start:
    if child is not the goal:
      for each grandchild and weight of grandchild of start:
        save the total of weight of child and weight of grandchild in path
        place child as predecessor of grandchild
        add child to toDelete

  for node in toDelete:
    if the node is not in path:
      pop the node in the temporary graph
      if node is in predecessor:
          pop the node in the predecessor

  clear start of graph

  for node and weight of path:
    insert new children and weight of children of start

  Clear path, temporary graph, and toDelete

final answer = goal
add the nodes in the predecessor to the final answer
add the start node in the final answer
reverse the final answer

output final answer
```

4. Methodology

This study employs an experimental research design. This research design was adopted by the researchers to keep track of the variables that could influence the study's outcome. This type of research design is used to find the difference in the results when the node combination algorithm is included in the enhanced Dijkstra Algorithm.

The use of experimental research allows researchers to demonstrate the relationship between the variables in the study. Independent variables are monitored in order to learn how they may influence dependent variables. With this explanation in mind, performing this research through experimental study is an adaptive decision that aids the researchers in fixing the current problem of the Dijkstra Algorithm.

Other resources were also used within the study. This includes online articles, journals, and existing research. With these resources, the researchers formulated a methodology that best fit the study. These resources were used as references in collecting data and producing solutions in solving the problem of the algorithm. As a result, the study's application is dependent on these resources, allowing for the algorithm to be improved by incorporating the idea of node combination.

4.1. Adding Node Combination Algorithm to the Enhanced Dijkstra Algorithm

The Node Combination Approach is the method that the researchers planned to use to improve the Dijkstra algorithm. According to Fitro et al. (2018), by the term itself, combination of two nearest nodes is done. These nodes are expressed by the vertex weights between the two. By merging the nodes with the least distance into a single node, this node combination technique saves memory on each iteration.

4.2. Combining the weight of the child node and the grandchild node

Before starting the process of combining nodes, a toDelete list will be initialized that will contain the nodes identified the child node excluding the goal node. In the process of traversing through each node in a specific path, the deletion of the child nodes will be done after checking if the child node is the goal node. Then it will check if the node to delete is in the path. If the node is not in the path, then deletion of nodes inside the toDelete should take place. After the deletion of nodes, changes happen within the value of the child nodes and the weights which are the sum of the previous child and grandchild. The last step is the clearing of the nodes inside the path, temporary path, and toDelete lists.

4.2.1. Pseudocode of Node Combination

```
for node in toDelete:
    if the node is not in path:
        pop the node in the temporary graph

clear start of graph

for node and weight of path:
    insert new children and weight of children of start

Clear path, temporary graph, and toDelete
```

4.2.2. Graph of nodes and distances

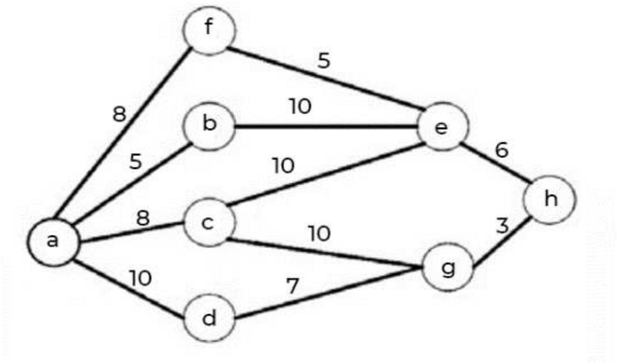


Fig. 1. Illustration of a graph of nodes with their distances

Figure 1 is an example diagram that will be used for this study. It shows a graph of nodes and their corresponding distances from each other. It contains the parent nodes, child nodes, and the grandchild nodes. The grandchild node stands for the child of the child nodes. This diagram is based on the study of Lu et.al. (2011), Finding the shortest paths by node combination.

Table 1. Possible Paths with corresponding Total Distances

Path No.	Nodes	Distance
1	a-f-e-h	8+5+6 = 19m
2	a-b-e-h	5+10+6 = 21m
3	a-c-e-h	8+10+6 = 24m
4	a-c-g-h	8+10+3 = 21m
5	a-d-g-h	10+7+3 = 20m

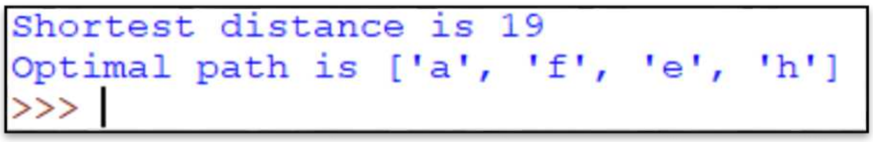
Table 1 above displays all of the pathways that may be created using the existing graph. There are a total of 5 pathways that may be formed, each with a different distance. Path 1 is 19m long and contains nodes A-F-E-H. Path 2 is 21m long and has nodes A-B-E-H. Path 3 has a total path length of 24m, with nodes A-C-E-H. Path 4 has a total path length of 21m, with nodes A-C-G-H. Path 5 has a total distance of 20m and nodes A-D-G-H. Path 1 has the shortest distance of all the pathways created from the graph, with 19m, making it the shortest and optimum path of the graph.

4.2.3. Comparing Existing Algorithm with Enhanced Dijkstra Algorithm in terms of Memory Consumption

To determine the memory allocated for the processes of the algorithms, the tracemalloc module in Python's library will be used, specifically to identify the starting memory and the peak memory. The researchers use the values to know the memory usage by subtracting the peak memory value to the starting memory value. Then, the differences will be compared to see which has the smaller value.

5. Results and Discussion

5.2. Results in Existing Algorithm

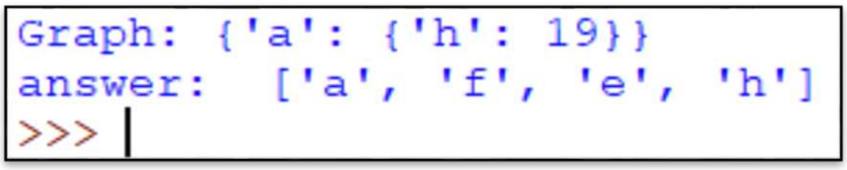


```
Shortest distance is 19
Optimal path is ['a', 'f', 'e', 'h']
>>> |
```

Fig. 2 The result after running the Traditional Dijkstra Algorithm while using the graph at Fig. 1

Figure 2 shows that the Traditional Algorithm successfully found the optimal path with the shortest distance. Among all paths that can be chosen within the given graph at Fig 1, the Traditional Dijkstra Algorithm still chose Path 1 as the optimum path with a total distance of 19m.

5.3. Results in the Enhanced Dijkstra Algorithm



```
Graph: {'a': {'h': 19}}
answer:  ['a', 'f', 'e', 'h']
>>> |
```

Fig. 3 The result after running Dijkstra Algorithm with the application of Node Combination while using the graph at Fig. 1

Figure 3 shows that the Enhanced Dijkstra Algorithm chose Path 1 as the optimum path among all existing paths within the graph. Within the figure shows the start node A and the goal node H with a total weight of 19, which is the total distance of the chosen path. This shows that the enhanced algorithm successfully chose the optimum path in terms of distance.

5.4. Comparison of Existing Algorithm and Enhanced Dijkstra Algorithm

Table 2. Analysis of Memory Consumption through the Use of Tracemalloc

Algorithm	Current Size	Peak Size	Difference
Existing Dijkstra	3456	4451	995
Enhanced Dijkstra	4448	5277	829

Table 2 shows the analysis of memory consumption of the existing and enhanced Dijkstra Algorithm where the researchers use the current size and peak size to find the difference to know the used memory of the algorithm. The existing algorithm's current size amounted to 3456 bytes and its peak size amounted to 4451 bytes, giving a difference of 995 bytes. Whereas the enhanced algorithm's current size amounted to 4448 bytes, and its peak size amounted to 5277 bytes, giving a difference of 829 bytes. Computing the difference between the current and peak size is done in order to show the gap of the memory consumed within the whole process in each algorithm. It can be interpreted that the enhanced Dijkstra algorithm successfully reduced the memory consumed in comparison to the existing algorithm. This result was obtained because of the implementation of the node combination method, which aims to reduce unnecessary nodes while looking for the optimum path from the start until the goal node.

Table 3. Analysis of Memory Consumption through the Use of Memory Profiler

Algorithm	Kibibyte	Megabyte
Existing Dijkstra	16632	17.0311685243
Enhanced Dijkstra	16612	17.0106877379

Another tool that was used to check the memory usage of each algorithm is the Memory Profiler. This tool checks the amount of kibibytes and the megabytes used by the algorithm while running the program. To explain the value of kibibyte and megabyte, 1 KiB is equivalent to 1024 bytes, whereas, 1 mb is equivalent to 1,000,000 bytes. As shown above, Table 3 contains the total quantity of Kibibyte and megabyte after the graph had been processed by the existing and the enhanced Dijkstra algorithm. The existing algorithm generated 16632 kibibytes, whereas the enhanced algorithm generated 16612 kibibytes, indicating that the existing algorithm is 20 kibibytes larger than the enhanced algorithm. Another distinction is the megabyte generated by the old method (17.0311685243) against the megabyte produced by the upgraded algorithm (17.0106877379). The existing algorithm is 0.0204807864 megabytes larger than the enhanced algorithm. Hence, the enhanced algorithm, though with a small margin, reduces the memory consumption in terms of kibibyte and megabyte when compared to the existing Dijkstra algorithm.

6. Conclusion

Following the completion of the study, the researchers can draw the following conclusions:

- The enhanced algorithm with the application of node combination was able to find the optimal path considering other possible paths with their corresponding distances.
- That combination of nodes in Dijkstra Algorithm can help with the reduction of memory consumption when looking for the optimal path based on distance.

7. Recommendation

Based on the findings of this study, the researchers can recommend the following:

- Using data that exists from actual locations
- Find out what other methods researchers can use to see if it can affect the results
- Use more complex graphs, e.g., graphs that contain a larger number of nodes and a larger number of weights
- Use different types of graphs that may affect the performance of the algorithm

Acknowledgements

The researchers are grateful to God for providing wisdom and direction, to the professors of Pamantasan ng Lungsod ng Maynila for their support and teachings; to their thesis adviser, who guided and assisted them in their research writing; and most importantly, to their family for providing inspiration to push through during the busiest days of their research; and their friends who have encouraged them in their endeavors, especially China Marie Lao, Clarissa Faye Gamboa, Enrico Sebastian Digman, and John Alfred Velasco for providing assistance in completing their study.

References

Qing, G., Zheng, Z., & Yue, X. (2017, May). Path-planning of automated guided vehicle based on improved Dijkstra algorithm. In 2017 29th Chinese control and decision conference (CCDC) (pp. 7138-7143). IEEE.

Wayahdi, M. R., Ginting, S. H. N., & Syahputra, D. (2021). Greedy, A-Star, and Dijkstra's Algorithms in Finding Shortest Path. International Journal of Advances in Data and Information Systems, 2(1), 45-52.

Gbadamosi, O. A., & Aremu, D. R. (2020, March). Design of a Modified Dijkstra's Algorithm for finding alternate routes for shortest-path problems with huge costs. In 2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS) (pp. 1-6). IEEE.

Fitro, A., P Sulistio Ilham, A., B Saeful, O., & Frendianata, I. (2018). Shortest path finding in geographical information systems using node combination and dijkstra algorithm. SHORTEST PATH FINDING IN GEOGRAPHICAL INFORMATION SYSTEMS USING NODE COMBINATION AND DIJKSTRA ALGORITHM, 9(2), 755-760.

Yalin, C., Liyang, Z., Longbiao, Z., Xiaojiang, S., & Heng, W. (2017). Research on path planing of parking system based on the improved dijkstra algorithm. Modern Manufacturing Engineering, 443(8), 63.

Amaliah, B., Fatichah, C., & Riptianingdyah, O. (2016). FINDING THE SHORTEST PATHS AMONG CITIES IN JAVA ISLAND USING NODE COMBINATION BASED ON DIJKSTRA ALGORITHM. International Journal on Smart Sensing & Intelligent Systems, 9(4).

Lu, X., & Camitz, M. (2011). Finding the shortest paths by node combination. Applied Mathematics and Computation, 217(13), 6401-6408.