International Baccalaureate

Mathematics: Analysis and Approaches Internal Assessment

Higher Level

Employing Travelling Salesman Problem and Floyd-Warshall Algorithm to minimise distance of a delivery route

Number of pages (excl. Bibliography and Appendix): **20**

Introduction

I have always loved business and entrepreneurship. I fondly remember helping my maternal aunt launch her enterprise (Anika Fashions) during the Covid-19 pandemic. She specialises in sourcing unique cultural garments and accessories from different regions in India. Customers request specific items via WhatsApp, and she fulfils orders four times a week through self-delivery using her car.

Initially, operations were centred around her residence in Ganesh Nagar (a city in Hyderabad, India), reducing delivery costs (i.e., less spending on petrol). With time, her business has expanded, and her orders increased, however the customers were located further away. Therefore, this increased her **travel distance**, consequently increasing travel time (currently is **up to 3 hours**), and petrol costs. Hence, she is considering relocating closer to her new customer base. However, relocation is a big change and can introduce additional costs which can hinders her business' growth.

General Aim

Therefore, my aim is to optimise her current delivery route by **minimising travel distance**, such that travel time takes **less than 3 hours**.

Methodology

Before I can minimise her travel distance, I firstly need to understand her business model (Figure 1). From Figure 1, it is possible for my aunt's business to be modelled in terms of the **Travelling Salesman Problem (TSP)**. However, for this, an important assumption needs to be made - the shortest distance taken to deliver goods to her customers' places would be a Hamiltonian Circuit.

	Aunt's Business Objectives
1	She starts at her house.
2	She travels to her customers' places to sell/ deliver her garments (in the shortest distance).
3	She returns back to her apartment.

	Travelling Salesman Problem (TSP)
1	A salesman starts from an initial location.
2	The salesman travels to n cities to sell his goods, in a Hamiltonian Circuit.
3	The salesman returns back to his initial location.

Figure 1: Modelling Business in terms of TSP

Understanding TSP and Hamiltonian Circuits

The TSP is a classic optimisation problem, formulated in the context of Graph Theory. The problem involves a salesman, who objective is to sell his goods to $\boldsymbol{n}$ cities (vertices) [$n \in$ N], via the shortest distance Hamiltonian circuit.

Hamiltonian Circuits

From Figure 2, each city is denoted as a vertex, and the path connecting two cities or vertices, is an edge. Edges can also represent the distance between connected vertices. Essentially, **Hamiltonian Circuits** thereby refers to:

1. A route where the starting and ending vertex is the same.
2. A route where a non-starting vertex is visited only once, while the starting vertex is visited twice; once at beginning and once at end]

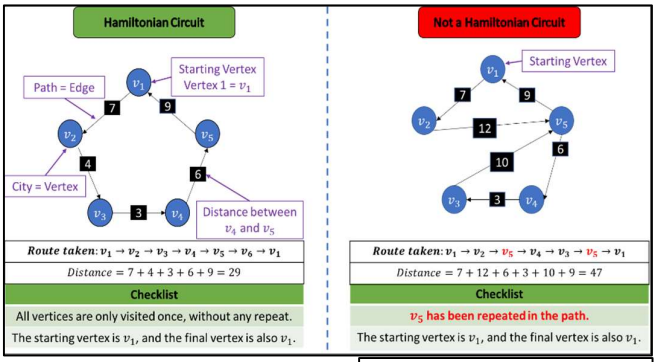


Figure 2: Hamiltonian Circuits

Therefore, solving TSP involves determining the shortest distance Hamiltonian circuit of *n* vertices.

Modified Aim

Hence, after modelling my aunt's business in terms of TSP, the primary aim has been modified to: ***determining the shortest distance Hamiltonian Circuit my aunt takes to visit her customers' places, given her place of residence is the starting vertex, such that travel time takes less than 3 hours.***

Solving TSP

In general, solving a problem encompasses 2 stages: determining a solution (1st), and verifying the identified solution (2nd). Therefore, in the context of TSP, this involves:

Stage 1: *Determining a shortest distance HC, while ensuring it satisfies the criteria of being a HC (Figure 2).*

Stage 2: *Verifying if the identified shortest distance HC is indeed the shortest.*

In **Stage 2,** the distance of the identified HC from **Stage 1** is compared with that of a pool comparing other potential HCs. However, to increase the accuracy of the identified HC's, it is essential for this pool of alternatives to expand as well (i.e., comparing with even more potential HCs). Without this expansion, there remains a chance that a HC not included within the pool could be the true shortest distance HC, rendering the identified HC from **Stage 1** inaccurate. Thus, to obtain the most accurate shortest distance HC, the **Exhaustive Search Algorithm** is used, because it exhaustively compares the distance of the identified HC, against the distances of **all the possible HCs** (*Search Algorithms*, n.d.). However, as the number of cities or vertices (***n***) increase, the total number of possible HCs increases **factorially** (grows at a rate described by a factorial function), proven in Table 1 below.

Exhaustive Search Algorithm and Total Number of HCs

To determine the total number of HCs formable with ***n*** vertices, some assumptions need to be made, according to those of the TSP (Gilbert & Halim, n.d.):

1. There is an edge connecting every pair of vertices.
2. The edges connecting the vertices are bi-directional. This means that one can travel either direction, from vertex 1 $(v_1) \rightarrow v_n$, or from $v_n \rightarrow v_1$. (Figure 3)

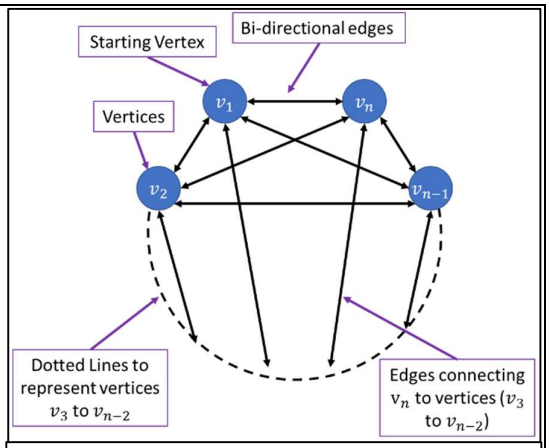


*Figure 3: Complete Unweighted Graph of **n** vertices*

Let Figure 3 be an unweighted graph visualizing ***n*** number of vertices, and edges connecting these ***n*** vertices.

In Graph Theory (study of vertices and edges) (Joshi, 2017), a **graph** is a mathematical structure that illustrates the relationships between vertices and edges. And this graph is **unweighted** because the edges have no value associated with it. For instance, the graphs represented in Figure 2 are **weighted graphs** because its edges represent the distance between its vertices, but here, these edges merely illustrate the presence of a path connecting each pair of vertices.

With reference to Figure 3, let us consider a scenario with ***n*** vertices. Based on the first assumption, there is an edge connecting every pair of vertices. Hence, this means that each vertex is connected to the remaining $(n-1)$ vertices, through $(n-1)$ edges.

Let v_1 be the starting vertex. v_1 is connected to the remaining $(n-1)$ vertices. Therefore, starting from v_1, there are a total of ${}^{n-1}C_1 = (n-1)$ combinations of paths that can be formed. These paths can include

- v_1 to v_2
- v_1 to v_3
- v_1 to v_4 and so on.

Let us consider the construction of the path 'v_1 to v_2'. Similar to v_1, v_2 is also connected to the remaining $(n-1)$ vertices. However, v_2 cannot return back to v_1, because then, the starting vertex would have been returned back to, closing the circuit. And this is not feasible because all the vertices have not been visited, thereby not fulfilling the condition of a HC. Therefore, there can only be a total of ${}^{n-2}C_1 = (n-2)$ combinations of paths that can be formed, continuing from v_2.

Let us consider the further construction of the path '$v_1 \rightarrow v_2 \rightarrow v_3$'. Similarly, v_3 cannot travel back to v_1, because it closes the circuit, and cannot travel back to v_2 because a non-starting vertex would then be visited twice; violating the conditions of a HC. Therefore, from v_3, there can only be a total of ${}^{n-3}C_1 = (n-3)$ combinations of paths that can be formed.

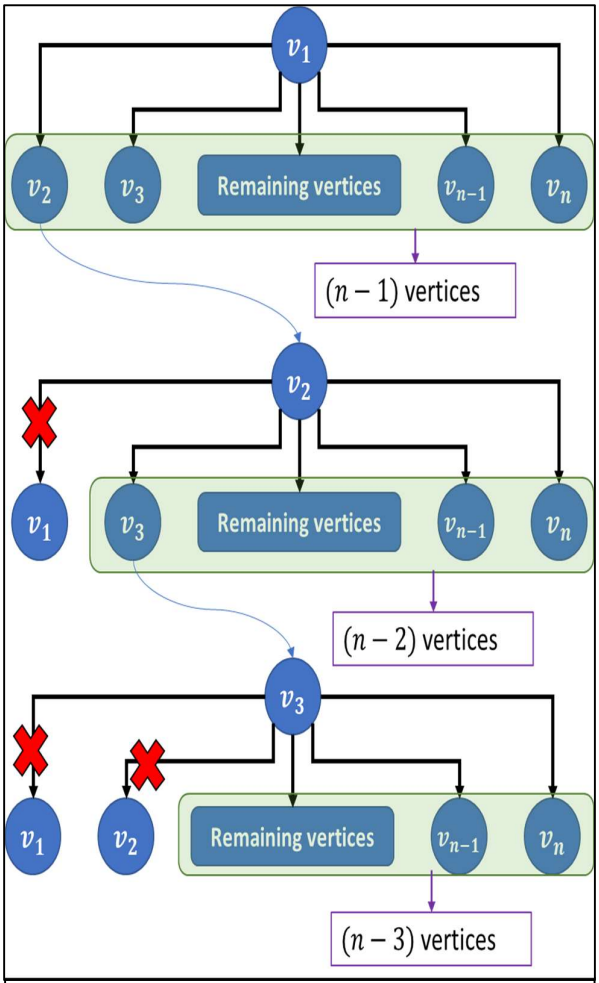


Figure 4: v_1 to v_3 from Figure 3

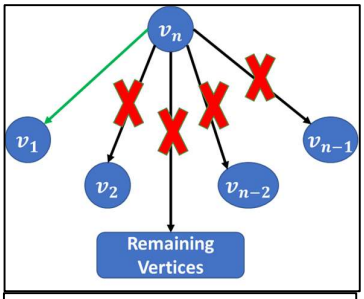


Figure 5: v_n from Figure 3

Let us assume this path continues from '$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v_n$', where v_n is the last non-starting vertex being visited. Upon reaching v_n, all the non-starting vertices would have already been visited (exactly once), therefore the objective is to close the HC by returning back to the starting vertex (v_1). Therefore, from v_n, only ${}^{1}C_1 = 1$ combination of path can be formed, which is '$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots v_n \rightarrow v_1$'. The return to the starting vertex marks the completion and formation of a HC.

From this example, it is evident that numerous combinations of paths could have been formed. For instance, from v_1, instead of v_2, the path could have been $v_1 \rightarrow v_3$ or $v_1 \rightarrow v_4$.

Mathematically, this means that there is a total of

$$^{n-1}C_1 \times {}^{n-2}C_1 \times {}^{n-3}C_1 \times \ldots \times {}^{1}C_1 = (n-1) \times (n-2) \times (n-3) \times \ldots \times 1 = \boldsymbol{(n-1)!}$$

HCs that could have been formed, in a scenario involving $\boldsymbol{n}$ vertices. But this is only for $n \geq 2$, because a HC can only be formed with 2 or more vertices.

However, each HC forms a closed loop, and given the second assumption on Page 3 (all edges connecting the vertices are **bi-directional**), this means that each HC can have both clockwise and anti-clockwise arrangements (Figure 6), and are thus mirror images of one another. Therefore, these HCs would be having the same distance and thus are not distinct in terms of distance travelled. Therefore, to avoid double-counting, these $(n-1)!$ HCs are divided by 2, to provide **distinct HCs** (i.e., distinct HCs with distinct distances).

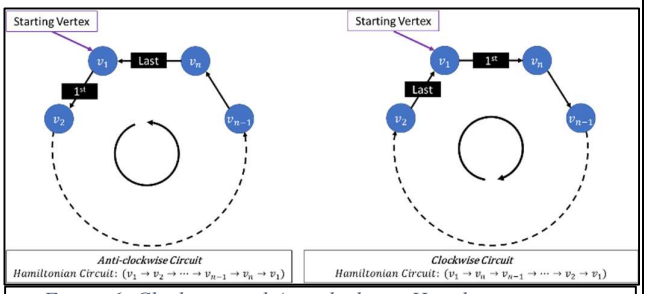


Figure 6: Clockwise and Anti-clockwise Hamiltonian circuits

$$\therefore \boldsymbol{Distinct\ No.\ of\ HCs\ formable\ with\ n\ vertices} = \frac{\boldsymbol{(n-1)!}}{\boldsymbol{2}}, \boldsymbol{n \geq 2}$$

Therefore, as $\boldsymbol{n}$ increases, no of HCs increases factorially.

Table 1: Formula for Total Distinct Number of HCs of $\boldsymbol{n}$ vertices

This factorial increase makes it impractical to exhaustively search **every permutation of vertex** to determine the shortest distance HC, because as observed in Figure 7, as $\boldsymbol{n}$ increases, number of distinct HCs increase at an increasing rate. This thereby highlights the un-scalability of the Exhaustive Search Algorithm (ESA).

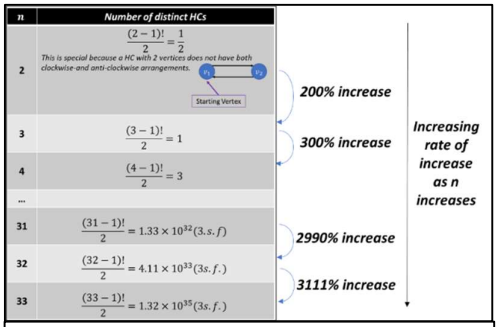


Figure 7: Factorial Increase in No. of HCs as n increases

Within the fiscal year 2022-23 (1st April 2022 to 31st March 2023), my aunt had **32** customers, each living at different locations. Including her place of residence, there are thus a total of **33** vertices to consider. From Table 1, this means there are a total of $\frac{(33-1)!}{2} = \boldsymbol{1.32 \times 10^{35}} (3\ s.f.)$ HCs to consider. Clearly, conducting an exhaustive search from these 1.32×10^{35} HCs, to determine the shortest distance one, is impractical. Therefore, I have created two solutions to reduce the number of HCs being formed and considered.

Solution 1 – Reducing number of vertices

The total number of distinct HCs that can be formed with $\boldsymbol{n}$ vertices $= \frac{(n-1)!}{2}$. But, reducing $\boldsymbol{n}$ also factorially decreases the distinct number of HCs formed, and with less HCs to consider, this thereby simplifies the task of identifying the shortest distance HC, making it an effective solution.

Therefore, in this exploration, I will be reducing the number of vertices present for analysis, under **Chapter 1: Streamlining and Identifying Important Vertices**.

Solution 2 – Using Heuristic Algorithms instead of Exhaustive Search Algorithm

By employing ESA for **33** vertices, a total of 1.32×10^{35} number of HCs are being formed and considered. However, employing heuristic algorithms like the **Nearest Neighbour Algorithm (NNA)** significantly decreases the number of HCs being formed (Gutin & Punnen, n.d.), proven below under Table 2,thereby simplifying the process of verifying the shortest distance HC.

Nearest Neighbour Algorithm (NNA) (Lecture9.Pdf, n.d.)

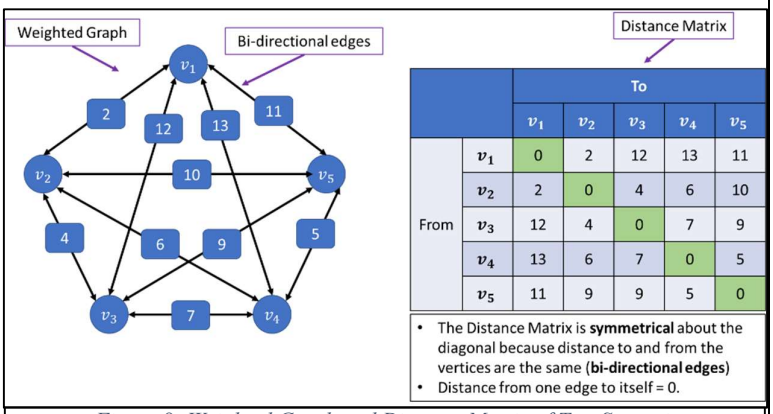
The NNA is a simple heuristic algorithm for constructing a HC. It is advantageous because of its minimal computational complexity, and can be simply executed by both hand calculations and code.

The process starts by selecting a random vertex, and subsequently choosing the nearest unvisited vertex (in terms of distance) at each step until all the vertices have been visited, ultimately returning to the initial vertex. This process repeats iteratively until all the vertices have been considered as an initial vertex, therefore creating ***n*** vertices.

However, a pre-requisite for the implementation of the NNA is the requirement of a **distance matrix;** a table which represents the distances between every pair of vertices. Therefore, one can consider a distance matrix as a repository of the distances along the edges connecting every pair of vertices (*Distance Matrix - an Overview | ScienceDirect Topics*, n.d.).

Test Scenario of NNA

To prove that NNA yields less HCs than the Exhaustive Search Algorithm, let us consider a test scenario with 5 vertices, visually depicted by the weighted graph and distance matrix in Figure 8. In order to compare NNA with the Exhaustive Search Algorithm, assumptions as listed on Page 3, need to be carried over:



From \ To	v_1	v_2	v_3	v_4	v_5
v_1	0	2	12	13	11
v_2	2	0	4	6	10
v_3	12	4	0	7	9
v_4	13	6	7	0	5
v_5	11	9	9	5	0

- The Distance Matrix is **symmetrical** about the diagonal because distance to and from the vertices are the same (**bi-directional edges**)
- Distance from one edge to itself = 0.

Figure 8: Weighted Graph and Distance Matrix of Test Scenario

1. There is an edge connecting every pair of vertices.
2. The edges connecting the vertices are bi-directional.

Using NNA

Let V be the set of vertices: $V \in \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$.

Let U be the set of unvisited vertices, such that $U \subseteq V$. Since none of the vertices are visited at the beginning of the algorithm, $U \in \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$, however, it reduces as more vertices are being visited.

Step 1: Choosing an initial vertex

NNA starts by selecting an initial vertex from U. Let this vertex $= v_1$. Let P_1 denote the HC being constructed, with v_1 as its initial vertex. Since v_1 is visited, v_1 is removed from the set U $[U \backslash \{v_1\}]$. Thus, $U \in \{v_2, v_3, v_4, v_5\}$.

Step 2: Choosing the nearest vertices

	v_1	v_2	v_3	v_4	v_5
v_1	**0** ✗	**2**	**12**	**13**	**11**
v_2	**2** ✓	0	4	6	10
v_3	**12**	4	0	7	9
v_4	**13**	6	7	0	5
v_5	**11**	9	9	5	0

Figure 9: v_1 in Distance Matrix

Consequently, from v_1, P_1 extends by choosing the closest vertex to v_1, from U. This can be done so, by referring to the distance matrix, because the closest vertex to v_1 would correspondingly refer to the edge connecting that vertex to v_1. Therefore, referring to Figure 9, we need to choose the edge with the minimum value which is non-zero, otherwise that would signify v_1 travelling back to itself. As observed, the smallest value is 2, corresponding to the edge connecting v_1 and v_2. Therefore, P_1 extends itself by travelling from v_1 to v_2. $[v_1 \rightarrow v_2]$

Since v_2 has now been visited, $U\backslash\{v_2\}$ because it is no longer an unvisited vertex. Therefore, now, $U \in \{v_3, v_4, v_5\}$. P_1 once again extends itself by choosing a vertex from U, which is closest to v_2 (in terms of distance). As observed in Figure 10, this would be v_3. Thus, P_1 is now $[v_1 \rightarrow v_2 \rightarrow v_3]$. After which, v_3 would be visited, resulting in $U\backslash\{v_3\}$ and $U \in \{v_4, v_5\}$.

	v_1	v_2	v_3	v_4	v_5
v_1	0	**2** ✗	12	13	11
v_2	**2** ✗	**0** ✗	**4** ✓	**6**	**10**
v_3	12	**4** ✓	0	7	9
v_4	13	**6**	7	0	5
v_5	11	**9**	9	5	0

Figure 10: v_3 in Distance Matrix

Step 2 repeats itself till there are no more elements (i.e., vertices) in U; $U \in \{\}$.

At this point, P_1 would be $[v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5]$. Upon reaching v_5, all non-starting vertices have been visited (exactly once), therefore the objective is to close the HC by returning back to the starting vertex v_1. Thus, P_1 is $[v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_1]$, as illustrated in Figure 11.

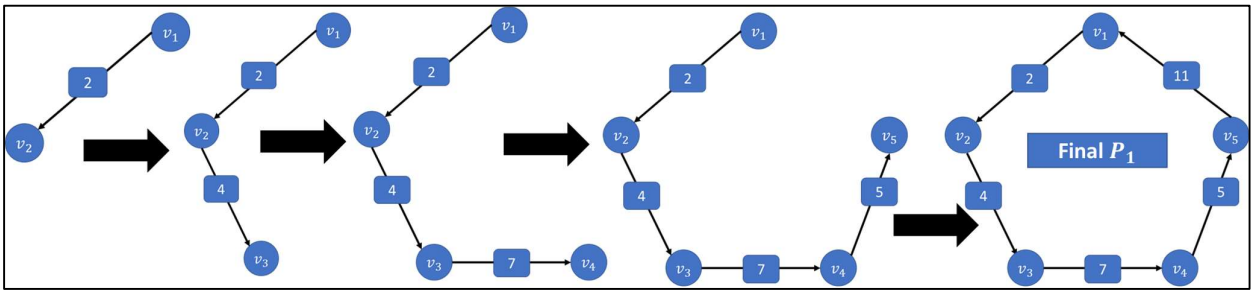


Figure 11: Construction of P_1

Now, NNA repeats iteratively until all the vertices have been selected as the initial vertex, meaning the formation of P_2, P_3, P_4, P_5, illustrated in Figure 12. Hence, a total of 5 HCs is constructed.

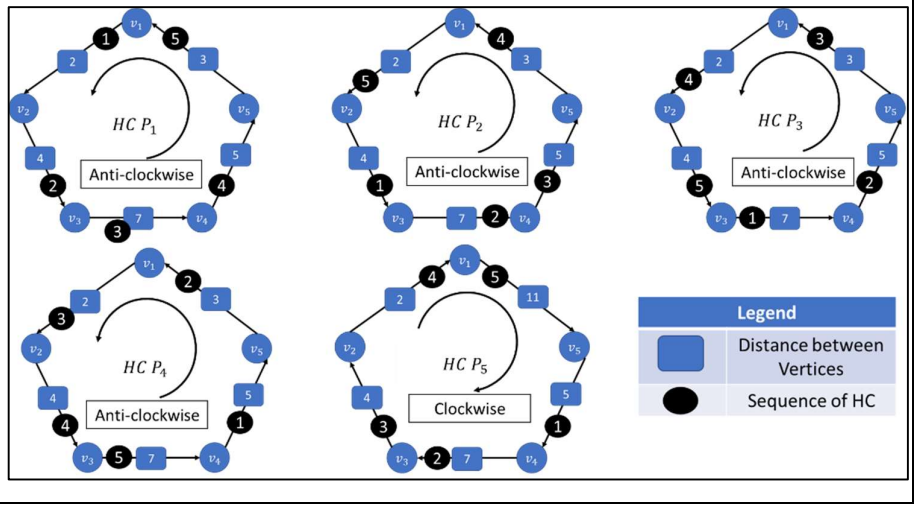


Figure 12: Hamiltonian Circuits Formed through NNA for Test Scenario ➔

However, all P_1 to P_5 are the same HC, but are in either clockwise or anti-clockwise orientations. Therefore, there is only **1** distinct HC provided from NNA. Hence, it is observed that the number of distinct HCs formed by NNA can be $\leq n$, where in this scenario, it is **1**.

Using Exhaustive Search Algorithm

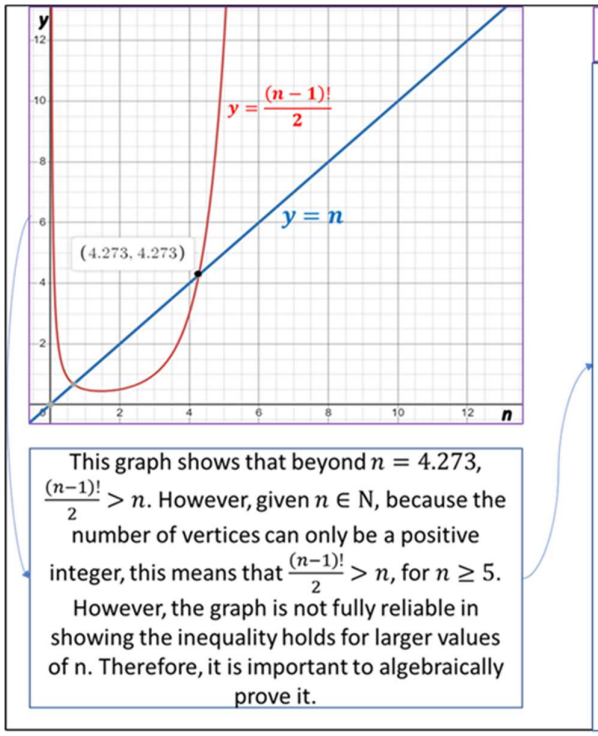
According to the formula derived in Table 1, the total number of distinct HCs that can be formed $= \frac{(5-1)!}{2} = \mathbf{12}$.

$[1 < 12]$ Hence, NNA generated less distinct HCs in the test scenario.

Table 2: The Nearest Neighbour Algorithm (NNA)

Therefore, from Table 2, it is observed that the NNA can generate and evaluate fewer distinct HCs. Without more test scenarios, it is unable to generalise the distinct number of HCs NNA can generate, but it is possible to consider a range of $\leq n$ (not exceeding n). This is because NNA only generates n HCs, where each HC has a distinct starting vertex, as shown in Table 2.

Nevertheless, HCs generated by NNA is less than that of Exhaustive Search Algorithm ($n_{HC} = \frac{(n-1)!}{2}$), which considers all possible permutations of HCs. However, this proposition is only true for $n \in \mathrm{N}, n > 4$, as per Figure 13.



To Show $\frac{(n-1)!}{2} > n$, for $n \geq 5$ and $n \in \mathrm{N}$

Let P_n be the proposition that $\frac{(n-1)!}{2} > n$, for $n \geq 5$.

Consider $n = 5$.

$LHS = \frac{(5-1)!}{2} = 12$	$RHS = 5$
$\therefore LHS > RHS$	

Hence we can assume P_n holds true for $n = k$, where $k \geq 5$ and $k \in \mathrm{N}$.

Assume $\frac{(k-1)!}{2} > k$

Consider $n = k + 1$

$$LHS = \frac{((k+1)-1)!}{2} = \frac{k!}{2} = \frac{k(k-1)!}{2}$$

As assumed earlier, $\frac{(k-1)!}{2} > k$. Additionally, given $k \geq 5$,

$$\Rightarrow k \times \frac{(k-1)!}{2} \geq 5 \times \frac{(k-1)!}{2} > 5k$$

$$RHS = k + 1$$

For $k \geq 5, 5k \geq 20$ and $k + 1 \geq 6$

$$\Rightarrow 5k > k + 1$$

$$\Rightarrow k \times \frac{(k-1)!}{2} \geq 5 \times \frac{(k-1)!}{2} > 5k > k + 1$$

$$\therefore k \times \frac{(k-1)!}{2} > k + 1$$

Since the proposition holds true for $n = 1$ and $n = k$ is assumed to be true, $\Rightarrow n = k + 1$ is true. Therefore, the proposition is true for all $n \in \mathrm{N}, \mathrm{n} \geq 5$

Figure 13: Proof of $\frac{(n-1)!}{2} > n$, where $n \geq 5, n \in N$

Although NNA simplifies determining the shortest distance HC, it comes with a limitation. Since NNA does not generate all possible HCs formable, there is a possibility that out of the one it generates, the true shortest distance HC does not exist. Therefore, **the shortest distance HC identified by NNA may not always be the true shortest distance HC**. Therefore, NNA may yield a less accurate final solution compared to ESA, as illustrated in the Venn Diagram (Figure 14).

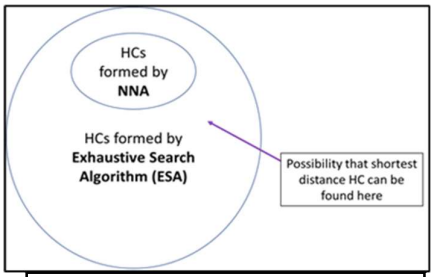


Figure 14: Venn Diagram of HCs formed

Nevertheless, in this exploration, I will:

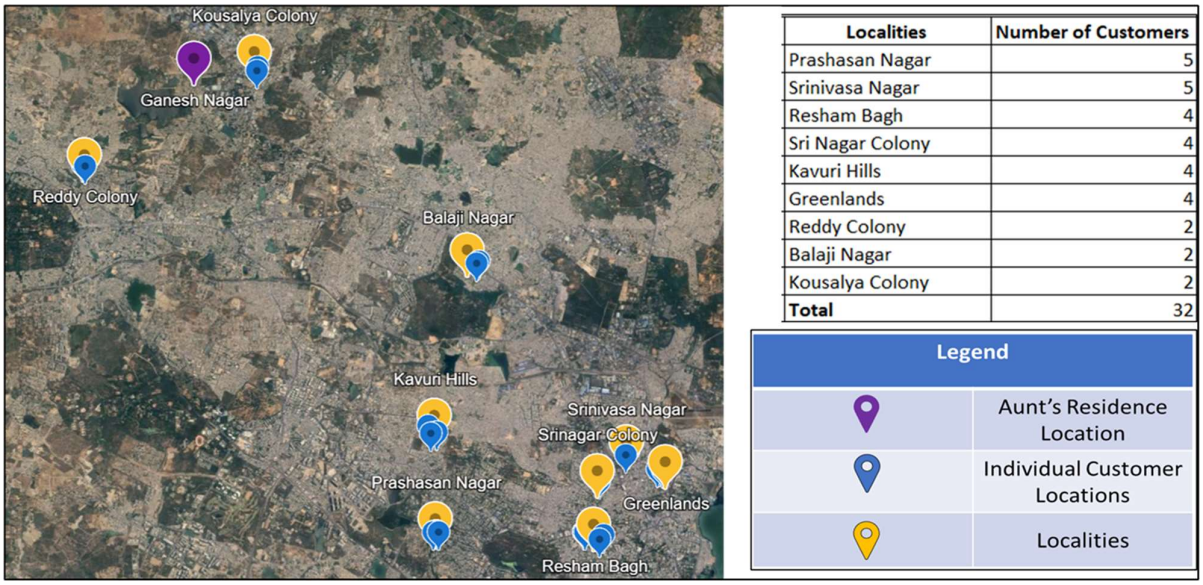
- Within **Chapter 1,** reduce the number of vertices for analysis,
  - This reduces the complexity of determining the shortest HC
- Within **Chapter 2**, create a corresponding distance matrix with vertices identified from Chapter 1
  - This is a pre-requisite for the employment of NNA.
- Within **Chapter 3**, determine the shortest HC using NNA

Chapter 1: Streamlining and Identifying Important Vertices

As aforementioned in Page 5, within the Fiscal year 2022-23, my aunt had a total of **32** customers. Including her place of residence, and **assuming she travels to all her customers' places every time she makes a delivery visit**, this however would generate around $\mathbf{1.32 \times 10^{35}}$ distinct HCs, and determining the shortest one from these is simply impractical. Therefore, by streamlining the number of vertices, this would reduce the number of HCs being formed and considered, thus simplifying the process of identifying the shortest distance HC.

Grouping Customers based on Localities

To start, I will plot all the customers' locations on Google Maps. This way, I can get a clear visual understanding of the customers' distribution (Figure 15).



Localities	Number of Customers
Prashasan Nagar	5
Srinivasa Nagar	5
Resham Bagh	4
Sri Nagar Colony	4
Kavuri Hills	4
Greenlands	4
Reddy Colony	2
Balaji Nagar	2
Kousalya Colony	2
Total	32

Legend	
	Aunt's Residence Location
	Individual Customer Locations
	Localities

Figure 15: Visual Representation of Customer Locations

Upon plotting the customers' locations, I realised that there were clusters, which upon researching, were the different localities within the city my aunt was living in, which is Hyderabad in India. Hence, I grouped the customers' locations, based on their respective localities (represented by the yellow markers in Figure 15). Therefore, the number of vertices has decreased **from 33 to 10**, including my aunt's residence.

Selecting only Frequent Customers

However, I believe that I can reduce this number further, by considering only my aunt's frequent customers. However, this adjustment challenges the initial assumption that my aunt visits all her customers on a busy day. But focusing on her most frequent customers provides a more realistic perspective, because these customers, having made more purchases from my aunt's enterprise, would require more visits from her, thus on a delivery day, it is highly likely for these customers' places to be on her delivery route to be there.

Hence, given these customers are more significant to my aunt's enterprise, they should be analysed and considered for the TSP. Thus, I created a bar graph of the average monthly orders my aunt receives from each of the localities (Figure 16).

Figure 16: Bar Graph of Average Monthly Orders from Different Localities

As observed in Figure 16, localities have been arranged in descending order for easy comparison, and as such, I was able to notice a significant decrease in average monthly orders between the localities Greenlands and Kousalya Colony. Therefore, the localities Kousalya Colony, Reddy Nagar and Balaji Nagar **will not be considered** for analysis as they are less significant to my aunt's enterprise.

Therefore, the list of important localities are as follows in Table 3. Each locality is associated with a specific vertex, for easier identification. Additionally, I decided to create an unweighted graph (Figure 17). This visual representation will help in identifying respective edges between vertices, and simplify the process of creating a distance matrix in **Chapter 2**.

Vertex	Locality
v_0	Ganesh Nagar (Aunt's Residence)
v_1	Kavuri Hills
v_2	Prashasan Nagar
v_3	Resham Bagh
v_4	Sri Nagar Colony
v_5	Greenlands
v_6	Srinivasa Nagar

Table 3: Table of Important Localities and Respective Vertices

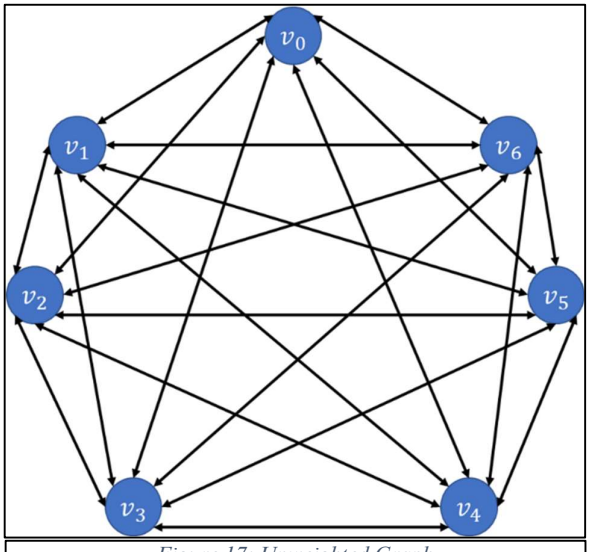


Figure 17: Unweighted Graph

Chapter 2: Creating a Distance Matrix

Now, we need to create a distance matrix to facilitate the employment of NNA to determine shortest distance HC. Information regarding distance between vertices is obtained from Google Maps since it is credible ((*How We Kept Information on Maps Reliable in 2021*, 2022).

My aim involves minimising travel distance to minimise travel time. This means that I could have also created a time matrix, and use NNA to find a HC which can be covered in the shortest time. However, during data collection, I encountered numerous challenges accounting for variables like accidents, traffic jams and weather, which can be highly volatile, when creating a time matrix. Hence, to simplify, I have opted to create a distance matrix, as shown in Table 4, and modify my approach such that the shortest distance HC would be determined.

Distance will be measured in **kilometres** because the minimum distance between 2 vertices is greater than 1.0km (at 1.9km). Additionally, they will be represented in **1 decimal place**, since Google Maps presents the data that way. Consequently, to find total travel time, I would divide the total distance by the average vehicular speed in Hyderabad (the city this exploration is conducted in). According to (Deshpande, 2023), this is 25km/h.

Vertices	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	15.8	19.0	21.2	19.2	20.0	17.7
v_1	15.8	0	3.0	6.1	6.5	9.2	6.9
v_2	19.0	3.0	0	5.8	6.6	9.8	9.0
v_3	21.2	6.1	5.8	0	1.8	4.0	4.8
v_4	19.2	6.5	6.6	1.8	0	3.0	2.2
v_5	20.0	9.2	9.8	4.0	3.0	0	1.9
v_6	17.7	6.9	9.0	4.8	2.2	1.9	0

Table 4: Complete Distance Matrix of vertices v_0 to v_6

During data collection, I realised something odd. The data from *Google Maps* presents how when travelling from v_0 to v_2, it is actually shorter to take the path $[v_0 \rightarrow v_1 \rightarrow v_2]$, than the direct path $[v_0 \rightarrow v_2]$ (through the edge).

Let $D(v_i v_k v_j)$ represent the distance of the path from v_i to v_k, then from v_k to v_j [$\therefore D(v_i v_k v_j) = D(v_i v_k) + D(v_k v_j)$].

$$\therefore D(v_0 v_1 v_2) = D(v_0 v_1) + D(v_1 v_2) = 15.8 + 3.0 = 18.8$$

$$D(v_0 v_2) = 19.0 > D(v_0 v_1 v_2)$$

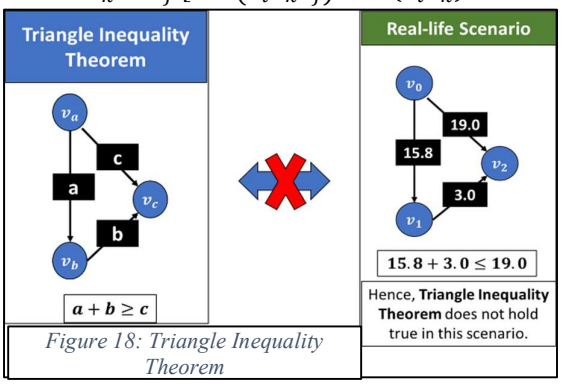


Figure 18: Triangle Inequality Theorem

The reason I found this phenomenon odd was because it goes against the ***Triangle Inequality Theorem***, which states that within a triangle, the sum of any two sides of a triangle is greater than or equal to the third side (Safra, n.d.) (Figure 18). This is because of how simplified TSP is, that it cannot accurately model real-life scenarios. In real-life, the roads connecting v_0, v_1 and v_2 will not be a straight line (it might be slanted at some regions), and thus, they cannot be considered a triangle. Therefore, the ***Triangle Inequality Theorem*** does not apply.

In this scenario, it is best to remove the direct edge connecting v_0 and v_2. This is because, currently, there exists a shorter connected path connecting these 2 vertices; $[v_0 \rightarrow v_1 \rightarrow v_2]$. This adjustment would be **beneficial**, because removing this edge forces the NNA to prioritise the path $[v_0 \rightarrow v_1 \rightarrow v_2]$ over $[v_0 \rightarrow v_2]$, when constructing a HC (Figure 19). This alteration thereby increases the likelihood of constructing a HC with a lower total distance, because traversing the path $[v_0 \rightarrow v_1 \rightarrow v_2]$ covers more vertices at a shorter distance compared to the direct connection between $[v_0 \rightarrow v_2]$.

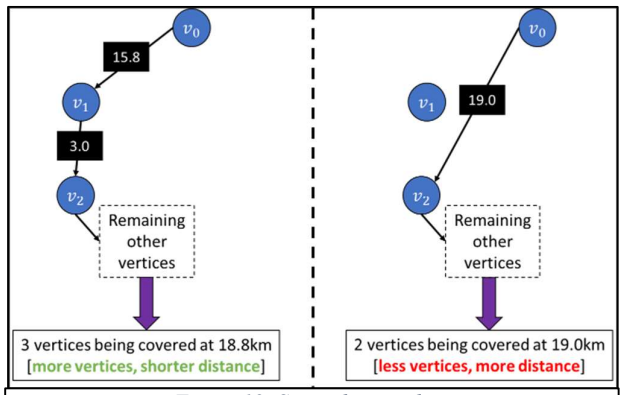


Figure 19: Streamlining edges

Therefore, the edge connecting v_0 and v_2 would be referred to as an **inefficient edge**, because there exists a more **efficient (indirect) path** connecting the 2 vertices, which travels more vertices at a shorter distance.

However, the belief that removing inefficient edges increases the chances of yielding the shortest distance HC, is merely a logical assumption. There can be still a possibility that the **true shortest distance HC involves an inefficient edge**, and thus, is this assumption's **limitation**. Therefore, to check this assumption's reliability, I need to compare a HC, obtained from the involvement of this 'streamlining' process, to one where it has not been implemented.

Nevertheless, for now, I believe that there are other instances within the data I have collected (in Table 4), where the **Triangle Inequality Theorem** was violated. To increase the likelihood of NNA providing the most accurate shortest distance HC, I need to remove other such inefficient edges, where a shorter indirect path exists.

I can simplify this process by using the **Floyd Warshall Algorithm**, which goes as follows (Table 6):

Floyd Warshall Algorithm (FWA) (Floyd-Warshall Algorithm – Algorithm Hall of Fame, 2024)

For an edge connecting v_i and v_j, FWA determines if there exists an intermediate vertex v_k such that there exists a shorter indirect path connecting vertices v_i and v_j through an intermediate vertex v_k. [i.e., $D(v_i v_k v_j) < D(v_i v_j)$]

Test Scenario

Let E represent the set of edges, where $(v_i v_j)$ denotes the edge connecting v_i and v_j.

Step 1: Select 2 distinct vertices from the Distance Matrix in Table 4.

For instance, let v_0 and v_3 be considered. The edge connecting v_0 and v_3 is $(v_0 v_3)$, and $(v_0 v_3) \in E$.

Step 2: Compare $D(v_0 v_3)$ with $D(v_0 v_k v_3)$, where $v_k \in Q$, such that $D(v_0 v_k v_3) < D(v_0 v_3)$.

This step repeats iteratively until all possible v_k have been considered for the above inequality.

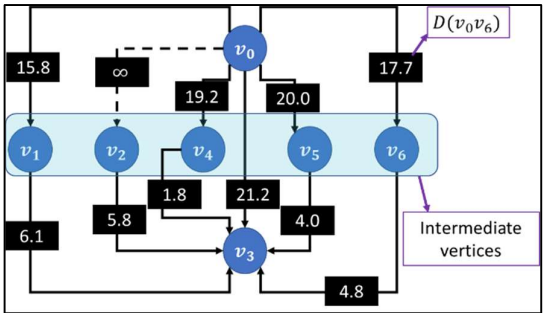


Figure 20:Illustration of possible v_k for $(v_0 v_3)$

In this scenario, v_k is an intermediate vertex for the edge $(v_0 v_3)$, and according to Figure 20, possible v_k are $\boldsymbol{v_1, v_2, v_4, v_5, v_6}$.

To start, let us consider $\mathbf{v_k = v_1}$.

$\therefore D(v_0v_1v_3) = D(v_0v_1) + D(v_1v_3) = 15.8 + 6.1 = 21.9.$

Moving on, let us consider another possible v_k, such as v_2.

This is special, because in Page 12, it was decided that the direct edge (v_0v_2) will be removed, because there exists a shorter path connecting vertices v_0 and v_2 $[v_0 \rightarrow v_1 \rightarrow v_2]$. With the absence of this edge [i.e., $(v_0v_2) \notin E$], it is not possible for a path $[v_0 \rightarrow v_2 \rightarrow v_3]$ to exist. Therefore, to prevent v_2 from being considered as v_k, we need to assign such a large value to $D(v_0v_2)$, such that $D(v_0v_2v_3) = D(v_0v_2) + D(v_2v_3) > D(v_0v_3)$. Instead of the orthodox method of assigning a large finite value (*Floyd-Warshall - Finding All Shortest Paths - Algorithms for Competitive Programming*, n.d.), I decided it would be better to assign a value of infinity (∞) to $D(v_0v_2)$ because this is always greater than any finite value of $D(v_0v_3)$.

For instance,

$$D(v_0v_2v_3) = D(v_0v_2) + D(v_2v_3) = \infty + 5.8 = \infty \text{ and } D(v_0v_3) = 21.2 < \infty$$

$$\Rightarrow D(v_0v_2v_3) > D(v_0v_3)$$

Therefore, v_2 is not suitable as a possible intermediate v_k, leading to its rejection.

Nevertheless, after all possible v_k have been considered, the results are tabulated in Table 5.

v_k	$D(v_0v_kv_3)$	$D(v_0v_3)$
v_1	$= D(v_0v_1) + D(v_1v_3) = 15.8 + 6.1 = 21.9$	21.2
v_2	$= D(v_0v_2) + D(v_2v_3) = \infty + 6.1 = \infty$	
v_4	$= D(v_0v_4) + D(v_4v_3) = 19.2 + 1.8 = 21.0$	
v_5	$= D(v_0v_5) + D(v_5v_3) = 20.0 + 4.0 = 24.0$	
v_6	$= D(v_0v_6) + D(v_6v_3) = 17.7 + 4.8 = 22.5$	

Table 5: Results of $D(v_0v_kv_3)$

Additionally, here it is observed that by assigning a value of ∞ to $D(v_0v_2)$, I was able to instantly reject v_2 as v_k, $D(v_0v_2v_3) = \infty$ is undoubtedly greater than any other finite values of $D(v_0v_kv_3)$, leaving it undesirable. As observed, $D(v_0v_4v_3) < D(v_0v_3)$, hence $v_k = v_4$.

Step 3: Updating the Distance Matrix

Since it was identified that (v_0v_3) is an **inefficient edge** because there exists a shorter, indirect path $[v_0 \rightarrow v_4 \rightarrow v_3]$ connecting vertices v_0 and v_3. Therefore, similar to the removed inefficient edge (v_0v_2), (v_0v_3) will also be removed and the distance matrix would be updated such that $D(v_0v_3) = \infty$, to prevent the edge (v_0v_3) being considered as future intermediate paths. This is observed in Figure 21.

Figure 21: Updating a Distance Matrix ➔

Vertices	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	15.8	19.0	21.2	19.2	20.0	17.7
v_1	15.8	0	3.0	6.1	6.5	9.2	6.9
v_2	19.0	3.0	0	5.8	6.6	9.8	9.0
v_3	21.2	6.1	5.8	0	1.8	4.0	4.8
v_4	19.2	6.5	6.6	1.8	0	3.0	2.2
v_5	20.0	9.2	9.8	4.0	3.0	0	1.9
v_6	17.7	6.9	9.0	4.8	2.2	1.9	0

Initial Distance Matrix

Vertices	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	15.8	∞	∞	19.2	20.0	17.7
v_1	15.8	0	3.0	6.1	6.5	9.2	6.9
v_2	∞	3.0	0	5.8	6.6	9.8	9.0
v_3	∞	6.1	5.8	0	1.8	4.0	4.8
v_4	19.2	6.5	6.6	1.8	0	3.0	2.2
v_5	20.0	9.2	9.8	4.0	3.0	0	1.9
v_6	17.7	6.9	9.0	4.8	2.2	1.9	0

Updated Distance Matrix

Now, this updated distance matrix in Figure 21 would be used for future iterations of **FWA**, where other pairs of vertices would be analysed to determine if there exists an intermediate vertex v_k, which shortens the distance between the pair. And whenever there is a scenario of an **inefficient edge** being removed, its distance would be updated to ∞, and the distance matrix would once again be updated, similar to what occurred in Figure 21.

Step 4: Repeat Steps 1 to 3 until all pairs of distinct vertices have been considered

Summary

- The distance between 2 vertices will be 0, if the 2 vertices are the same. $[D(v_i v_j) = 0 \because v_i = v_j]$
- The distance between 2 vertices will be ∞, if there exists an inefficient edge connecting these 2 vertices. $[D(v_i v_j) = \infty \because D(v_i v_j) > D(v_i v_k v_j)]$
- The distance between 2 vertices will remain as it is, if there is not an inefficient edge connecting these 2 vertices. $[D(v_i v_j) = D(v_i v_j) \because D(v_i v_j) < D(v_i v_k v_j)]$

Table 6: Floyd Warshall Algorithm

As observed, **FWA** has repeated iteratively until all pairs of vertices have been considered, to determine if there exist inefficient edges in the presence of shorter, indirect paths. Therefore, numerous **inefficient edges** have been removed, and the distance matrix in Table 4, has thus been updated multiple times according to **Step 3** in Table 6.

The new revised Distance Matrix is now in Table 7, along with its corresponding weighted graph in Figure 22.

Vertices	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	15.8	∞	∞	19.2	∞	17.7
v_1	15.8	0	3.0	6.1	6.5	∞	6.9
v_2	∞	3.0	0	5.8	6.6	9.8	∞
v_3	∞	6.1	5.8	0	1.8	4.0	∞
v_4	19.2	6.5	6.6	1.8	0	3.0	2.2
v_5	∞	∞	9.8	4.0	3.0	0	1.9
v_6	17.7	6.9	∞	∞	2.2	1.9	0

Table 7: Updated Distance Matrix of vertices v_0 to v_6

I could have employed the **FWA** again on Table 7, to further streamline the updated distance matrix. However, this would definitely cause more edges to be removed, to the extent that there are insufficient edges present to even construct a HC. Hence, the **FWA** has only been applied once.

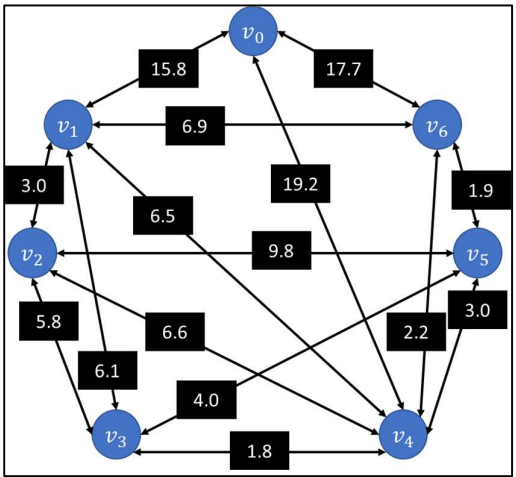


Figure 22: Incomplete Weighted Graph (Table 7)

The **weighted graph** provided in Figure 22 is referred to as an '**incomplete weighted graph**', because there does not exist an edge between every pair of vertices. Therefore, a **complete weighted graph** is the one provided in Figure 17 (page 10), because there exists an edge between every pair of vertices.

Chapter 3: Using NNA to determine Shortest Distance HC

As aforementioned in **Methodology**, the Exhaustive Search Algorithm (ESA) is impractical to determine the shortest distance HC, because it generates too many HCs for consideration. From **Chapter 1**, a total of **7** vertices have been obtained, and according to Figure 13 (Page 8), it is proven that that ESA would generate more HCs for consideration than NNA would, as number of vertices exceed five ($7 > 5$).

$$Number\ of\ distinct\ HCs\ generated\ by\ ESA = \frac{(7-1)!}{2} = 360$$

$$Number\ of\ distinct\ HCs\ generated\ by\ NNA \leq 7$$

Therefore, it is practical to use NNA algorithm over the ESA. Hence, with reference to the distance matrix presented in Table 7, I will be employing NNA to determine the shortest distance HC.

Using NNA

Determining List of HCs

Let V be the set of vertices: $V \in \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$.

Let U be the set of unvisited vertices, such that $U \subseteq V$. Since none of the vertices are visited at the beginning of the algorithm, $U \in \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$, however, it reduces as more vertices are being visited.

Step 1: Choose an initial vertex

NNA starts by selecting an initial vertex from U. Let v_0 be selected from U. P_0 denotes the HC being constructed with v_0 as its initial vertex. Since v_0 is visited, v_0 is removed from the set U $[U\backslash\{v_0\}]$. Therefore, $U \in \{v_1, v_2, v_3, v_4, v_5, v_6\}$

Step 2: Choosing the nearest vertices

Vertices	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	15.8	∞	∞	19.2	∞	17.7
v_1	15.8	0	3.0	6.1	6.5	∞	6.9
v_2	∞	3.0	0	5.8	6.6	9.8	∞
v_3	∞	6.1	5.8	0	1.8	4.0	∞
v_4	19.2	6.5	6.6	1.8	0	3.0	2.2
v_5	∞	∞	9.8	4.0	3.0	0	1.9
v_6	17.7	6.9	∞	∞	2.2	1.9	0

Figure 23: v_0 in Distance Matrix (Table 7)

Consequently, from v_0, P_0 extends by choosing the closest vertex to v_0, from U. This can be done so, by referring to the distance matrix, because the closest vertex to v_0 would correspondingly refer to the edge connecting that vertex to v_1. Therefore, referring to Figure 23, we need to choose the edge with the minimum value which is non-zero, otherwise that would signify v_0 travelling back to itself. As observed, the smallest value is 15.8, corresponding to the edge (v_0v_1). Therefore, P_1 extends itself by travelling from v_0 to v_1. $[v_0 \rightarrow v_1]$

Vertices	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	15.8	∞	∞	19.2	∞	17.7
v_1	15.8	0	3.0	6.1	6.5	∞	6.9
v_2	∞	3.0	0	5.8	6.6	9.8	∞
v_3	∞	6.1	5.8	0	1.8	4.0	∞
v_4	19.2	6.5	6.6	1.8	0	3.0	2.2
v_5	∞	∞	9.8	4.0	3.0	0	1.9
v_6	17.7	6.9	∞	∞	2.2	1.9	0

Figure 24: v_1 in Distance Matrix (Table 7)

Since v_1 has been visited, $U\backslash\{v_1\}$ because it is no longer an unvisited vertex. Therefore, now, $U \in \{v_2, v_3, v_4, v_5, v_6\}$. P_0 once again extends itself by choosing a vertex from U, which is closest to v_1 (in terms of distance). As observed in Figure 24, this would be v_2. Thus, P_1 is now $[v_0 \rightarrow v_1 \rightarrow v_2]$. After which v_2 would be visited, resulting in $U\backslash\{v_2\}$ and $U \in \{v_3, v_4, v_5, v_6\}$.

Step 2 repeats itself till there are no more elements (i.e., vertices) in U; $U\epsilon\ \{\}$.

At this point, P_0 would be $[v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_5]$. Upon reaching v_5, all non-starting vertices would have been visited (exactly once), therefore the objective is to close the HC P_0 by returning back to the starting vertex v_0. For this, there should be a direct edge (v_5v_0), otherwise travelling to the starting vertex involves visiting the non-starting vertices again, violating the conditions of a HC. But the edge (v_5v_0) was removed by FWA (in **Chapter 2**), because it was identified to be **inefficient**, and thus assumed to be undesirable in the formation of a HC, because it would then involve travelling less vertices at a longer distance, which could then have a greater possibility of not being the shortest distance HC.

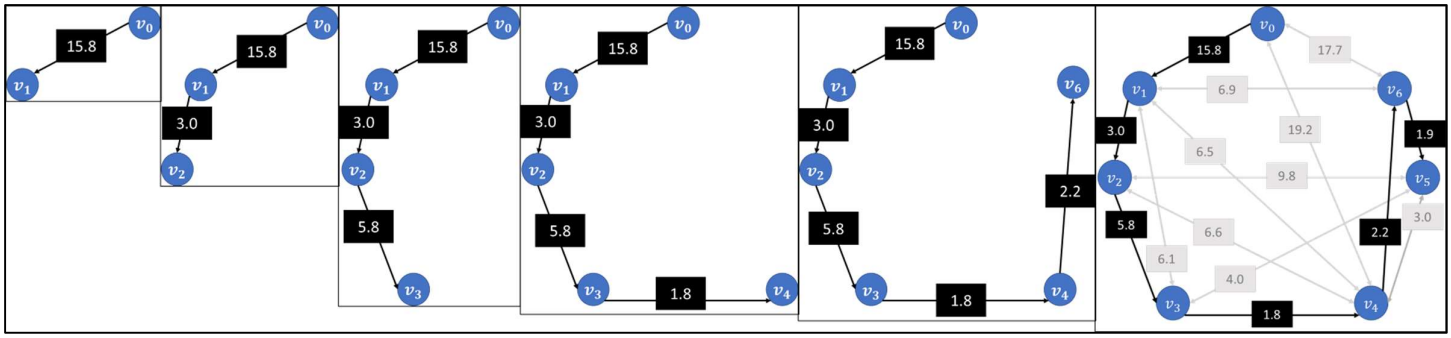


Figure 25: Construction of P_0 $[v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_5]$.

Therefore, as observed in Figure 25, P_0 is unable to close due to the lack of the **inefficient edge** (v_5v_0). Visibly, it is easy to rectify this solution by editing P_0 such that,

$$[v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_5] \rightarrow [v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6]$$

This is because the direct edge (v_6v_0) allows P_0 to close. Nevertheless, this discrepancy is observed only in the first iteration of NNA. Nevertheless, NNA repeats iteratively until all the vertices have been selected as the initial vertex, meaning the formation of $P_1, P_2, P_3, P_4, P_5, P_6$, illustrated in Figure 26. Hence, a total of 7 HCs is constructed.

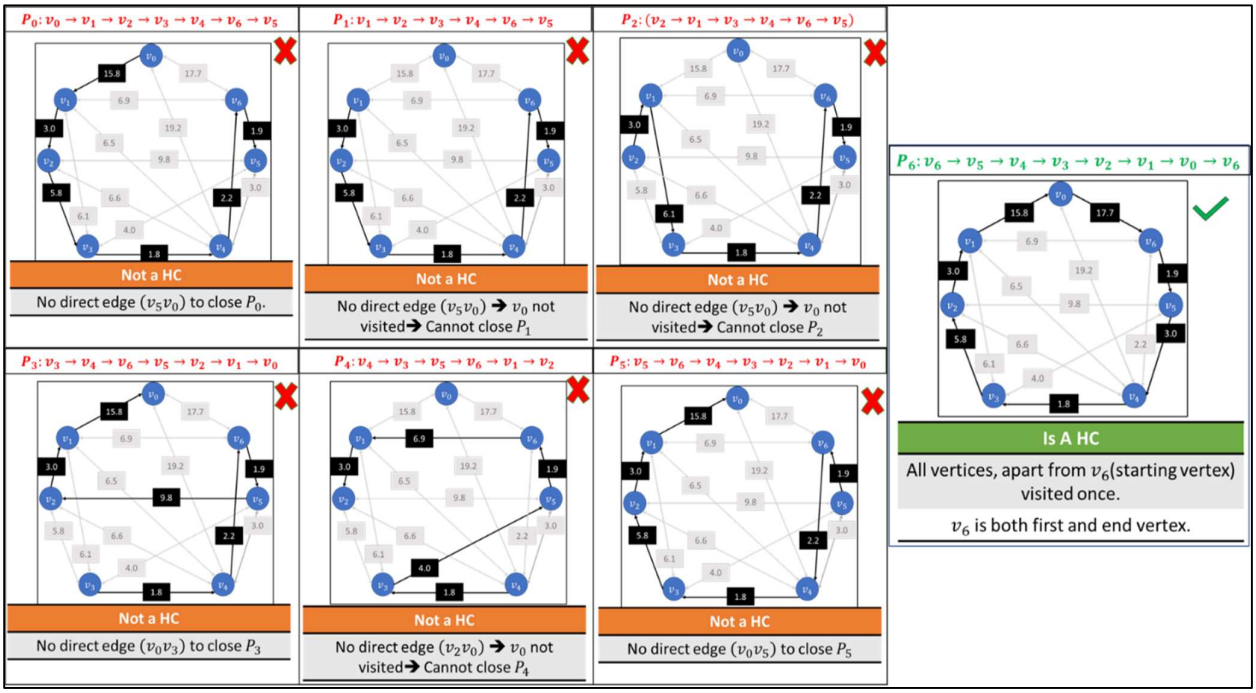


Figure 26: Distinct HCs formed using NNA →

Identifying Shortest Distance HC from NNA

Before determining the shortest distance HC, how do we calculate the distance of a HC? This distance of each HC is calculated by the addition of the distances between the vertices part of the path. Let us consider P_0.

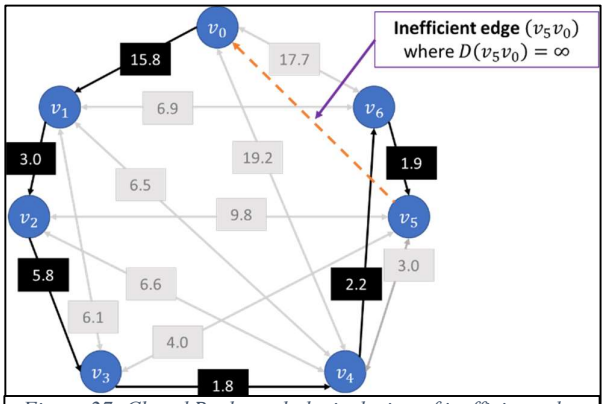


Figure 27: Closed P_0 through the inclusion of inefficient edge

From NNA, the identified path of P_0 is

$$[v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_5].$$

However, in order to close P_0, P_0 should have been

$$[v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_5 \rightarrow v_0],$$

which includes the inefficient edge (v_5v_0) [Figure 27].

Therefore, the distance of P_0, or

$$\begin{aligned} D(v_0v_1v_2v_3v_4v_6v_5v_0) &= D(v_0v_1) + D(v_1v_2) + D(v_2v_3) + D(v_3v_4) + D(v_4v_6) + D(v_6v_5) + D(v_5v_0) \\ &= 15.8 + 3.0 + 5.8 + 1.8 + 2.2 + 1.9 + \infty \\ &= \infty \end{aligned}$$

Therefore, for whichever HC identified in Figure 26 that is not closed, which are P_1, P_2, P_3, P_4 and P_5, it would require the presence of an inefficient edge to close it. Given that these inefficient edges have a corresponding assigned distance of ∞, these 'open' HCs would have an overall distance of ∞ if it were to be closed through the inclusion of an inefficient edge, as observed in Table 8.

Hamiltonian Circuit	Calculations	Distance
P_0	$15.8 + 3.0 + 5.8 + 1.8 + 2.2 + 1.9 + \infty = \infty$	∞
P_1	$3.0 + 5.8 + 1.8 + 2.2 + 1.9 + \infty + 15.8 = \infty$	∞
P_2	$3.0 + 6.1 + 1.8 + 2.2 + 1.9 + \infty + \infty = \infty$	∞
P_3	$1.8 + 2.2 + 1.9 + 9.8 + 3.0 + 15.8 + \infty = \infty$	∞
P_4	$1.8 + 4.0 + 1.9 + 6.9 + 3.0 + \infty + 19.2 = \infty$	∞
P_5	$1.9 + 2.2 + 1.8 + 5.8 + 3.0 + 15.8 + \infty = \infty$	∞
P_6	**$1.9 + 3.0 + 1.8 + 5.8 + 3.0 + 15.8 + 17.7 = 49km$**	**$49km$**

Table 8: Distances of each HC

In Table 8, it became obvious that by attributing a distance of ∞ to the **inefficient edges** in **FWA**, it was easy to distinguish which HCs contained inefficient edges and which did not. Apart from $D(P_6)$, every other HC had a distance which was ∞, because summations involving ∞ would only return ∞. Therefore, the verification process was straightforward and it was determined that **P_6 is the shortest distance HC**, because finite distances are always less than infinite distances.

However, the starting vertex of P_6 is v_6, which is not my aunt's place of residence (v_0). Nevertheless, given that a HC is a closed loop, changing the starting vertex is essentially just rotating the HC. Therefore, as observed in Figure 28, P_{min} is as such:

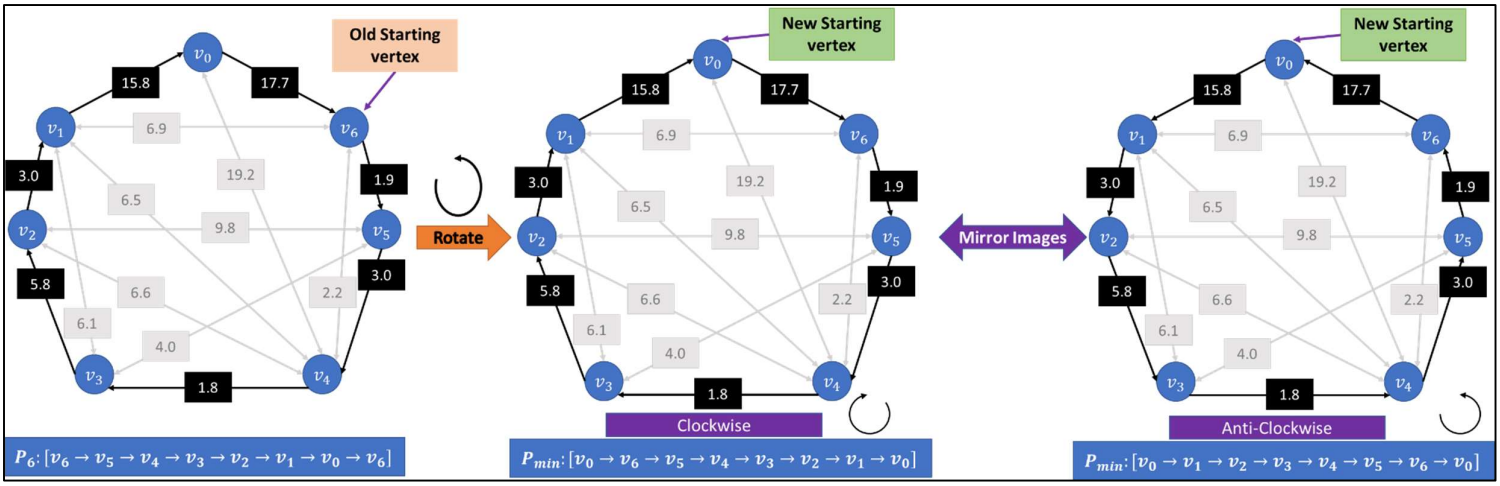


Figure 28: Changing the initial vertex of P_6

Results from NNA

Therefore, the shortest distance Hamiltonian Circuit determined from the NNA (P_{min}) is either $[\boldsymbol{v_0 \rightarrow v_6 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1 \rightarrow v_0}]$ or its anti-clockwise arrangement, as illustrated in Figure 28. Nevertheless, both arrangements will have a corresponding distance of **49km**, as determined in Table 8, because the edges are bi-directional, meaning that $D(v_0v_6v_5v_4v_3v_2v_1v_0) = D(v_0v_1v_2v_3v_4v_5v_6v_0)$.

But how reliable is this result? As observed in the Venn Diagram in Figure 14 (page 8), there is a possibility that the **shortest distance HC identified by NNA would not be the true shortest distance HC**.

Additionally, a limitation of the **Floyd Warshall Algorithm (FWA)** employed in **Chapter 2** is that there is a possibility that it may remove an inefficient edge that could have been part of the true shortest distance HC. Hence, this **could have prevented the formation of the true shortest distance HC through NNA.**

Therefore, the only method to verify if the current obtained P_{min} is indeed the shortest distance HC, is to compare it against the results of ESA, which would always determine the true shortest distance HC. ESA would use the original distance matrix (Table 4), on which FWA has not been employed, meaning that distance matrix contains both efficient and inefficient edges, to consider the scenario where the shortest distance HC contains an inefficient edge.

Verifying the accuracy of P_{min} determined using ESA

In the current scenario, there are a total of **7** vertices, which means that a total of $\frac{(7-1)!}{2} = 360$ distinct HCs will be formed. It is impractical to consider all these HCs by manual calculations. Nevertheless, through code, determining if P_{min} is indeed the **true shortest distance HC** becomes less tedious. The code is displayed in **Appendix A**, and its results are observed in Figure 29.

```
Circuit: [0, 1, 2, 3, 4, 5, 6, 0], Distance: 49.0
  ↳ Anti-clockwise arrangement

Circuit: [0, 6, 5, 4, 3, 2, 1, 0], Distance: 49.0
  ↳ Clockwise arrangement
```

Figure 29: Code Results from ESA

From Figure 29, it is determined that the true shortest HC (P_{true}) is $[\boldsymbol{v_0 \rightarrow v_6 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1 \rightarrow v_0}]$ or its anti-clockwise orientation. Nevertheless, given $P_{true} = P_{min}$, this means that both the FWA pre-treatment to remove **inefficient edges**, and the heuristic NNA are reliable because they yielded the **true minimum distance** HC.

Analysis

From **Chapters 1 till Chapter 3**, I have optimised the delivery route my aunt takes to visit her frequent customers, grouped based on their differing localities. Additionally, the aim of this exploration was to:

determine the shortest distance Hamiltonian Circuit my aunt takes to visit her customers' places, given her place of residence is the starting vertex, such that ***travel time takes less than 3 hours*** (Page 3).

Therefore, in order to determine the efficiency of the new optimised delivery route, it has to be compared against the prevailing one, by determining the percentage decrease **in either travel distance or travel time**. However, my aunt has no records of the distance of her delivery routes, but rather she provided an average of 3 hours travel time.

Hence, as aforementioned in Page 11, under **Chapter 2**, in order to find travel time, I would be dividing the distance of the optimised delivery route, with the average vehicular speed in the city my aunt and her customers are (i.e., Hyderabad in India), which is 25km/h (Deshpande, 2023).

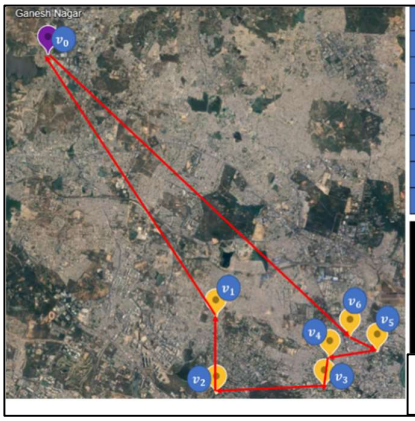
$$\therefore Travel\ time = \frac{distance}{speed} = \frac{49.0}{25} = 1.96h \approx 1\ hour\ 58min$$

$$Percentage\ decrease\ in\ travel\ time = \frac{|1.96 - 3|}{3} \times 100\% = 34.7\%$$

Conclusion

The aim of the investigation is to determine the shortest distance Hamiltonian circuit my aunt takes to visit her customers' places, such that her travel time is less than 3 hours.

Through this exploration, it was determined that the shortest distance HC determined through NNA (P_{min}) is $[\boldsymbol{v_0 \rightarrow v_6 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1 \rightarrow v_0}]$, or its anti-clockwise arrangement (Figure 28). This optimised delivery route has **significantly reduced travel time by 34.7%**, and given that its travel time is less than 3 hours at 1.97 hours, **the aim has therefore been fulfilled.** With reference to Table 3, the vertices have been converted to the respective localities they represent, and the optimised delivery route is as observed in Figure 30:



Vertex	Locality
v_0	Ganesh Nagar (Aunt's Residence)
v_1	Kavuri Hills
v_2	Prashasan Nagar
v_3	Resham Bagh
v_4	Sri Nagar Colony
v_5	Greenlands
v_6	Srinivasa Nagar

Route (Hamiltonian Circuit) Taken:
$[v_0 \rightarrow v_6 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1 \rightarrow v_0]$

[Ganesh Nagar → Srinivasa Nagar → Greenlands → Sri Nagar Colony → Resham Bagh → Prashasan Nagar → Kavuri Hills → Ganesh Nagar]

Figure 30: Shortest Distance Hamiltonian Circuit

Potential Extensions

In the future, as my aunt's business expands, it may eventually become impractical for my aunt to deliver her goods to all her customers herself. Therefore, a potential solution would be her hiring delivery personnel. However, contrary to the current exploration where only one postman or woman (i.e., my aunt) handles deliveries, multiple postmen would be involved. Therefore, in terms of TSP, this shift translates from a singular postman problem (i.e., this exploration) to one involving multiple postmen.

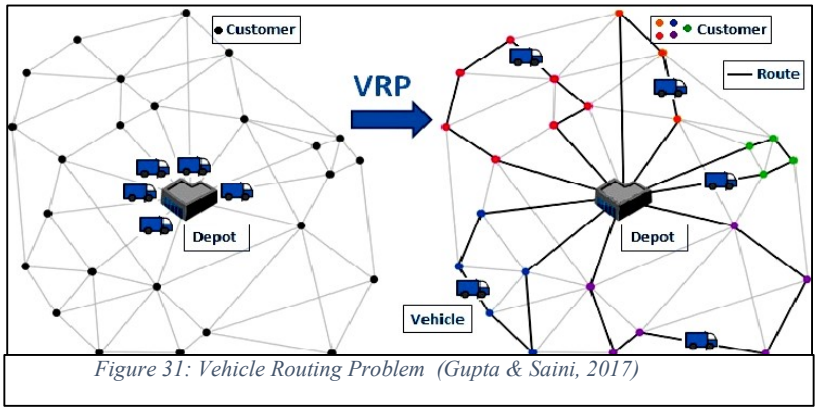


Figure 31: Vehicle Routing Problem (Gupta & Saini, 2017)

This therefore transforms from a 'Travelling Salesman Problem' to a 'Vehicle Routing Problem (VRP)'. VRP involves designing optimum (shortest distance) routes for a fleet of vehicles, where each vehicle delivers to a specific group of customers (Gora et al., 2020).

Nevertheless, as observed in Figure 31, a HC is still used to create a route between every group of customers, because each customer is visited only once, and the starting and ending location is the same (i.e., the Depot). Therefore, VRP can be divided into 2 parts:

1. Determining the different groups of customers
2. Determining the shortest distance HC for each group of customers

And in the second part of VRP, this exploration still becomes relevant, because we have earlier identified how to determine the shortest distance HC for my aunt to visit all her frequent customers (i.e., a single group of customers).

Evaluation of Methods

However, in this exploration, some significant assumptions and considerations were made to help determine the shortest distance HC. And these assumptions have their corresponding limitations, which could have incorrectly answered the initial aim determined in this exploration (Page 2).

Assumptions and Considerations

1. ***The shortest distance taken to travel to the customers' places would be a HC.***

This assumption was made solely to transpose and solve my aunt's problem within the context of TSP. Although the notion of a HC is slightly accurate because the starting and ending locations are the same (i.e., my aunt's location), there is still a possibility that there is an alternative, shorter route that involves travelling to frequent customers' locations more than once. Hence, the shortest distance HC discovered through this exploration may not have been the shortest route to travel to the frequent customers' places.

2. *Determining the efficiency of the optimised delivery route*

In this exploration, my aunt's frequent customers were grouped based on their respective localities, to minimise the number of vertices present and simplify the identification of the shortest distance HC. However, this implies that the shortest distance HC we have identified (P_{min}) only represents the shortest distance to travel between the various localities to which my aunt frequently delivers to. As depicted in Figure 15 (Page 9), each of these localities being analysed have more than one customer residing in it. Therefore, in this exploration, we have thus assumed that the travel distance, and consequently, travel time within each locality to reach the individual customers' locations is **negligible**.

However, my aunt's average delivery time of 3 hours encompasses both the travel time to the locality and within the locality to reach the customers' exact locations. This means that her average delivery time to just the localities would be less than 3 hours, meaning that the optimised route only reduces travel time by **less than 34.7%**.

Evaluating Scalability

As my aunt's business expands, the number of customers and localities she would be serving to increases – a direct increase in number of vertices. Although this exploration suggests that my optimisation approach is reliable in determining the shortest distance HC, this reliability may not hold for other scenarios with more vertices. Therefore, this begs the question whether or not my optimisation approach is truly scalable.

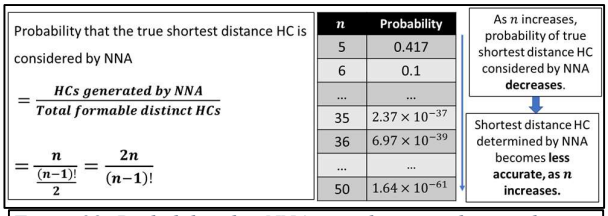


*Figure 32: Probability that NNA considers true shortest distance HC as **n** increases*

NNA is designed to generate the shortest distance HC, but it may not consistently produce optimal results as number of vertices increases, as suggested in Figure 32. Additionally, NNA's efficiency depends on FWA's removal of inefficient edges. If there exist too many inefficient edges within the distance matrix, their removal by FWA might prevent the formation of a HC altogether. Besides, there is a possibility that some of the inefficient edges removed were part of the shortest distance HC in certain scenarios.

Therefore, it is not possible to definitively ascertain whether my optimisation approach involving NNA and FWA 'pre-treatment' is indeed scalable without testing it across additional scenarios with varying number of vertices.

Strengths

Manipulating distance of edges to enable more algorithmic control

In FWA, it has been established why ineffective edge are removed and reassigned a value of ∞ (Page 13). By extension, this approach also enables my aunt to manipulate other routes by assigning them a distance of ∞ if they are not preferred. For instance, if my aunt wishes to avoid travelling from v_i to v_j through the edge $(v_i v_j)$, she can assign its distance as ∞, even if it is not an inefficient edge. This thereby forces NNA to generate a HC that does not include this edge, providing my aunt more control over the algorithmic outcome. Nevertheless, this strategy means that the overall route connecting all vertices is suboptimal (i.e., not the shortest).

References

Deshpande, A. (2023). *Average speed of vehicles in Hyd. Is 25 kmph, say police—The Hindu*. https://www.thehindu.com/news/cities/Hyderabad/average-speed-of-vehicles-in-hyd-is-25-kmph-say-police/article37905650.ece

Distance Matrix—An overview | ScienceDirect Topics. (n.d.). Retrieved 25 March 2024, from https://www.sciencedirect.com/topics/mathematics/distance-matrix

Floyd-Warshall Algorithm – Algorithm Hall of Fame. (2024, January 3). https://www.algorithmhalloffame.org/algorithms/graph-algorithms/floyd-warshall-algorithm/

Floyd-Warshall—Finding all shortest paths—Algorithms for Competitive Programming. (n.d.). Retrieved 25 March 2024, from https://cp-algorithms.com/graph/all-pair-shortest-path-floyd-warshall.html

Gilbert, S., & Halim, S. (n.d.). TRAVELLING-SALESMAN-PROBLEM (4 variants).

Gora, P., Bankiewicz, D., Karnas, K., Kaźmierczak, W., Kutwin, M., Perkowski, P., Płotka, S., Szczurek, A., & Zięba, D. (2020). Chapter 2 - On a road to optimal fleet routing algorithms: A gentle introduction to the state-of-the-art. In J. Nalepa (Ed.), *Smart Delivery Systems* (pp. 37–92). Elsevier. https://doi.org/10.1016/B978-0-12-815715-2.00014-2

Gupta, A., & Saini, S. (2017). An Enhanced Ant Colony Optimization Algorithm for Vehicle Routing Problem with Time Windows. 267–274. https://doi.org/10.1109/ICoAC.2017.8441175

Gutin, G., & Punnen, A. (n.d.). THE TRAVELING SALESMAN PROBLEM AND ITS VARIATIONS.

How we kept information on Maps reliable in 2021. (2022, March 24). Google. https://blog.google/products/maps/how-we-kept-maps-reliable-2021/

Joshi, V. (2017, May 25). A Gentle Introduction To Graph Theory. *Basecs*. https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8

Lecture9.pdf. (n.d.). Retrieved 25 March 2024, from https://math.hawaii.edu/~les/m100/lecture9.pdf

Safra, J. (n.d.). *Triangle inequality | Theorem, Geometry, & Degenerate | Britannica*. Retrieved 25 March 2024, from https://www.britannica.com/science/triangle-inequality

Search Algorithms. (n.d.). Retrieved 25 March 2024, from https://sawtoothsoftware.com/help/lighthouse-studio/manual/search-algorithms.html

Appendix A – Python Code for Exhaustive Search Algorithm

```python
import itertools
def hamiltonian_circuits(distance_matrix):
    n = len(distance_matrix)
    start_vertex = 0
    vertices = list(range(1, n))
    for circuit in itertools.permutations(vertices):
        circuit = [start_vertex] + list(circuit) + [start_vertex]
        distance = sum(distance_matrix[circuit[i-1]][circuit[i]] for i in range(1,
n+1))
        yield circuit, distance

distance_matrix = [
    [0, 15.8, 19.0, 21.2, 19.2, 20.0, 17.7],
    [15.8, 0, 3.0, 6.1, 6.5, 9.2, 6.9],
    [19.0, 3.0, 0, 5.8, 6.6, 9.8, 9.0],
    [21.2, 6.1, 5.8, 0, 1.8, 4.0, 4.8],
    [19.2, 6.5, 6.6, 1.8, 0, 3.0, 2.2],
    [20.0, 9.2, 9.8, 4.0, 3.0, 0, 1.9],
    [17.7, 6.9, 9.0, 4.8, 2.2, 1.9, 0]
]
for circuit, distance in hamiltonian_circuits(distance_matrix):
    print(f"Circuit: {circuit}, Distance: {distance}")
```