Local Re-adjustment based approaches for Load Balanced Network Voronoi Diagram

by

Ankita Mehta

Under the supervision of Dr. Viswanath Gunturi

Indraprastha Institute of Information Technology, Delhi

May 2017

Local Re-adjustment based approaches for Load Balanced Network Voronoi Diagram

by

Ankita Mehta

Submitted
in partial fulfilment of the requirements for the degree of
Master of Technology

to

Indraprastha Institute of Information Technology, Delhi

May, 2017

Certificate

This is to certify that the thesis titled ***"Local Re-adjustment based Approaches for Load Balanced Network Voronoi Diagram"*** being submitted by **Ankita Mehta** to the **Indraprastha Institute of Information Technology, Delhi** for the award of Master of Technology, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr. Viswanath Gunturi
Department of Computer Science
IIIT, Delhi

May, 2017

Acknowledgements

I take this opportunity to express my profound gratitude and deep regards to my supervisor **Dr. Viswanth Gunturi** for giving me the opportunity to work on this project. The door to his office was always open whenever I had a doubt regarding my research or writing. He constantly provided with mentorship and steered me in the right direction whenever I needed it. I would not have been able to complete my thesis work this smoothly in IIIT Delhi without his handholding and consistent support.

I would like to express my gratitude to Dr. Ojaswa Sharma, IIITD, Dr. Vikram Goyal, IIITD, and Prof. Ramnath Sarnath, St. Cloud State University, for their feedback and support during the formation of the solution of this project.

I extend my gratitude to my team mates, Kapish Malik, Anurag Goel who helped me with their support in the project. Best of k paths algorithm and min-cost bipartite matching algorithm (two very important modules of this thesis) were both coded by Kapish. Without his support, my results would never have been accomplished. I would like to thank Pooja Sethia and Aditi Aggarwal who contributed in the project while providing comparative analysis with the related work. Their contribution highly benefitted this project during the submission of this work in DASFAA 2016, SSTD 2017.

I would like to thank Infosys Center for Artificial Intelligence at IIIT Delhi and DST-SERB (grant number ECR/2016/001053) for their fund and resource support.

Finally, I would like to thank my parents and my friends for providing me with unfailing support and continuous encouragement throughout the journey of my thesis.

Abstract

Input to the Load Balanced Network Voronoi Diagram (LBNVD) consists of the following: (a) a transportation network represented as a directed graph; (b) locations of public service units (e.g., schools in a city) as vertices in the graph and; (c) locations of demand (e.g., school children) also as vertices in the graph. In addition, each service center is also associated with a notion of capacity and a penalty which is incurred if it gets overloaded. Given the input, the goal of the LBNVD problem is to determine a partition of the demand vertices around service centers where each service center gets a unique partition. The objective here is to generate a partition which minimizes the sum of the total distance between demand vertices and their associated service center and the penalties incurred. The problem of LBNVD finds its application in the domain of urban planning. The current state of the art related to LBNVD makes simplifying assumptions such as infinite capacity or total capacity being equal to the total demand. We propose a novel concept of Local Re-Adjustment based approaches for the problem of LBNVD. Using this concept, we propose three algorithms for the LBNVD problem. The proposed algorithms are evaluated both theoretically and experimentally using real datasets.

Contents

List of Figures

Chapter 1

Introduction

Transportation is a fundamental activity in modern society. It encompasses from something as simple as commuting to office or dropping-off your child at school to something as complicated as evacuating several towns or districts as was the case in case of cyclone Phailin (ref the Times of India 13 Oct 2013) which affected about 9 million people and required evacuating several lakhs of people from the coasts of Odisha and Andhra Pradesh in 2013. The former example on daily commute could be referred to as micro-level transportation (terminology followed in this proposal) since each individual is attempting to optimize his or her own travel, whereas the latter example on evacuation is referred to as macro-level transportation (terminology followed in this proposal) as we need to plan the movement of a large mass of population.

Figure 1.1 show a sample map of delhi extracted using osmar library of r via open street maps 3.1. This graph shows a sample out of voronoi diagram with service centers as major metro stations and demand as roadways.

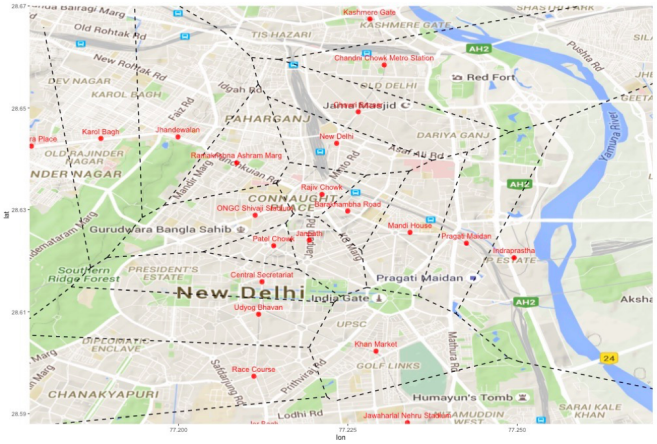

FIGURE 1.1: Voronoi Diagram of Map of Delhi with Service centers as Metro stations

In this thesis we solve an assignment problem, where some demand is assigned some service units. Given the input:(a) a transportation network as a directed graph G(V,E) with V being the set of vertices and E being the set of edges; (b) a set of service centers (e.g. schools, police stations, hospitals, etc.) with positive integer capacities as vertices in graph; (c) a set of penalty costs for service centers which denotes the *"cost"* that needs to be paid to overload a particular service center and; (d) a set of demand vertices, also as vertices in the graph. The goal of Load Balanced Network Voronoi Diagram (LBNVD) problem is to determine a partition of the input graph where each service center is associated with a set of unique demand vertices.

1.1 Problem Motivation:

The problem of LBNVD finds its application in the area of urban planning, e.g. defining the zones of operation of schools or police stations based on their capacities. For instance, consider the task of defining the zone of operation for each school (school catchment area). One can imagine that a school would have only a limited number of teachers and infrastructure support which would define its *capacity*. Furthermore it is also conceivable that overloading a school would result in decrease in the quality of education (this is modelled as penalty). Now in such a situation if we were to define the zone of operation of each school, one would have to partition the home addresses (demand vertices) by considering the following factors: (1) parents would usually prefer to send their children to a school which is close-by and, (2) quality of education in schools should not be bad. This scenario can be mapped to our LBNVD problem.

1.2 Outline:

The rest of this thesis is organized as follows: In Section 2, we provide the related work to the problem. In section 3 we provide a background on the terms and techniques used in the proposed approaches as necessary for this thesis. In section 4, we provide the basic concepts and the problem statement and also a detailed execution trace of our proposed approaches. Section 5 presents our proposed approaches. Section 6 provides an experimental evaluation of our approaches (and the related work) on real datasets.

Section 7 provide detailed theoretical analysis, correctness and complexity of our proposed approaches. Finally in Section 8 we conclude the report. As part of appendix we also provide a reduction of problem of LBNVD to min-cost bipartite matching algorithm (which is the optimal solution to LBNVD problem).

Chapter 2

Related Work

This chapter provides details on the related work with respect to the problem statement of this thesis.

2.1 Network Vornoi Diagram

The current state of the art most relevant to our problem includes the work done in the area of Network Voronoi Diagram [1, 2] and Capacity Constrained Network Voronoi Diagram (CCNVD) [3, 4]. Network Voronoi Diagram (NVD) assumes that the capacity of service center is infinite. Capacity Constrained Network Voronoi Diagram (CCNVD) extends NVD by incorporating the notion of capacity. However, they assume that the total capacity of the service centers should be equal to the total demand. Our problem can be considered more general in the sense that we don't constraint the capacity. For e.g., the total capacity of service centers in our problem may be less, equal or more than the total demand. Furthermore, by relaxing the constraint on capacity, we bring in the novel concept of "Load Balancing" in network voronoi diagrams.

2.2 Facility Location and k-means Problems

In relation to the proposed problem, one could also consider the work done in the area of facility location problem [5–8] and k-median [9] problem.
In their most general form, facility location and k-median problems aim to find a set of optimal locations such that opening facilities at these locations would yield minimum total distance between demand locations and their nearest service centers. More general

forms of these problems have been proposed to include notions like capacity, annual operating costs [8], etc.

The proposed LBNVD problem is different from these. Unlike these problems, in our case we already have a set of k facilities which are up and running, and we want to load balance the demand around them.

2.3 Optimal Solution

One my note that literature has also proposed [10] solutions based on Min-Cost Bipartite Matching to produce optimal solution(s) for our problem. However, they do not scale for large graphs.

2.3.1 Min-Cost Bipartite Matching Algorithm

The algorithm [10] finds a minimum cost complete matching between the two vertex subsets of a given weighted bipartite graph. Using Hungarian algorithm, we can provide an assignment of nodes of one part of the bipartite graph to its other part. This is analogous to assigning demand nodes to service centers. We provide the reduction of our problem to min-cost bipartite matching algorithm as part of Appendix A.

2.3.2 Limitation of Optimal Solution

However, our experiments reveal that the solution provided by min-cost bipartite matching algorithm are not scalable in case of large transportation networks.

Chapter 3

Background

This chapter explains the terms and techniques used with regard to this thesis.

3.1 Open Street Maps

Open Street Maps [11] is a collaborative project to support and enable the development of freely-reusable geospatial data. The map data is crowd-sourced and is available under an open license. With the advent of inexpensive portable satellite navigation devices has primarily enabled the growth of open street maps. Map data is collected from scratch by volunteers performing systematic ground surveys using tools such as a handheld GPS unit, a notebook, digital camera, or a voice recorder. Crowd-sourced OSM has outdone Google Maps at various instances, one of them being provision of detailed maps of areas in and around Sochi during the 2014 Winter Olympics.

The data generated by the OSM project is now used in many applications like its usage by MapQuest Open, Foursquare (replaced Google Maps), Apple, etc. Though the quality of collected data varies worldwide, the current number of registered users nearing 1.5 million and still growing is the reason for the growth of Open Street Maps. The project was launched in 2004 by Steve Coast (initially focusing on mapping the United Kingdom) and has been awarded by the Free Software Foundation Europe (FSFE) for its work on Open and Emerging Standards on occasion of the Document Freedom Day.

We use OSM for the data collection due to its open license and ease of usage.

3.2 Shortest Path Algorithms

In this section we provide the details of shortest path algorithm we use as part of the project.

3.2.1 Scope

While shortest path computation is an important component of the algorithm, it is however not the focus of this thesis. We use shortest path algorithm as a black box. As a proof of concept in our algorithm, we use Floyd-Warshall[12] algorithm for computing the shortest paths. One can easily replace Floyd-Warshall algorithm with other shortest path algorithms[13–20]. Evaluating alternative choices for shortest path algorithm can be taken as part of future work.

3.2.2 Floyd-Warshall Algorithm

Floyd-Warshall [12] algorithm is an algorithm for finding the shortest path distance between every pair of vertices in a graph with positive or negative edge weights (but with no negative cycles).

Consider a graph G(V, E) with V vertices and E edges. Consider a function **shortestPath(i, j, k)** which returns the shortest path between vertex *i* and vertex *j* with set *{1, 2, ..., k}* being the intermediate vertices. The goal of this function is to find the shortest path using only vertices from the set *{1, 2, ..., k}*. From each of the pair (i,j), the shortest path could be either:

(1) a path that uses only vertices from the set *{1, 2, ..., k}*, or

(2) a path that goes from *i* to *k + 1* and then from *k+1* to j.

Thus from the concept of dynamic programming, the recursive formula for finding the shortestPath(i, j, k) is given as:

$$1)\ \ \text{shortestPath(i,j,0)} = \text{w(i,j)},\ \ \text{as}\ \ \text{best}\ \ \text{case}$$

$$2)\ \ \text{shortestPath(i,j,k)} = \min\Big(\text{shortestPath(i,j,k)},\ \text{shortestPath(i,k+1,k)} + \text{shortestPath(k+1,j,k)}\Big),\ \ \text{as}\ \ \text{recursive}\ \ \text{case}$$

We can use Floyd-Warshall algorithm to find the all pair shortest path by setting k = N, where N is the no of vertices in a connected graph. The pseudo code given in the Algorithm 3 is the simplest for of Floyd-Warshall [21] and is provided for reader's benefit. One may refer [12] for details.

Algorithm 1 FloydâĂŞWarshall Algorithm: shortestPath (i,j,k)

Input: (a) A graph G(V,E), (b) Source vertex *i*, (c) Target vertex *j*, (d) Set of intermediate vertices from *{1,2,..., k}*

Output: Shortest path distance for all (i,j) pairs

```
1: Let Dist. be a ||V|| × ||V|| array of minimum distances initialized to infinity
2: for each vertex v
3:         Dist[v][v] ← 0
4: for each edge (u,v)
5:         Dist[u][v] ← w(u,v){the weight of the edge (u,v)}
6: for k from 1 to ||V||
7:         for i from 1 to ||V||
8:                 for j from 1 to ||V||
9:                         if Dist[i][j] > Dist[i][k] + Dist[k][j]
10:                                Dist[i][j] ← Dist[i][k] + Dist[k][j]
```

3.3 Heap

Here we discuss Heap data structure used in the our proposed solution. A heap [22] is a specialized tree-based data structure that satisfies the heap property.

Heap Property: If A is a parent node of B, then the key (the value) of node A is ordered with respect to the key of node B with the same ordering applying across the heap.

Heap Property can be applicable as either **min-heap** or **max-heap** with respect to its heap property followed by the parent node is minimum or maximum. Figure 3.1 shows an example of min-heap with integer keys. Note that minimum key must always be present in the root node of the tree.

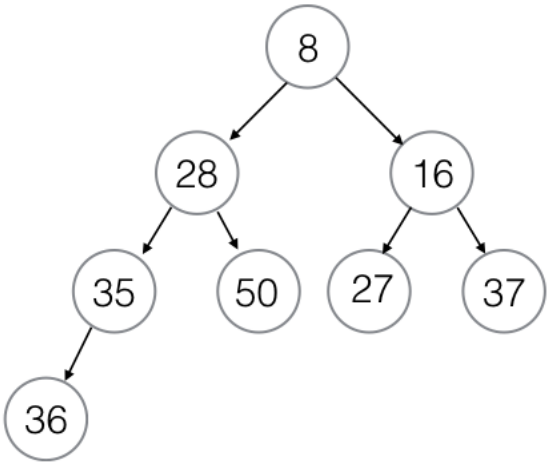

FIGURE 3.1: An example of binary min-heap with integer keys.

Below we explain the two major operations of heap: Insert and extract-min.

3.3.1 Insert operation

In order to add an element to a min heap, a up-heap operation is performed. Following is the psuedocode for the same.

Algorithm 2 Insert in a Heap: insertAndHeapify (heap, key)

Input: (a) Heap Data Structure (H), (b) New Key
Output: Heap with key inserted.

1: Add the element to the bottom level of the heap.
2: Compare the added element with its parent; if they are in the correct order, stop.
3: If not, swap the element with its parent and return to the previous step.

3.3.2 Extract-min operation

In order to get in the minimum element and remove it from the binary heap, we perform the following extract-min operation:

Algorithm 3 Extract minimum from a Heap: extract-min (heap)

Input: (a) Heap Data Structure (H)
Output: Minimum Key.

1: Replace the root of the heap with the last element on the last level.
2: Compare the new root with its children; if they are in the correct order, stop.
3: If not, swap the element with one of its children and return to the previous step. (Swap with its smaller child in a min-heap and its larger child in a max-heap.)

We use binary **min-heap** in our proposed solution as explained in Chapter 5.

Chapter 4

Problem Statement

This chapter formally defines the problem of Load Balanced Network Voronoi Diagram. We first explain the basic concepts and definitions and next detail the problem statement by detailing the input, output and the objective function.

4.1 Basic Concepts

Definition 4.1. A Transportation Network is represented as weighted directed graph G(V,E) with a set of vertices $V = \{v_1, ..., v_{n_v}\}$ and a set of edges $E = \{e_1, ..., e_{n_e}\}$. Each vertex represents a road intersection. A road segment between two intersections is represented as a directed edge. Each edge is associated with a weight $w(u, v)$ which represents the cost to reach from vertex **u** to vertex **v**.

Definition 4.2. A Service Center is a vertex in the transportation network representing a public service unit of a particular kind (for example, schools, police stations or hospitals in a city). A set of service centers is represented as $S = \{s_1, ..., s_{n_s}\}$, where $S \subset V$ and n_s is the number of service centers.

Definition 4.3. A Demand Vertex (a vertex in transportation network) denotes the location of a unit population which is interested in accessing the previously defined service center. A set of demand vertices is represented as $V_d = \{v_{d_1}, ..., v_{d_{n_{v_d}}}\}$, where $V_d \subset V$ and n_{v_d} is the number of demand vertices[1].

Definition 4.4. Network Voronoi Diagram (NVD)

The basic version of the NVD problem takes the following three inputs: (a) a graph G(V,E) representing a transportation network, (b) location of n_s service centers (as

[1]For simplicity, we assume that all population of a city resides on **vertices** of the graph G.

vertices in the graph) and their capacities (an additional input in some variants, refer definition 5) and, (c) spread of population (as set of demand vertices in the graph). The output of the NVD problem is a partition of the demand vertices such that the total sum of distances between every demand vertex and its âĂIJallottedâĂİ service center is minimum. Mathematically, the NVD problem can be stated using the following objective function:

$$\textbf{\textit{Minimize}} \left\{ \sum_{\substack{s_i \in Service \\ Centers}} \left\{ \sum_{\substack{V_{d_j}^{s_i} \in Demand\ vertex \\ alloted\ to\ s_i}} Dist(V_{d_j}^{s_i}, s_i) \right\} \right\} \tag{4.1}$$

Here, s_i, $i \in [1, n_s]$, is a service center and $V_{d_j}^{s_i}$, $j \in [1, n_d]$, is the j_{th} demand vertex assigned to the service center s_i. $Dist(V_{d_j}^{s_i}, s_i)$ is the shortest path distance between $V_{d_j}^{s_i}$ and s_i.

Figure 4.1 illustrates a sample transportation network represented as a directed graph. Here, s_1, s_2 and s_3 are three service centers and $a..k$ are demand vertices. For the sake of simplicity, each vertex is this graph represents unit demand and we assume that a vertex in the input graph cannot be both a demand vertex and a service center. This graph is provided as an input to the NVD problem. Figure 4.2 shows a network voronoi diagram on the input where each partition is shown by filling the demand vertices with the same pattern as of their allotted service center.

Definition 4.5. Capacity of a service center ($c_{s_i}\ \forall\ c_{s_i} \in C = \{c_{s_1}, ..., c_{s_{n_s}}\}$) is the prescribed number of unit demand that a service center s_i can accommodate. For example, the number of beds in a hospital or the number of students a school can admit would correspond to the notion of capacity defined in this paper.

Consider Figure 4.2, which shows a sample allocation of demand vertices to their respective service center for the input graph in Figure 4.1. In the Figure 4.2, service centers s_1, s_2 and s_3 have a capacity of 4, 2, and 4 respectively. Note that in this allocation, s_1 has a capacity of 4 but 5 demand vertices are allotted to it. Thus, the service center s_1 is said to be *overloaded* by 1. However, if s_1 had weaker infrastructure (or facilities), then overloading s_1 may not be wise. To this end, we introduce the notion of load balancing in which only those service center(s) are overloaded which can "take up" the extra load. And if there is more demand than the available capacity, load balancing is an immediate need. We now introduce the notion of penalties to help in load balancing.

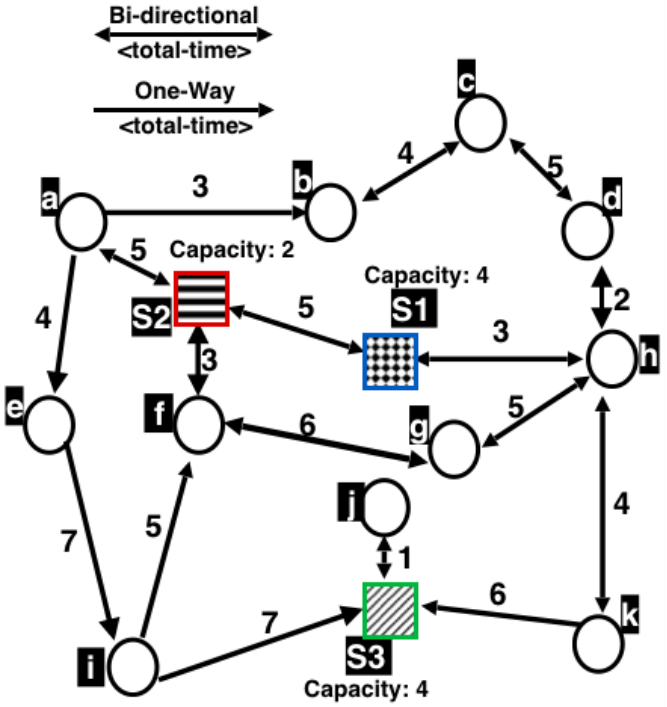

FIGURE 4.1: A transportation network represented as graph with 3 service centers s_1, s_2, and s_3.

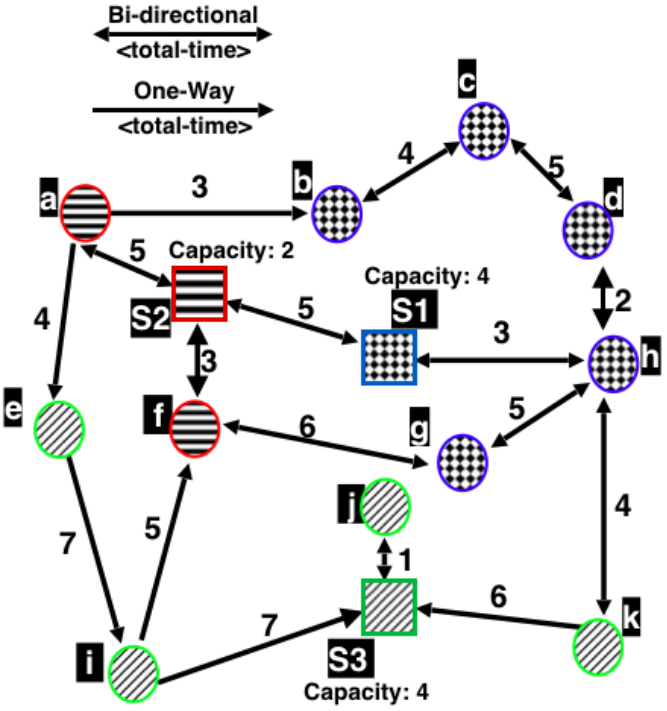

FIGURE 4.2: Network Voronoi Diagram(NVD) for the input graph shown in figure 4.1

Definition 4.6. Penalty cost for a service center ($q_{s_i} \forall\ q_{s_i} \in Q = \{q_{s_1}, ..., q_{s_{n_s}}\}$) denotes the amount of âĂIJextra costâĂİ that must be paid for each new allotment to service center s_i beyond its capacity c_{s_i}. If allotment is done within the capacity of a service center, then no penalty needs to be paid. Examples of penalty cost could be the cost to add additional infrastructure and/or faculty in a school.

Consider again the graph shown in Figure 4.1, assume that s_1, s_2, and s_3 have penalty costs of 20, 5, and 25 units respectively. This means that every new allotment to service center s_1 beyond its capacity (4) will result in an increase in the objective function by 20 units. Similarly, every new allotment, beyond 2 allotments, to s_2 will result in an increase in the objective function by 5 units.

4.2 Input

Given:

1. A transportation network G(V,E), where each edge $e \in E$ has a positive cost.
2. A set of service center $S = \{s_1, ..., s_{n_s}\}$ where $S \subset V$.
3. A set of demand vertices $V_d = \{v_{d_1}, ..., v_{d_{n_{v_d}}}\}$ where $V_d = V - S$.
4. A set of positive integer service center capacities $C = \{c_{s_1}, ..., c_{s_{n_s}}\}$.
5. A set of positive integer service center penalties $Q = \{q_{s_1}, ..., q_{s_{n_s}}\}$.

4.3 Output

A Network Voronoi Diagram which is a partition of the input demand vertices.

4.4 Objective Function

$$\textbf{\textit{Minimize}} \left\{ \sum_{\substack{s_i \in Service \\ Centers}} \left\{ \sum_{\substack{V_{d_j}^{s_i} \in Demand\ vertex \\ alloted\ to\ s_i}} Dist(V_{d_j}^{s_i}, s_i) \right\} + TotalPenalty \right\} \tag{4.2}$$

Difference in the objective function described in Equation 4.1 and Equation 4.2 is the parameter of penalty cost (refer definition 6) represented by ***"Total Penalty"*** in Equation 4.2.

We assume that for every demand vertex in V_d, there exist a path to reach at least one

service center. Also, we consider the case where total capacity of all service centers is lesser than the total demand. Thus, there will certainly be some service center(s) which will get overloaded, hence their penalty cost will be added to the objective function. Apart from the previously mentioned case, there can be two other cases. First, the total capacity is equal to the total demand. And second, the total capacity is greater than the total demand. In both these cases, some service centers can still get overloaded as a demand vertex can choose to forcibly (as it may lead to lower objective function value) go to a nearby service center (which is already full) and pay the penalty instead of going to a far-off "free" service center.

Chapter 5

Proposed Approach

In this chapter we explain the details of the proposed solutions for the problem of LB-NVD. Here, we first provide the key idea of the algorithm and later explain each of the approaches one by one. Lastly, we provide an execution trace of the algorithm with an example.

5.1 Key idea of the algorithm

Consider again Equation 4.2, if it had only the left part, i.e., a double summation over $Dist(V_{d_j}^{s_i}, s_i)$ and no constraints on the capacity of service centers, then it would have been trivial to construct a polynomial time algorithm to get an optimal solution. In the trivial algorithm, we would first compute the shortest distance between all pairs of demand vertices and service centers. Following this, each demand vertex would be assigned to its nearest service center. However, we have two additional aspects to consider in our LBNVD problem. First, the notion of capacity of a service center, and second the notion of overload penalty.

We proposes a synergistic combination of greedy approach and a notion of local re-adjustment to solve the LBNVD problem. First step of this greedy method would be same as the trivial algorithm discussed earlier, i.e., we would compute the shortest distances between all pairs of demand vertices and service centers and thus associate a demand vertex V_{d_j} with its closest service center s_i. Following this, sort the <demand vertex âĂŞ closest s_i> pairs $(V_{d_j}, closest\ s_i)$ in the increasing order of the shortest distance. After this, the algorithm would pick the pair $(V_{d_j}, closest\ s_i)$ (in increasing order of distance) and ***try*** to assign V_{d_j} to s_i. At this stage, one of the following two cases can happen:

Case 1 s_i is not yet full (in terms of capacity) in which case the allotment goes through or,

Case 2 s_i is already full and we need to âĂIJpay a penaltyâĂİ for this assignment.

Given that we perform the allotment in increasing order of distance, case 1 guarantees that we would increase the objective function with the least possible amount. We now describe our approach for case 2.

For addressing case 2, we propose the concept of *local re-adjustment* of the current partially built network voronoi diagram. It is important to note that once a demand vertex V_{d_j} is allotted to a service center s_i it need not be removed from s_i unless it is required during local re-adjustment.

5.1.1 Concept of Local Re-Adjustment of Partial Voronoi Diagram

Figure 5.1 shows a sample transportation network with 5 service centers and 11 demand nodes. Figure 5.2 illustrates a partially constructed network voronoi diagram on the transportation network shown in Figure 5.1. In the figure, the demand vertices which are allotted a service center are filled using the same pattern as of their allotted service center. Here, first few <demand vertex- closest service center> pairs (in increasing order of distance) have already been processed. All these pairs were processed through case 1. s_1,s_2 and s_3 are now full in capacity.

Now consider the case when the pair $<i,\ s_3>$ is being processed. Ideally the demand vertex i should be assigned to service center s_3 as per the nearest service center criteria. Since, s_3 is full we need to consider one of the following options:

1. Associate i to s_3 and pay the penalty of overloading s_3;
   Total increase in objective function: 4+ q_{s_3} (penalty of s_3)

2. Associate i to s_3, but push a border vertex (say d) to another service center. (marked in Figure 5.3);
   Total increase in objective function to associate i to s_3 and then transfer d - to s_1: 4 + (5 - 1) + q_{s_1} (penalty of s_1)
   - to s_2: 4 + (9 - 1) + q_{s_2} (penalty of s_2)
   - to s_4: 4 + (13 - 1) (No penalty since under-loaded)
   - to s_5: 4 + (10 - 1) (No penalty since under-loaded)

3. Associate i to s_3 but push a border vertex f to the neighbouring service center (marked in Figure 5.3);

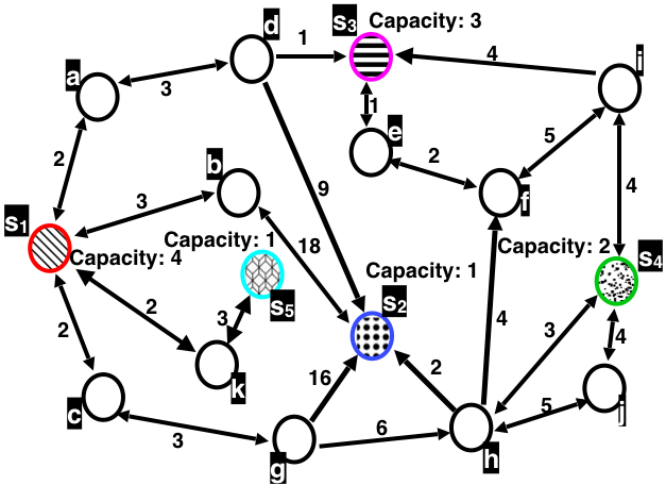

FIGURE 5.1: A sample transportation Network shown.

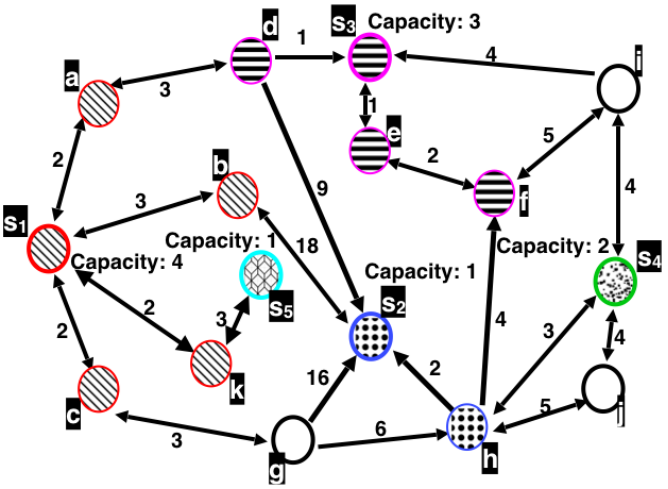

FIGURE 5.2: Partially constructed LBNVD for Network shown in Figure 5.1

Total increase in objective function to associate i to s_3 and then transfer f - to s_1:

4 + (35 - 3) + q_{s_1} (penalty of s_1)

- to s_2: 4 + (14 - 3) + q_{s_2} (penalty of s_2)

- to s_4: 4 + (9 - 3) (No penalty since under-loaded)

- to s_5: 4 + (40 - 3) (No penalty since under-loaded)

The algorithm would evaluate the cost of each of these options (in terms of change (increase) in objective function value) and choose the minimum.

Note that in option (2) and (3), s_1, s_2, s_4, or s_5 could in-turn also consider re-adjusting the load to their own neighbours if they get (or are themselves) overloaded. One such case could be if i was allocated to s_3 and d was pushed s_1. Then service center s_1 will get overloaded by 1. Now, s_1 in-turn can choose between the following two decisions:

1. Accept d and overload itself,

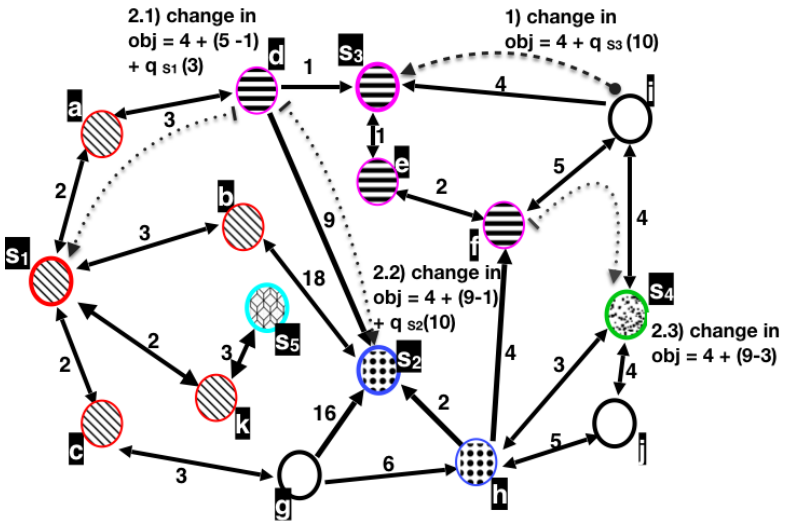

FIGURE 5.3: Illustrating local re-adjustment options for demand node i.

2. Accept d and in-turn push one of its allocated boundary[1] vertices to a neighbouring service center (for e.g., k to s_5).

For simplicity Figure 5.3 does not show further load re-adjustment, we explain these details next.

It is important to note that our proposed approach is greedy in nature and (in some cases) it may not help in attaining the optimal value of objective function. This is because the final output depends on various factors like the order in which the demand vertices are processed, the capacity and penalty cost of each service center.

5.2 Details of the algorithm

This section explains the different data structures used in the proposed approaches. Next, we explain the detailed algorithms and their pseudocode.

5.2.1 Data structures used in the algorithm

In our proposed approach we use the following data structures:

Definition 5.1. Vertex Color A color X_i is associated to each service center s_i. Initially, every demand vertex is uncolored. When a demand vertex is allotted a service center, it gets the color of its service center.

Definition 5.2. Set of Boundary Vertices $B_{s_i} = \{B^1_{s_i}, B^2_{s_i}.., B^m_{s_{n_s}}\}$ for a service center s_i (where $i \in [1, n_s]$) is a set of demand vertices allotted to the service center s_i

[1]Please see Definition 8

where each vertex in B_{s_i} either has an outgoing edge to a vertex of different color or an uncolored vertex.

Definition 5.3. Distance Matrix is a 2-D Array data structure which has the shortest path distance between all pairs of demand vertices and service centers.

Definition 5.4. MinDistance Heap is a binary min-heap data structure. This heap is ordered on the distance between a demand vertex and its closest service center.

5.2.2 Proposed approaches and their details

Our proposed algorithms start by first loading the pre-computed shortest path distances into the *Distance Matrix*. Demand vertices are then put in *MinDistance heap*. Demand vertex to be processed next is picked from the MinDistance heap using the extract-min operation. Initially all vertices are uncolored and the goal of algorithm is to assign a color to each demand vertex, i.e., allocate a service center to each of them. A *while* loop is executed till the heap is not empty. This ensures that all demand vertices are processed.

Let the result of extract-min operation be a demand vertex v_d and its closest service center be s_i. Currently v_d is uncolored and as per the closest distance metric it should go to the service center s_i. However, before allocating a demand vertex to a service center, the algorithm checks if s_i can accommodate the demand or not. If service center s_i has the required capacity, then v_d is allocated to s_i by assigning v_d the same color as of s_i and the objective function is incremented by the distance to reach s_i from v_d. However, if the service center is full or overloaded, we reach the scenario of local re-adjustment. The concept of local re-adjustment is implemented through the following three different approaches as explained next:

1. **Bounded Greedy Hop LoRAL Algorithm:** Continuing from the previous scenario of demand vertex v_d being allotted to its closest service center s_i; if s_i is full in capacity, the algorithm makes one of the two decisions: **(a)** either forcibly overload the service center s_i and pay the penalty cost for overloading; or **(b)** accept v_d vertex and reject a vertex from the set of boundary vertices of s_i (locally re-adjust). The latter will keep the load of s_i same as before.

For option **(b)**, we calculate the sum of the following three terms for every pair of boundary vertex $b^i_{s_i} \in B_{s_i}$ and service centers $s_j \in S - \{s_i\}$: (1) shortest distance

between $b^i_{s_i}$ in the boundary set of s_i (B_{s_i}) and service center s_j ($s_j \in S - \{s_i\}$), (2) penalty cost of the service center s_j (if any) and, (3) negative distance between boundary vertex b_{s_i} and service center s_i. Following this we choose the pair of $b^i_{s_i}$ and s_j which gives us the lowest sum. We call the boundary vertex in the selected pair as the *best boundary* vertex v_{bbs_i}.

Now let us consider the costs of the two decisions (for s_i) by calculating:

(1) Cost of overloading a service center which is sum of distance between v_d and s_i (Dist(v_d,s_i)) and the penalty cost for s_i (q_{s_i}),

(2) Cost of load re-adjustment as the sum of following three terms: a) distance of a best boundary vertex $v_{bb_{s_i}}$ of s_i to another service center (say s_j) (Dist($v_{bb_{s_i}}$,s_j)), b) penalty cost of the new service center s_j (q_{s_j}, in case s_j is full as well) and c) negative of distance between $v_{bb_{s_i}}$ and s_i (Dist($v_{bb_{s_i}}$,s_i)).

The goal of LBNVD problem is to reach a least possible objective function. Overloading a service center will increase the objective function with the cost of overload but in case there is another service center then local re-adjustment may lead to a smaller increase in objective, and thus may be preferred over forced overloading (option(1)).

Bounded Greedy Hop LoRAL algorithm compares the cost of the previously described option (a) and option (b) and then chooses the option which leads to lower increase in the objective function value. It is important to note that local readjustment may cascade. The rejected vertex is now forwarded to service center s_j. s_j may accept this vertex or it may in-turn try to further push a vertex from its boundary vertices leading to a cascade of local re-adjustments. In this algorithm we limit the number of readjustments caused during allotment of one uncolored vertex (v_d) by a parameter $\boldsymbol{l}$.

2. **Unbounded Greedy Hop LoRAL Algorithm:** In bounded greedy hop LoRAL algorithm, number of local re-adjustments are limited by a parameter $\boldsymbol{l}$. In the unbounded version of LoRAL algorithm, this limitation is essentially loosened by allowing *unbounded (till it reaches a local minima)* number of cascade steps instead of l. We allow local re-adjustments to continue till we reach a local minima. In practice, the algorithm makes a note of objective function before the local re-adjustment. Further re-adjustment(s) (pushing a boundary vertex) is executed only if there is an improvement in the objective function.

Note that both the bounded and unbounded greedy hop algorithm implicitly creates a "path" of local re-adjustments. Nodes of this path are the service centers and edges

denote a transfer of the boundary vertex from one service center to another. Starting from a service center s_i (as the starting node of this path), both bounded and unbounded greedy hop keep choosing the best boundary vertex (to transfer to a service center s_j) creating the "next edge" on this implicit path.

Bounded greedy hop limits the length of the implicit path to l, whereas unbounded greedy hop continues local re-adjustment until a minima (local) is reached.

3. **Best of k paths LoRAL Algorithm:** The third variant of LoRAL described next is slightly different from previously stated algorithms. Instead of choosing "the best boundary" vertex and committing to it, it simulates implicit paths of transfers for top k boundary vertices of the current service node s_i under consideration. Note that for efficiency purposes this top k solution is limited at only the "starting node" of this path. At the subsequent "nodes", we select the best boundary vertex similar to bounded or unbounded greedy hop.

Algorithm 4 Best of k paths LoRAL Algorithm

Input: (a) Transportation Network G, (b) Set of Service Centers S, (c) Set of Demand Vertices V_d, (d) Set of Service Center Capacities C, (e) Set of Service Center Penalties Q and, (f) Number of Paths k, (g) Length of path $l \leftarrow$ set to number of service centers

Output: LBNVD with Objective Function value Δ

Load distance matrix with the shortest path distance between every demand vertex in V_d and every service center in S.
Load MinDistance heap with all <demand vertex-closest service center> pairs.
while MinDistance heap is not empty **do**
  Load $<v_d, s_i>$ from the result of extract-min operation of MinDistance heap.
  if s_i has vacancy **then**
    (Allocate v_d to s_i) Set $v_{d_{color_i}} \leftarrow s_{color_i}$, $c_{s_i} \leftarrow c_{s_i} - 1$, and $\Delta = \Delta + d(v_d, s_i)$
  else
    Find the set of top k boundary vertices (v_{bbs_i}) and their corresponding service center (s_j) for re-adjustment.
    for each i in k **do**
      Initialize cost of $path^i$: $\delta_2^i = 0$.
      while $path^i$ doesn't terminates or length of $path^i$ is less than l **do**
        if s_j has vacancy **then**
          Increment δ_2^i by ($d(v_{bbs_i}, s_j)$ - $d(v_{bbs_i}, s_{j-1})$) and set the flags to terminate the $path^i$
        else
          Set cost of overloading s_j: $\beta_1^i \leftarrow d(v_{bbs_i}, s_j) - d(v_{bbs_i}, s_{j-1}) + q_{s_j}$
          Find the best boundary vertex (v_{bbs_j}) of s_j and its corresponding service center (s_{j+1}) for re-adjustment
          if s_{j+1} has been visited earlier **then**
            Increment δ_2^i by β_1^i and set the flags to terminate the $path^i$
          else

Cost of local re-adjustment for s_j: $\beta_2^i \leftarrow d(v_{bbs_i}, s_j) - d(v_{bbs_j}, s_j) + d(v_{bbs_j}, s_{j+1}) + q_{s_{j+1}}$ (if s_{j+1} gets overloaded.)
if $\beta_1^i < \beta_2^i$ **then**
Increment δ_2^i by β_1^i and set the flags to terminate the $path^i$
else
Increment δ_2^i by β_2^i and set $v_{bbs_i} \leftarrow v_{bbs_j}$, $s_j \leftarrow s_{j+1}$, $l \leftarrow l + 1$
end if
end if
end if
end while
Store $path^i$ and the cost of $path^i$.
end for
Set cost of overload: $\delta_1 \leftarrow d(v_d, s_i) + q_{s_i}$
Set cost of transfer: $\delta_2 \leftarrow$ minimum value of $cost\ of\ path^i\ \forall i \in [1, k]$
if $\delta_1 < \delta_2$ **then**
(Allocate v_d to s_i with penalty) set $v_{d_{color_i}} \leftarrow s_{color_i}$, $c_{s_i} \leftarrow c_{s_i} - 1$, and $\Delta = \Delta + \delta_1$.
else
(Allocate v_d to s_i with load transfers) set for each p in minimum path: set $v_{d_{color_p}} \leftarrow s_{color_p}$, $c_{s_p} \leftarrow c_{s_p} - 1$, set $\Delta = \Delta + \delta_2$.
end if
end if
end while

Once all the paths are simulated and their costs are evaluated, the path which generates the minimum increase in objective function is chosen. Finally, the algorithm chooses between performing overload at s_i or local re-adjustments as per the minimum cost path. Algorithm 4 shows the pseudocode for best of k paths algorithm.

5.3 Execution Trace

Figure 5.1 shows a sample input graph with 5 service centers and 11 demand vertices to our proposed LoRAL Algorithm. The capacity of the service center s_1, s_2, s_3, s_4, and s_5 are 4, 1, 3, 2 and 1 respectively and the penalty costs are 3, 10, 10, 15 and 15 respectively. Initial set of assignments, as shown in Figure 5.2, were performed without any local re-adjustments (as all service centers were still under-loaded) in the increasing order of distances. At this stage, the current value of objective function is 16. We now perform assignment of demand vertex i to its closest service center s_3. Here, s_3 is full, and hence we reach the scenario of local re-adjustment. Cost to overload the service center s_3 from the assignment of demand vertex i is 14, i.e. the sum of distance between i and s_3 (4) and penalty cost of s_3 (10).

Bounded greedy hop LoRAL algorithm (shown in figure 5.4) finds a best boundary vertex from the set of boundary vertices of s_3 (i.e. d and f). Increase in the objective function by accepting demand vertex i and in-turn pushing out f to s_4 (cost to send f to s_4 is minimum among all other service centers) is 10 which is the sum of distance between i and s_3 (4), distance between f and s_4 (9), and negative of distance between f and s_3 (3). Whereas, increase in the objective function value by accepting the demand vertex i and in-turn pushing out d to s_1 (cost to send d to s_1 is minimum among all other service centers) is 11 which is the sum of distance between i and s_3 (4), distance between d and s_1 (5), negative of distance between d and s_3 (1) and penalty cost of s_1 (3, since s_1 is full). From the very nature of bounded greedy hop algorithm, f is chosen as best boundary vertex.

Assigning i to s_3 and transferring the best boundary vertex f to $s4$ increases the objective function by 10 as opposed to forceful overloading s_3 with the assignment of i (14). Hence local re-adjustment is carried out. One may note that unbounded algorithm will also perform the same local re-adjustment. This is because s_4 is under-loaded and does not perform any further local re-adjustments.

Figure 5.5 shows the trace for best of k paths algorithm. When we reach the assignment of i to s_3, best of k paths algorithm considers k best boundary vertices instead of single best boundary vertex. In the given graph, two paths can be seen as:

Option 1: Assigning i to s_3, then pushing out f from s_3 to s_4. Path cost: 10

Option 2: Assigning i to s_3, then pushing out d from s_3 to s_1 and further pushing out k from s_1 to s_5. Path Cost: 9

Notice that neither bounded nor unbounded greedy hop algorithm considers option 2, since demand vertex d is not the best boundary vertex. Best of k paths algorithm performs assignments as per option 2, and increases the objective function by the least amount (9 as opposed to 10 by bounded/unbounded algorithm).

Figure 5.6 shows the final assignments by bounded and unbounded greedy hop algorithm (they are same in this example), and figure 5.7 shows the assignments by best of k paths algorithm. Path(s) considered by each of the algorithm during the assignment of last demand vertex g are highlighted. One may notice that best of k paths algorithm resulted in a reduced objective function value as compared to bounded(and unbounded) greedy hop algorithm.

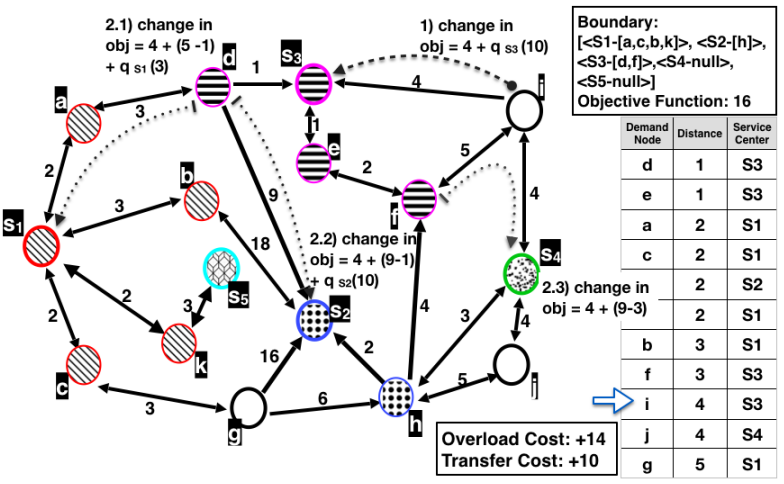

FIGURE 5.4: Execution Trace: Local Re-adjustments considered by Bounded, Unbounded LoRAL Algorithm

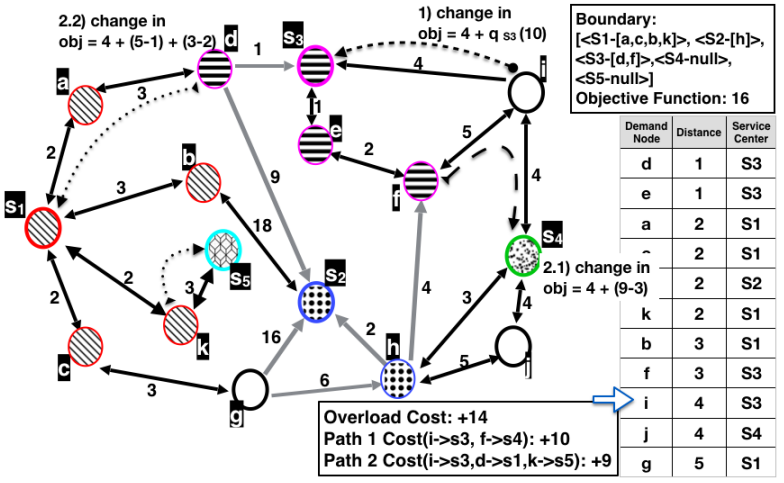

FIGURE 5.5: Execution Trace: Paths considered by Best of k Path Algorithm

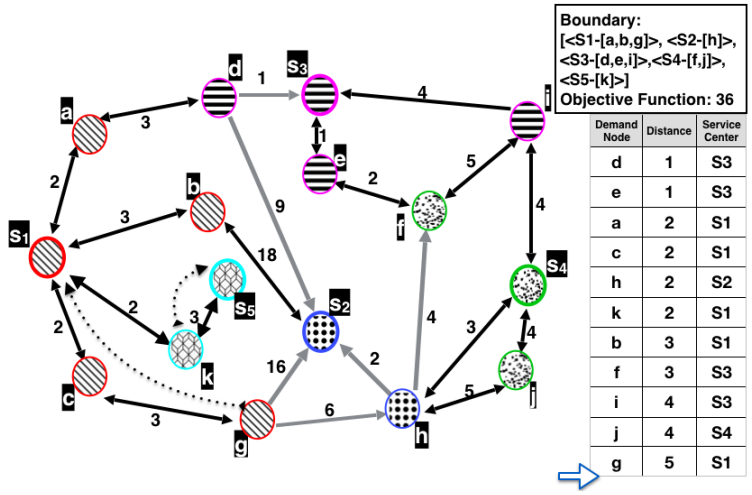

FIGURE 5.6: Execution Trace: Final assignments by Bounded, Unbounded LoRAL Algorithm

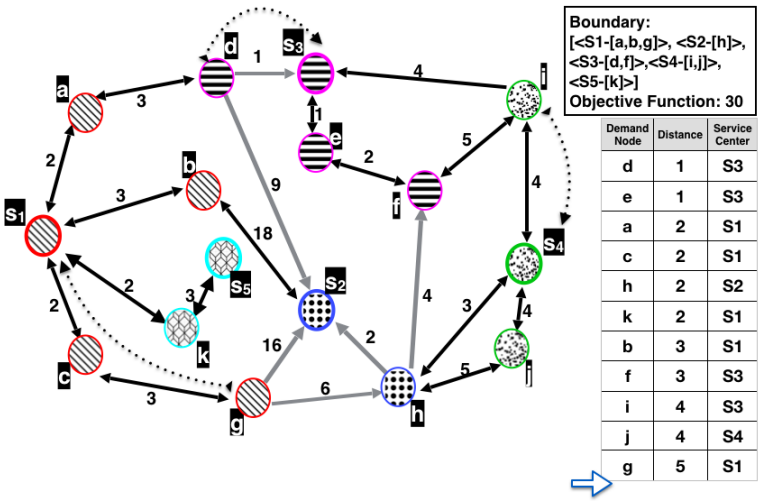

FIGURE 5.7: Execution Trace: Final assignments by Best of k Path Algorithm

Chapter 6

Experimental Evaluation

This chapter provides the details of experimental evaluations done as part of this thesis.

6.1 Experiment Results

6.1.1 Experimental setup:

In our experiments, we used the road network dataset of New Delhi, India. This dataset was taken from OpenStreetMap [11] and was converted into a directed graph. Some vertices (chosen randomly) of this graph became the service centers and other vertices were designated as the demand vertices (each with unit demand). All algorithms were implemented in Java 1.7 and the experiments were carried out on a Intel(R) Xeon(R) CPU machine (2.50GHz) with 96GB of RAM and Cent OS 6.5. We compared the performance of our three LoRAL algorithms against the optimal algorithm (Min-Cost Bipartite Matching algorithm [10]). In our experiments we measured run-time and final value of objective function as different parameters were varied.

6.1.2 Experiment 1: Comparing our algorithms with the optimal algorithm:

Figure 6.1 and 6.2 shows the results of this experiment. In this experiment, the number of service centers in each network was set according to the following scheme: 1 service center for every 400 demand vertices (denoted as service center ratio 400:1). Capacities of all service centers were chosen randomly (integers) from the range 250-350, and penalty cost was a random integer from the range 50-80.

Our experiment showed that LoRAL algorithms were much faster than the optimal algorithm while maintaining comparable values of the final objective function. Among the three LoRAL algorithms, best of k paths algorithm was closest to the optimal algorithm in terms of the final value of the objective function at the termination. The optimal algorithm did not scale up with the increase in the size of the input network. For this reason, the optimal algorithm was excluded from further experiments where networks of size 65000 vertices were used.

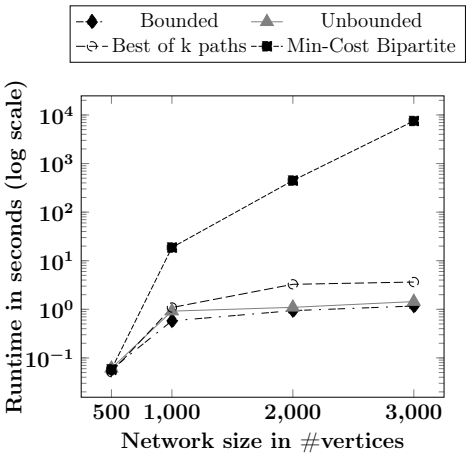

FIGURE 6.1: Exp. 1 - Run-time analysis.

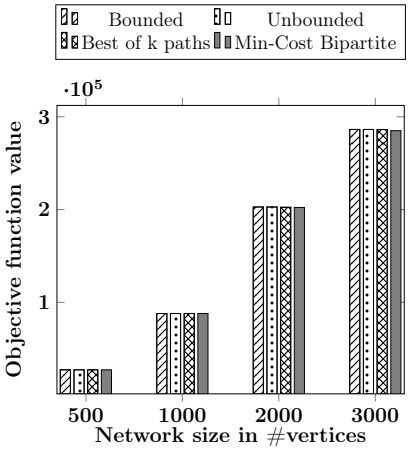

FIGURE 6.2: Exp. 1 - Objective func. analysis.

6.1.3 Experiment 2: Effect of number of service centers:

Figure 6.3, 6.4 and 6.5 shows the result of this experiment. In this experiment, the number of service centers was varied by changing the ratio of the # demand vertices to the # service centers. A smaller value of this ratio indicates a larger number of service centers. Capacities of all service centers were chosen randomly (integers) from the range 250-350, and penalty cost of each service center was a random integer from the range 50-80. We ran the best of k paths algorithm by upper bounding the values of k (k_{max}) to: (1) number of demand vertices and, (2) number of demand vertices/2, this is denoted as "best of k/2 paths" in the plots. By upper bounding we mean that each service center would explore min(its #boundary vertices, k_{max}) number of paths. Our experiments showed that the execution time of all LoRAL algorithms decreases as the number of service centers increases. This is because more number of service centers are getting available, so fewer local re-adjustments. Also, best of k paths algorithm took more time than the other two variants because it has a greater search space.

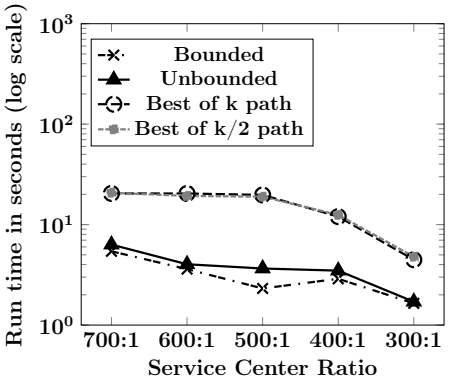

FIGURE 6.3: Effect of number of service center on graph with 11399 vertices and 24942 edges

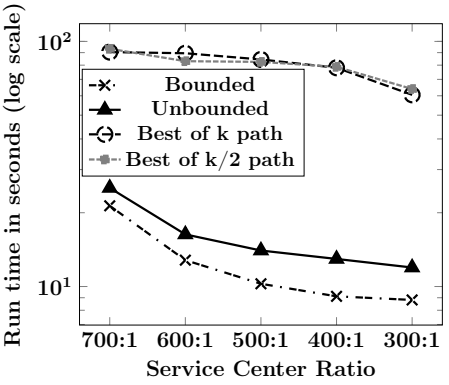

FIGURE 6.4: Effect of number of service center on graph with 23839 vertices and 51794 edges

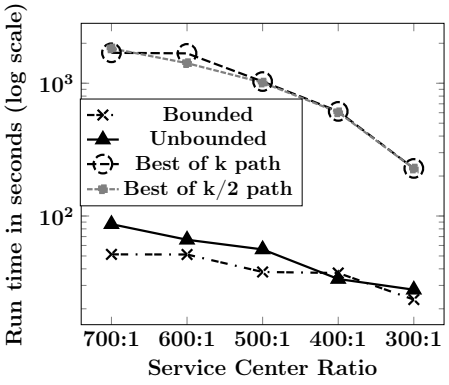

FIGURE 6.5: Effect of number of service center on graph with 65902 vertices and 144410 edges

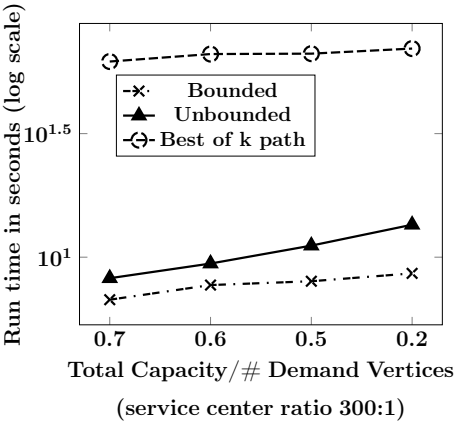

FIGURE 6.6: Effect of total capacity of service centers on graph with 23.8K vertices

6.1.4 Experiment 3: Effect of total capacity of service centers:

In this experiment, penalty costs were in the range of 50-80 units and the number of service centers were according to the ratio of 300:1 (Figure 6.6 and 6.7) and 500:1 (Figure 6.8). Our experiments showed that as the parameter total capacity/# demand vertices decreased, runtime increased as more local re-adjustments were made. Figure 6.9 shows that value of the objective function increased as the parameter total capacity/# demand vertices decreased since more penalty is being paid.

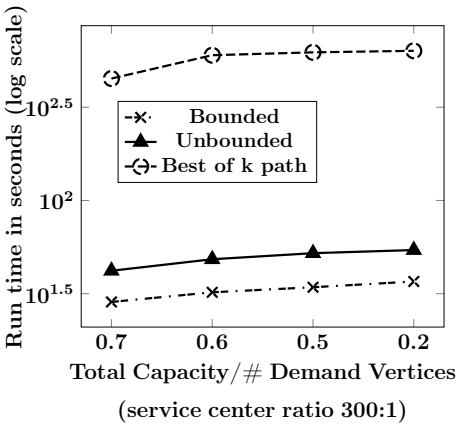

FIGURE 6.7: Effect of total capacity of service centers on graph with 65.9K vertices

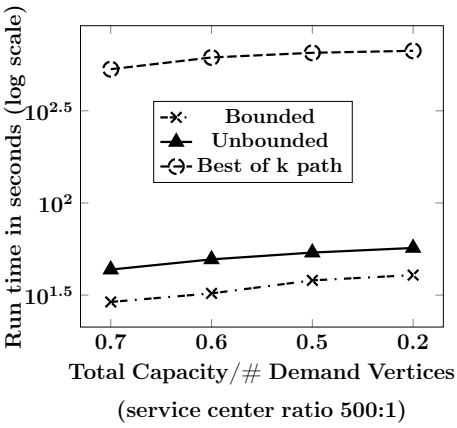

FIGURE 6.8: Effect of total capacity of service centers on graph with 65.9K vertices

Capacity Ratio	Bounded	Unbounded	Best of k paths
0.7	8688072	8679187	8664776
0.6	8887163	8881991	8871237
0.5	9113295	9109435	9098743
0.2	10127435	10126311	10123650

FIGURE 6.9: Effect of total capacity of service centers on graph with vertices (Objective Function)

6.1.5 Experiment 4: Effect of penalty costs:

In this experiment, we fixed the ratio total-capacity/#demand-vertices to 0.6 and the number of service centers according to the ratios 300:1 (Figure 6.10 and 6.11) and 500:1 (Figure **??**). These figures shows that as the penalty costs increased, runtime increased as more number of local re-adjustments were explored. Figure 6.13 shows the experiment with service center ratio 500:1. In the figure, one can notice that the value of the objective function increased with increasing penalty costs.

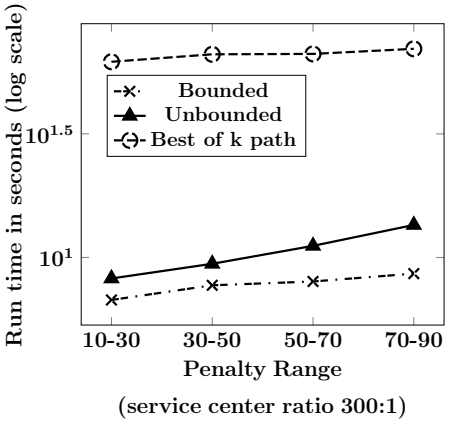

FIGURE 6.10: Effect of penalty costs on graph with 23.8K vertices

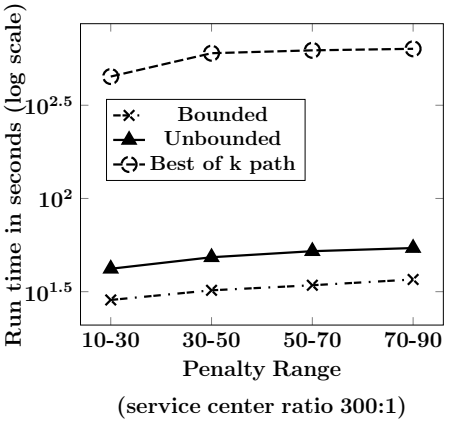

FIGURE 6.11: Effect of penalty costs on graph with 65.9K vertices

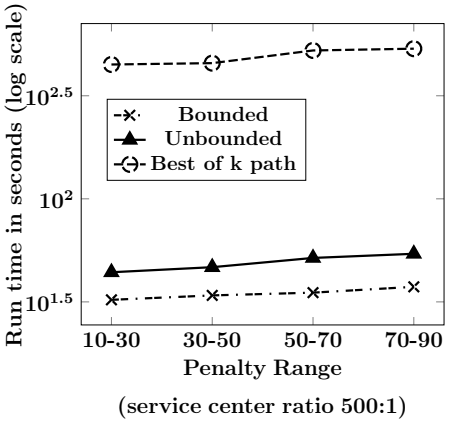

FIGURE 6.12: Effect of penalty costs on graph with 65.9K vertices

Penalty Range	Bounded	Unbounded	Best of k paths
10-30	7798550	7798225	7798106
30-50	8646181	8644853	8643786
50-70	9512618	9511183	9505962
70-90	10371867	10368714	10361819

FIGURE 6.13: Effect of penalty costs on graph with 65.9K vertices (Objective Function)

6.1.6 Acknowledgement

I would like to thank and express my gratitude to Kapish Malik for contributing all the results performed on Best of k path Algorithm and optimal algorithm - Min-cost bipartite matching algorithm.

Chapter 7

Theoretical and Analytical Analysis

7.1 Theoretical Analysis

Axiom 7.1. *There exists a global optimal solution to the problem of LBNVD.*

Lemma 7.2. *All variants of the LoRAL algorithm guarantee correctness. We say that an algorithm is correct if each demand vertex is assigned only one service center.*

Proof. Once assigned a service center, a demand vertex remains in the allotment set of the service center unless it is needed during local re-adjustment. If a demand vertex is locally re-adjusted, it is removed from its previously assigned service center and added to the allotment set of other service center. Thus, at any stage a demand vertex may be assigned to only one service center. □

Lemma 7.3. *All variants of the LoRAL algorithm guarantee completeness. We say that an algorithm is complete if every demand vertex is allotted a service center.*

Proof. All demand vertices are processed one at a time using the extract-min operation of MinDistance heap in the increasing order of their distances. Each variant of the LoRAL algorithm will terminate *iff* MinDistance heap is empty i.e. every demand vertex is allotted a service center. □

Lemma 7.4. *All variants of LoRAL algorithm guarantee termination.*

Proof. All variants of local re-adjustment based approaches assign demand vertices to service centers in the increasing order of distances. In case, a service center is full, local re-adjustment(s) may take place. We now provide the termination proof for the three approaches of local re-adjustments of LoRAL Algorithm:

1) *Bounded Greedy Hop Algorithm:* Bounded greedy hop algorithm has a bounded parameter l for number of local re-adjustments. Parameter l is non-negative and finite. Hence, the algorithm will terminate in a finite number of steps.
2) *Unbounded Greedy Hop Algorithm:* Search for the next local re-adjustment terminates in the unbounded greedy hop algorithm when the change in objective function is not significant. From axiom 7.1, we know that there exists a global minima for the problem. Therefore, in worst case, the algorithm may attempt to reach the global minima by undertaking many local re-adjustments, but it would still terminate as we cannot locally re-adjust beyond a global minima.
2) *Best of k Paths Algorithm:* In order to locally re-adjust demand vertices, best of k paths algorithm finds the top k boundary vertices from an overloaded service center. Each of the k boundary vertices initiate a simulated path of local re-adjustments. Parameter k is non negative, fixed and finite. The maximum length of the simulated path in the algorithm is bounded by the number of service centers (at most) given in the input. Therefore, algorithm is guaranteed to terminate in finite number of steps. □

7.2 Analytical Analysis

The input graph G(V,E) with V vertices and E edges contains n_d demand vertices and n_s service centers, where $n_s << n_d$.

7.2.1 Space Complexity:

We store the input graph of LBNVD problem in the form of an adjacency list which takes $O(V + E)$ space. All three variants of LoRAL algorithms use four main data structures: (1) vertex colors, which takes $O(n_d + n_s)$ space, (2) boundary vertices, which can take $O(n_d)$ space, (3) distance matrix, which takes $O(n_d \times n_s)$ space, and (4) MinDistance heap which takes $O(n_d)$ space. Rest of the variables like objective function value, intermediate costs, etc. trivially take $O(1)$ space. Thus, the total space complexity of all three variants of LoRAL algorithms is $O(V + E)$ + $O(n_d + n_s)$+ $O(n_d \times n_s)$ + $O(n_d)$.

7.2.2 Time Complexity:

Time complexity of unbounded greedy hop cannot be mathematically bounded as it may in principle attempt to reach a global minima. Complexity of other two LoRAL algorithms is given next. Both of them share initial few steps of execution, these are

steps 1-7 in Algorithm 1. In step 1, the shortest path distance between every demand vertex and service center is computed using floyd-warshall algorithm [12]. This step takes $O((n_d + n_s)^3)$ for floyd-warshall (this is pre-computed in our experiments) and takes $O(n_d+n_s)$ time to load the distance matrix. Next, the MinDistance heap is loaded which requires finding the closest service center for each demand vertex and thus takes $O(n_d \times n_s)$. In Step 3, the algorithm enters into a *while* loop which runs for each of the n_d demand vertices. In Step 4, each extract-min operation takes $O(log(n_d))$ time. Steps 5-7 take $O(1)$ each. In case of local re-adjustment, finding the best boundary vertex can take $O(n_d \times n_s)$ time (in worst case).

Next set of steps (after Step 7 but inside the loop on line 3) are different for both the LoRAL algorithms. Bounded greedy hop LoRAL algorithm would at most take l local re-adjustments. Cost of one local re-adjustment is $O(n_d \times n_s)$. Therefore, the overall cost of bounded greedy hop algorithm is $O(n_d+n_s)^3+O(n_d+n_s)+O(n_d \times l \times n_d \times n_s \times +n_d \times log(n_d))$ time. In best of k paths, we simulate k paths of re-adjustments, each of length n_s (at most). Therefore, total cost of best of k paths algorithm is $O(n_d + n_s)^3+O(n_d + n_s)+O(n_d \times k \times n_s \times n_d \times n_s \times +n_d \times log(n_d))$. If n_s is really less then we can drop that term in the above expressions.

Chapter 8

Conclusion and Future Work

In this thesis, we proposed novel local re-adjustment based approaches for the problem of Load Balanced Network Voronoi Diagram, i.e. a) Bounded Greedy Hop LoRAL Algorithm, b) Unbounded Greedy Hop LoRAL Algorithm and c) Best of K Paths LoRAL Algorithm. We evaluated our approaches both theoretically and experimentally. We took the Data set of New Delhi road network using open street maps and evaluated the performance of our approaches on the same. Our experiments indicated that LoRAL algorithms were significantly more scalable than the optimal algorithm (as provided in the literature, i.e. min-cost bipartite matching algorithm). The results obtained as a result of LoRAL algorithms were close to the optimal final objective function value. As part of the future work, we would like to work on other generalizations of network voronoi diagrams. We would also like to explore the optimal modification for best of k path LoRAL algorithm for all paths such that k = no of demand vertices, hence provide a global optimal solution.

Appendix A

Reducing LBNVD problem to Min-Cost Bipartite Matching

In this appendix, we provide the reduction of LBNVD problem to Min-Cost Bipartite Matching. This reduction was described in [10].

Consider a graph G(V, E) with V vertices and E edges. Consider the graph contains n_d no of demand vertices and n_s number of service centers. Set of demand vertex is defined as V_d = { v_{d_1} , v_{d_2} ,... $v_{d_{n_d}}$ } and set of service centers is defined as S = { s_1 , s_2 ,... s_{n_s} }.Then in order to reduce the problem of LBNVD to min-cost bipartite matching algortihm, the following steps are required:

Steps to reduce LBNVD problem to Min-Cost Bipartite Matching algorithm
Step 1: Create a bipartite graph by splitting the demand vertices and service centers into two sets.
Step 2: Create n_d copies of each of service center for the first set of the bipartite graph.
Step 3: Make the number of vertices in both sets equal by putting n_d demand vertices and creating (n_s-1) dummy vertices for the second set of bipartite graph.
Step 4: Create edge between every vertex in the second set to every vertex in the first set with the following cost of edge defined as follows:

1. **If a vertex in second set is not a dummy vertex:** Edge cost (vertex, j_{th} copy of s_i) = ShortestDistance(vertex, j_{th} copy of s_i) + penality of s_i (penalty term is added only if $j >$ capacity of s_i.

2. **If a vertex in second set is a dummy vertex:** Edge cost (vertex, j_{th} copy of s_i) = Infinite.

Step 5: Find a perfect matching (a matching in which every vertex of the graph is incident on only one of the edges of matching) of minimum weight (or cost) or this bipartite graph.

The result of step 5 will be the solution to the problem of LBNVD. One may note that we would have to remove the cost contributed by the dummy vertices to compute the value of final objective function for the problem of LBNVD.
Figure A.1 shows a sample graph with 7 vertices. This graph contains 5 demand vertices and 2 service centers. Service center S1 has capacity of 2 and penalty cost 5 units, and service center S2 has capacity of 2 and penalty cost 15 units.

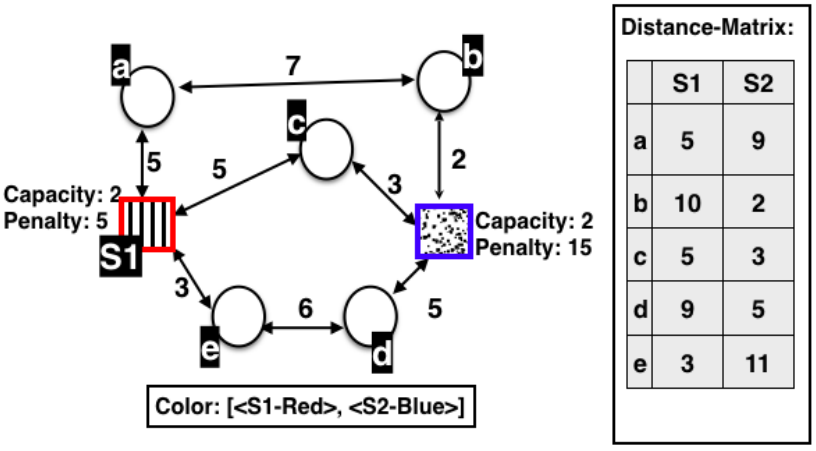

FIGURE A.1: A sample graph with 7 vertices (5 demand vertices and 2 service centers) and their shortest path distances.

Figure A.2 shows a bipartite graph created for the example shown in figure A.1. For simplicity edges for only one demand vertex and one dummy vertex is shown.

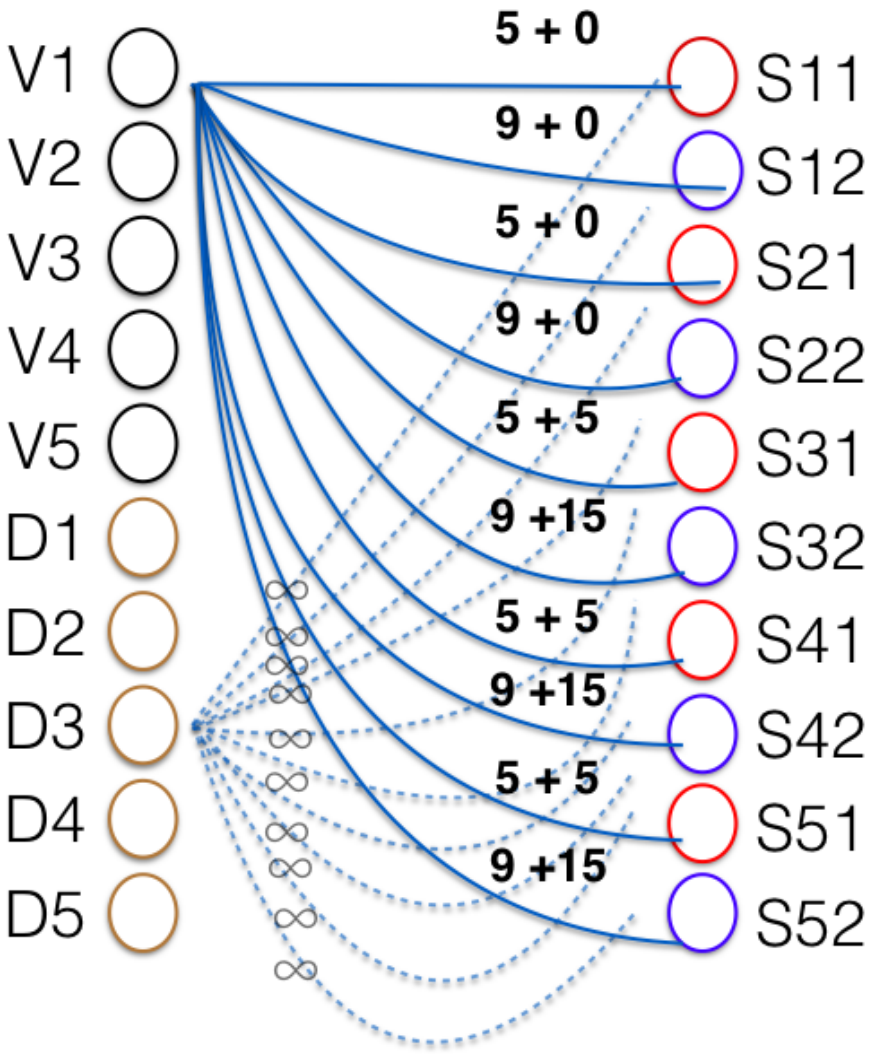

FIGURE A.2: A sample graph with 7 vertices (5 demand vertices and 2 service centers) and their shortest path distances.

A perfect matching for this graph would be the solution to LBNVD problem.

Bibliography

[1] A. Okabe and et al. Generalized network voronoi diagrams: Concepts, computational methods, and applications. *Intl. Journal of GIS*, 22(9):965–994, 2008.

[2] Ugur Demiryurek and Cyrus Shahabi. *Indexing Network Voronoi Diagrams*, pages 526–543. Springer Berlin Heidelberg, 2012.

[3] K. Yang and et al. Capacity-constrained network-voronoi diagram. *IEEE Trans. on Knowledge and Data Engineering*, 27(11):2919–2932, Nov 2015.

[4] KwangSoo Yang and et al. Capacity-constrained network-voronoi diagram: A summary of results. In *proc. of SSTD 2013*, pages 56–73.

[5] Aggarwal and et al. A 3-approximation algorithm for the facility location problem with uniform capacities. *Math. Program.*, 141(1-2):527–547, 2013. doi: 10.1007/s10107-012-0565-4. URL http://dx.doi.org/10.1007/s10107-012-0565-4.

[6] Ali Diabat. A capacitated facility location and inventory management problem with single sourcing. *Optimization Letters*, 10(7):1577–1592, 2016. doi: 10.1007/s11590-015-0950-z. URL http://dx.doi.org/10.1007/s11590-015-0950-z.

[7] Sanjay Melkote and Mark S. Daskin. Capacitated facility location/network design problems. *European Journal of Operational Research*, 129(3):481–495, 2001. doi: 10.1016/S0377-2217(99)00464-6. URL http://dx.doi.org/10.1016/S0377-2217(99)00464-6.

[8] Reza Zanjirani Farahani, Maryam SteadieSeifi, and Nasrin Asgari. Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, 34(7):1689 – 1709, 2010. ISSN 0307-904X. doi: http://dx.doi.org/10.1016/j.apm.2009.10.005.

[9] V. Arya et al. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004. doi: 10.1137/S0097539702416402. URL http://dx.doi.org/10.1137/S0097539702416402.

[10] Edward Bortnikov, Samir Khuller, Jian Li, Yishay Mansour, and Joseph Seffi Naor. The load-distance balancing problem. *Networks*, 59(1):22–29, 2012. ISSN 1097-0037. doi: 10.1002/net.20477. URL http://dx.doi.org/10.1002/net.20477.

[11] Wikipedia. Open street maps foundation. https://en.wikipedia.org/wiki/OpenStreetMap_Foundation, .

[12] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

[13] Rolf H. Möhring and et al. Partitioning graphs to speedup Dijkstra algorithm. *J. Exp. Algorithmics*, 11, February 2007.

[14] A.V. Goldberg and R.F. Werneck. Computing point-to-point shortest paths from external memory. In *Proc. of ALENEX 2005*, pages 26–40.

[15] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Better landmarks within reach. In *Proc. of WEA'07*, pages 38–51. Springer-Verlag, 2007.

[16] Robert Geisberger and et al. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proc. of WEA'08*, pages 319–333. Springer-Verlag, 2008.

[17] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *Proc. of ESA'05*, pages 568–579. Springer-Verlag, 2005.

[18] H. Bast and et al. In transit to constant time shortest-path queries in road networks. In *Workshop on Algorithm Engineering and Experiments*, pages 46–59, 2007.

[19] H. Bast and et al. Fast routing in road networks with transit nodes. *Science*, 316 (5824):566–566, 2007.

[20] Jing and et al.. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Trans. on KDE*, 10(3):409–432, 1998.

[21] Wikipedia. Floyd warshall algorithm. https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm, .

[22] Wikipedia. Heap (data structure). https://en.wikipedia.org/wiki/Heap_(data_structure), .