Branch and Bound Algorithm to Determine The Best Route to Get Around Tokyo

Muhammad Iqbal Sigid - 13519152
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: sigid.iqbal123@gmail.com

***Abstract*—This paper described using branch and bound algorithm to determined the best route to get around six selected places in Tokyo, modeled after travelling salesman problem. The bound function used is reduced cost matrix using manual calculation and a python program to check the solution.**

Keywords—TSP, pathfinding, Tokyo, BnB

I. Introduction

Pathfinding algorithm is used to find the most efficient route between two points. It can be the shortest, the cheapest, the faster, or other kind. Pathfinding is closely related to the shortest path problem, graph theory, or mazes. Pathfinding usually used a graph as a representation. The nodes represent a place to visit and the edges represent the nodes connectivity. There are a lot of algorithm to do pathfinding and the most basic one is through exhaustive search, which explore each possible way until the goal is found. One of the most famous pathfinding algorithms is Dijkstra's algorithm.

Pathfinding usually used in real life, but it can also be applied to other stuff, such as video games. In real life, pathfinding can be used to find the fastest a route to a destination. This can be combined with GPS to make it easier to use for people, like in Google Maps for example. The same goes for video games, but because video games have a pixel-based maps, pathfinding in video games can be difficult. To solve that, they use hierarchical path. In this paper, I will be using pathfinding to determine the best route to get around Tokyo.

II. Fundamental Theories

A. Branch and Bound Algorithm

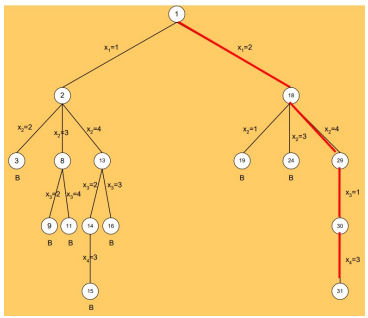

Fig. 1. State Space Tree example for Branch and Bound Algorithm (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf accessed May 8, 2021)

Branch and Bound is an algorithm design which are usually used to solve discrete and combinatorial optimization problems. Branch and bound uses state space tree by expanding its nodes (subset of solution set) based on the minimum or maximum cost of the active node. There is also a bound which is used to determine if a node can produce a better solution than the best one so far. If the node cost violates this bound, then the node will be eliminated and will not be expanded further.

Branch and bound algorithm is quite similar to Breadth First Search (BFS) algorithm with least cost search. BFS algorithm expand its tree based on the order of the active node. Apply least cost search to BFS and it will expand its tree based on the least cost of the active nodes, which is similar to Branch and Bound. The difference being BnB has a bound which makes it more efficient depending on how the bound is calculated. BnB is also similar to A* algorithm on how it expands based on the minimum cost, but A* doesn't search for other possibility once it reached a goal node.

Here is a more structural explanation on how branch and bound algorithm works.

1. Each node is given a cost c(i), where c(i) is a heuristic value of the cost to travel through node i to the goal node.
2. Expand the least cost among all the active node costs. Active nodes are the nodes that are connected to the expanding node.
3. If a goal node is found, eliminate all the active node with the cost bigger than the current goal node cost. If there is still an active node, continue expanding it.
4. Repeat step 3 until one goal node is found and there's no active node available left.

As mentioned before, BnB has a bound function which is used to eliminate nodes that seems unfitting for the best solution. It also differentiates BnB with another algorithm.

B. *Travelling Salesman Problem*

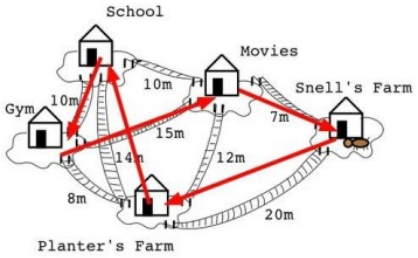

Fig. 2. Travelling Salesman Problem illustration (Source : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Brute-Force-(2021)-Bag1.pdf accessed May 8, 2021)

Travelling Salesman Problem is a mathematical problem which ask the question "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

This problem is commonly used as a benchmark for many optimization methods. There are many solutions to TSP, however not all solution algorithm is efficient enough to use. For example, TSP can be solved using exhaustive search algorithm which explore all the possible combination of solutions then comparing it with each other. This method guarantees the best solution, but the more city in the problem, the worse the algorithm gets. The complexity for exhaustive search to solve TSP is O(n!) for n cities. One of the efficient algorithms to solve TSP is branch and bound algorithm.

C. *TSP using Branch and Bound*

There are a number of methods on using branch and bound algorithm to solve TSP. In this paper, we will be using reduced cost matrix method. The matrix contains the cost to travel from one node to another. Here are the steps to solve it.

1. Form the cost matrix which represent the connection between each node and the cost to travel between each node. If two nodes aren't connected, then fill it with infinity (∞).
2. Reduce the matrix' row and column to form reduced matrix by subtracting each row and column with the least value. The matrix is reduced one every row and column has at least one zero. The sum of the subtrahend is the cost of the root node.
3. Given reduced matrix A is for node R and S is branch of node R. For each node connected to the starting node, do this to reduced matrix A.
    a. Change the value in row R and column S into infinity
    b. Change (S,1) into infinity.
    c. Reduced the matrix again.
    d. The minimum cost through node S is equal to

$$\hat{c}(S) = \hat{c}(R) + A(i,j) + r$$

    with r = sum of subtrahend
4. From the active nodes, choose one node which has the minimum cost and expand it by doing step 3 again.
5. If a goal node is reached, in this case when there's no more node to expand, eliminate all the other node that has a bigger cost than the goal node cost. If there is still an active node left, expand it. Repeat until there's no active node left except the solution.

D. *Tokyo*

Fig. 3. Tokyo City (Source : https://www.japan-guide.com/e/e2164.html accessed May 9, 2021)

Tokyo is the capital city of Japan and one of the most populous area in Japan, consisting of 23 central city wards. Tokyo offers a lot of choices of shopping, entertainment, culture, and dining to its visitors. The Shibuya district for example, has a lot of shopping and dining place, while the Asakusa district has museum, historic temple, and garden. Tokyo also has a great subway system which is a really convenient way to get around the city, both for locals and tourists.

III. Solving The Problem

A. *Picking the Locations*

As stated in the previous section, Tokyo is a large area so it's impossible to cover every location in this problem. With that, I chose six of its famous locations based on japan-guide.com, which consists of these places.

1. Asakusa, shortened to ASA
2. Shibuya, shortened to SHB
3. Shinjuku, shortened to SHJ
4. Ginza, shortened to GNZ
5. Akihabara, shortened to AKB
6. Ikebukuro, shortened to IKB

There are other famous places, but to simplify on solving the problem, I decided to keep it as six. The other reason for choosing these six places is because all six of them has a quite a large station. Because all six of these places are an area, not a specific location, I chose each of its station as their specific location. Reason being, it makes it easier to calculate the costs of travel to and from each of the locations.

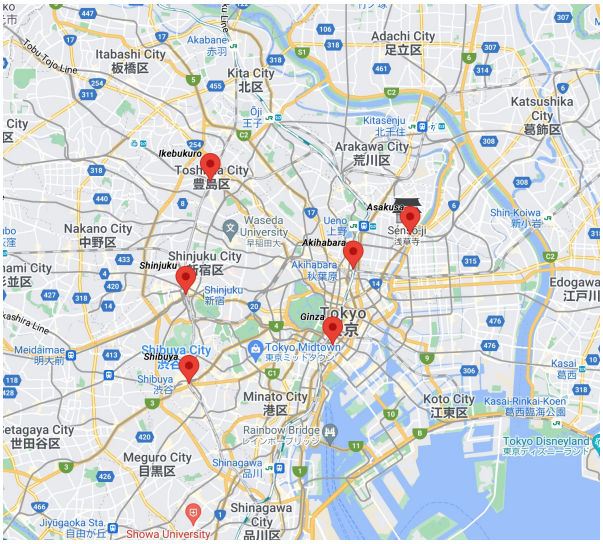

Fig. 4. Map of Tokyo with six chosen places (Source : Google Maps taken May 9, 2021)

B. *Gathering the Cost*

The are two types of cost we will be using. First, the cost of time and second, the cost of money. Because there are several ways to travel between each place. I chose one method of travel, which is by subway or train. The reason why I chose train is because It's one of the fastest ways to travel around the city and a really convenient one.

The railway map for Tokyo area is really complicated to look at, and so I made a simplified version which includes the six places that were mentioned.

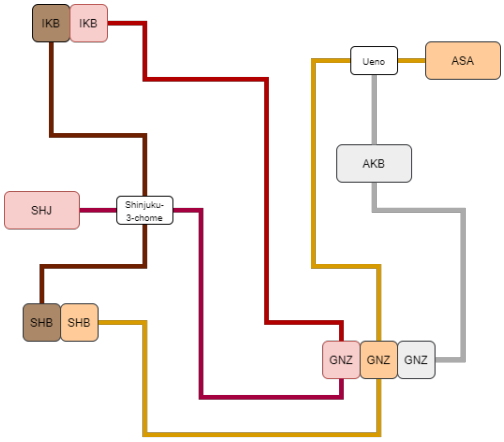

Fig. 5. A simplified tokyo railway map of six stations

Due to the limitation of taking the data myself, I used the information that I can get on the internet. For the time cost, I used Google maps estimation time to and from each station. The data isn't entirely accurate, but based on my experience it's still fairly accurate. There are several alternatives, so I used the least estimation time out of all the option on google maps, and here is the result.

	ASA	SHB	SHJ	GNZ	AKB	IKB
ASA	∞	33	25	18	13	27
SHB	33	∞	7	16	22	11
SHJ	25	7	∞	17	13	7
GNZ	18	16	17	∞	10	19
AKB	13	22	13	10	∞	20
IKB	27	11	7	19	20	∞

Fig. 6. Cost matrix to travel between six places in Tokyo based on the time it takes. Taken May 9, 2021. All value is in minutes.

Based on google estimation time, the time to and from one place to another is slightly different. To simplify this, I made the time to and from each place to be equal.

For the price cost, I will be using Tokyo Metro train fare. Each origin station has its own price to a certain destination. Usually train fare map is available on every station, but Tokyo Metro has a website to calculate train fare as well as the route it takes. Based on that website, I got the following data.

	ASA	SHB	SHJ	GNZ	AKB	IKB
ASA	∞	250	250	200	170	250
SHB	250	∞	170	200	200	200
SHJ	250	170	∞	200	200	170
GNZ	200	200	200	∞	170	200
AKB	170	200	200	170	∞	250
IKB	250	200	170	200	250	∞

Fig. 7. Cost matrix to between six places in Tokyo based on train fare. Taken May 9, 2021. All value is in yen.

All of these fares are adult ticket prices. For children, they are half priced which makes it the same scale. And of course, these prices don't apply for those who have the train pass. The train fares in Japan are fairly standardized, which explains the reason that train fares are pretty similar with each other.

The starting point of this travelling salesman problem will be Asakusa. The reason is Asakusa is the nearest location to Narita International Airport, which where most tourist arrived from.

C. *Solving Time-Based Cost*

Reducing the time-based cost matrix, we get the following matrix.

	ASA	SHB	SHJ	GNZ	AKB	IKB
ASA	∞	20	12	5	0	14
SHB	23	∞	0	9	15	4
SHJ	15	0	∞	10	6	0
GNZ	5	6	7	∞	0	9
AKB	0	12	3	0	∞	10
IKB	17	4	0	12	13	∞

Fig. 8. Reduced time-based cost matrix.

The cost of the root node is the sum of the subtrahend which equals to 57. Applying the branch and bound method explained in the second section of this paper, we get the following state space tree.

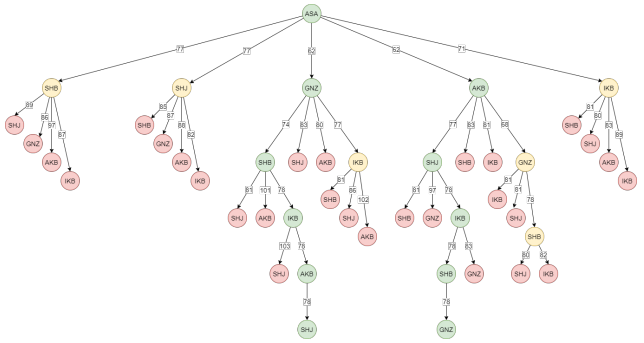

Fig. 9. State space tree for time-based cost

The algorithm starts at the first node, which is ASA. Then it will calculate the cost to travel to each node using the bound function. The cost of travel to each node from ASA are the following:

1. ASA → SHB: 77
2. ASA → SHJ: 77
3. ASA → GNZ: 62
4. ASA → AKB: 62
5. ASA → IKB: 71

Because GNZ and AKB has the least cost among five of them, and GNZ appears before AKB, it will expand on GNZ first. It will then calculate the cost of each unvisited nodes. Continue on after expanding GNZ, it will then expand on AKB, since it has the least cost. After that, it will expand on GNZ with 68 cost, then IKB with 71 cost, then all the nodes with 77 cost which none of them reach a goal node.

At this point there are two 78 cost node which is SHB and SHJ on level 2. Expanding the SHB node will then reach a goal node and provide us with a solution with a total cost of 78. Because a goal node has been reached, it will then eliminate all the nodes with a cost more than 78. The only node available left is the SHJ on level 2. Expanding this node will also reach a goal node with a total cost of 78, equal to the previous solution.

Now that the algorithm is done, there are two solutions to this problem. First route is Asakusa → Ginza → Shibuya → Ikebukuro → Akihabara → Shinjuku → Asakusa. The second route is Asakusa → Akihabara → Shinjuku → Ikebukuro → Shibuya → Ginza → Asakusa. Both of them has a total travel time of 78 minute.

D. Solving Price-Based Cost

Reducing the price-based cost matrix, we get the following matrix.

	ASA	SHB	SHJ	GNZ	AKB	IKB
ASA	∞	80	80	30	0	80
SHB	80	∞	0	30	30	30
SHJ	80	0	∞	30	30	0
GNZ	30	30	30	∞	0	30
AKB	0	30	30	0	∞	80
IKB	80	30	0	30	80	∞

Fig. 10. Reduced price-based cost matrix.

The cost of the root node is the sum of the subtrahend which equals to 1020. Applying the branch and bound method explained in the second section of this paper, we get the following state space tree.

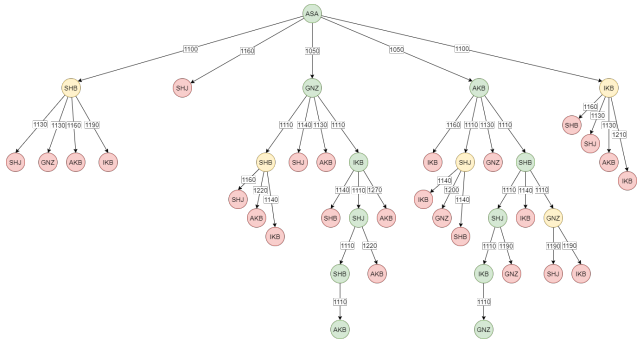

Fig. 11. State space tree for price-based cost

Same as the previous calculation, the algorithm starts at ASA which then expands to its adjacent nodes. The cost of travel to each node from ASA are the following:

1. ASA → SHB: 1100
2. ASA → SHJ: 1160
3. ASA → GNZ: 1050
4. ASA → AKB: 1050
5. ASA → IKB: 1100

Expands the least cost node which are GNZ and AKB. All the active nodes from the expansion cost more than 1100, which means expand on SHB and IKB on the first level. After that, we have 4 nodes with the cost of 1110 with two of those reaching a goal node provide us with the solution. After one of the branches reached a goal node, it will then eliminate all the nodes with the cost more than 1110. It then expands on the other 1110 node which reached another goal node.

The first solution is Asakusa → Ginza → Ikebukuro → Shinjuku → Shibuya → Akihabara → Asakusa. And the second solution is Asakusa → Akihabara → Shibuya → Shinjuku → Ikebukuro → Ginza → Asakusa. Both of them has a total ticket price of 1.110 yen.

E. *Checking the Solution with a Python Program*

All the calculations to find the solution in the previous section were done manually and are prone to calculation mistakes. To make sure that the solutions found in the previous section are correct, I will check it using a Python Program from Geeks for Geeks.

The program has a different bound function. Instead of using reduced cost matrix to determine the cost to travel, they use the complete tour cost bound function. In short, it calculates the sum of two minimum edges cost of all the nodes. This shouldn't be a problem as the result should be the identical to each other if the calculations were correct.

Given the following index

1. 0 : Asakusa
2. 1 : Shibuya
3. 2 : Shinjuku
4. 3 : Ginza
5. 4 : Akihabara
6. 5 : Ikebukuro

Inputting the time-based cost table into the program, we were given the following result.

```
Minimum cost : 78
Path Taken :  0 3 1 5 2 4 0
```

The result from the previous section matches with the one from the python program. Asakusa → Ginza → Shibuya → Ikebukuro → Akihabara → Shinjuku → Asakusa.

Inputting the price-cost matrix into the program, we get the following result.

```
Minimum cost : 1110
Path Taken :  0 3 5 2 1 4 0
```

This solution also matches with out first calculated solution. Asakusa → Ginza → Ikebukuro → Shinjuku → Shibuya → Akihabara → Asakusa.

The problem with this program is that it only shows one solution to the program even if there are more than one solution. To check the other solution, we can change the order of the matrix by swapping Ginza with Akihabara, which makes the index the following.

0 : Asakusa
1 : Shibuya
2 : Shinjuku
3 : Akihabara
4 : Ginza
5 : Ikebukuro

Inputting the time-based cost table into the program, we were given the following result.

```
Minimum cost : 78
Path Taken :  0 3 2 5 1 4 0
```

The second solution also matched with the program's solution as index 3 is now Akihabara and index 4 is Ginza. Asakusa → Akihabara → Shinjuku → Ikebukuro → Shibuya → Ginza → Asakusa.

Inputting the price-based cost table into the program, we were given the following result.

```
Minimum cost : 1110
Path Taken :  0 3 1 2 5 4 0
```

Which matches with the second solution Asakusa → Akihabara → Shibuya → Shinjuku → Ikebukuro → Ginza → Asakusa. All four of them also resulted in equal cost, which is 78 minutes for the time-based cost and 1110 yen for the price-based cost.

IV. Result and Conclusion

A. *Result*

Based on the previous calculation we have 4 solutions, two for each cost type to travel around the six selected places.

Time-based cost

First solution: Asakusa → Ginza → Shibuya → Ikebukuro → Akihabara → Shinjuku → Asakusa

Second solution: Asakusa → Akihabara → Shinjuku → Ikebukuro → Shibuya → Ginza → Asakusa

Price-based cost

First solution: Asakusa → Ginza → Ikebukuro → Shinjuku → Shibuya → Akihabara → Asakusa

Second solution: Asakusa → Akihabara → Shibuya → Shinjuku → Ikebukuro → Ginza → Asakusa

The solutions found here is a separate solution of the two cost. To find the best solution for both time and price cost, we would need to combine both cost into a single cost matrix. Adding them both directly won't give the correct data since both have a different scale, so finding it would need further calculation.

B. *Conclusion*

Branch and bound algorithm is a combination of other pathfinding algorithm, such as Breadth First Search algorithm and least cost search. It can also be compared to exhaustive search with some improvement to prevent it from searching the whole branch. The worst-case complexity for branch and bound algorithm is exponential ($O(n^2)$), however the average complexity is significantly lower.

Using a branch and bound algorithm manually isn't the best option since you need to do a calculation for every node. Especially in this travelling salesman problem using the reduced cost matrix bound function, where making the matrix itself takes quite a bit of time. Additionally, the worst-case scenario is exploring the whole state space tree.

Nevertheless, branch and bound algorithm is good for a program since it's a repeated calculation for every node in the state space tree.

CALCULATION PROOF

As mentioned before, the calculations done on this paper are done manually. Here is the link to the spreadsheet I used to save my calculation results.

https://drive.google.com/file/d/1B2nyo7vI4NbnKT7mDiaLTkO_uoJTo3TB/view?usp=sharing

The data in the spreadsheet is stored similar to a horizontal tree. The tree starts on the left side of the spreadsheet then expands to the right. It is also color-coded for the node cost.

ACKNOWLEDGMENT

I would like to express my gratitude towards Allah for all the grace and blessings. I would also like to give my thanks to Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph.D. and Dr. Ir. Rinaldi, M.T. the lecturers of Algorithm Strategy class at ITB for all the knowledge. Moreover, I'm grateful for the support given by my friends and family.

REFERENCES

[1] R. Munir, "Algoritma Branch and Bound", 2021, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian2.pdf

[2] GeeksforGeeks Team, "Travelling Salesmen Problem using Branch and Bound", taken from https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/, accessed May 10, 2021

[3] Japan-guide Team, "Tokyo", taken from https://www.japan-guide.com/e/e2164.html, accessed May 9, 2021

[4] N. Thakoor, V. Devarajan and J. Gao, "Computation complexity of branch-and-bound model selection," 2009 IEEE 12th International Conference on Computer Vision, 2009.

STATEMENT

I hereby declare that the paper I have written is my own writing, not an adaptation or translation of someone else's paper, and not plagiarized.

Bandung, 11th May 2021

Muhammad Iqbal Sigid - 13519152