A Combinatorial Algorithm for Fault Tolerant k-Median Clustering on a Line Metric

Authors
Nihesh Anderson - 2016059
Kushagra Arora - 2015049

BTP report submitted in partial fulfillment of the requirements for the Degree of B.Tech. in Computer Science & Engineering

on April 15, 2019

BTP Track: Research

BTP Advisor
Dr. Rajiv Raman
Dr Syamantak Das

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

We hereby declare that the work presented in the report entitled **'A Combinatorial Algorithm for Fault Tolerant k-Median Clustering on a Line Metric'** submitted by us for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Rajiv Raman** and **Dr. Syamantak Das**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

....	**.............................**	**Place & Date: ....**
Nihesh Anderson	**Kushagra Arora**	

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

....	**.............................**	**Place & Date: ....**
Dr. Rajiv Raman	**Dr. Syamantak Das**	

Abstract

Given a metric space X, a set of clients $C \subseteq X$, and a set of facilities $F \subseteq X$, such that $|X| = |F \bigcup C|$. the fault tolerant k-median problem (FTM) requires to open k facilities in F so as to minimize the sum of distances from each client to r nearest open facilities. Fault tolerant facility location (FTFL) allows opening of variable number of facilities, with an opening cost. FTFL and FTM clustering are known NP Hard problems, in any general metric space. Various approximation algorithms have been proposed to solve the problem in polynomial time.

In this work, we consider the input space to be a line metric. M. Hajiaghayi et al. have proposed an LP based technique to solve FTM in polynomial time. We propose the first combinatorial algorithm to solve FTFL in polynomial time. This paper talks about the structure theorem that clearly defines the structure of the underlying solution. The structure theorem is exploited to build a min cost max flow model to solve FTFL on a line metric. The eventual goal is to find a combinatorial polynomial time solution to FTM on a line metric and extend it to tree metric space.

Keywords: Facility Location, k-Median, Combinatorial Optimization

Acknowledgments

We are deeply grateful to Dr. Rajiv Raman and Dr. Syamantak Das for giving us an opportunity to work with them and introducing us to the problem. They have been guiding us constantly with their invaluable suggestions on how to approach the research problem.

Contents

Chapter 1

Introduction

Fault tolerant facility location falls under the class of combinatorial optimization problems. Formally, it is defined as follows:

Consider a metric space X, a set of clients $C \subseteq X$, and a set of facilities $F \subseteq X$, such that $|X| = |F \bigcup C|$. There is a requirement function $R : C \to \mathbb{Z}^+$ that maps each client $c \in C$ to the number of facilities that should serve it, and a function *open* : $F \to \mathbb{R}$ that maps each facility to an opening cost. The serving cost $D(c, F_{open})$ is the total cost of connecting a client $c \in C$ to $R(c)$ nearest facilities in $F_{open} \subseteq F$. The objective function is formulated as

$$\operatorname*{arg\,min}_{F_{open} \subseteq F} \left(\sum_{c \in C} D(c, F_{open}) + \sum_{f \in F_{open}} \text{open}(f) \right)$$

Fault tolerant k-median is a constrained version of facility location where there is no opening cost, however, at most k facilities can be opened. More formally, the objective function is formulated as

$$\operatorname*{arg\,min}_{F_{open} \subseteq F, |F_{open}| \leq k} \left(\sum_{c \in C} D(c, F_{open}) \right)$$

It is known that the k-median and fault tolerant clustering problems are NP Hard. In this work, we restrict the input domain to a line metric, i.e., $X \subseteq \mathbb{R}^1$. In 2016, M. Hajiaghayi et al. have shown that there is a linear programming based technique to solve the fault tolerant k-median problem in polynomial time [1].

In this work, we present a polynomial time combinatorial solution using minimum cost maximum flow model to the fault tolerant facility location problem with **uniform requirement** $R(c) = r$ $\forall c \in C$. The motivation behind finding a combinatorial solution is to extend the analysis to higher dimensional metric spaces. Chapter 3 and onwards, we consider only uniform requirement case. All lemmas and theorems assume that $R(c) = r$ $\forall c \in C$.

Chapter 2

Literature Review

2.1 Related Work

The fault tolerant k-median problem has a constant factor approximation algorithm in metric spaces. [1] It uses linear programming rounding to obtain an approximation ratio of 93. However, over the real metric, it can be proven that there exists a laminar family such that the constraint matrix of the LP becomes unimodular. Thus, the paper provides an LP approach to solve FTM on a real line in polynomial time. The proposed linear program has $O(|F||C|)$ constraints and $O(|F||C|)$ variables. The expected complexity of simplex method is bounded by $n^{1+\ln n}$ where n is the maximum number of variables and constraints. [2] In practice, simplex method is much faster, but still is an expensive algorithm. Other methods like ellipsoid method, prove a polynomial time bound for solving an LP. [2]

C.Swamy and D.B.Shmoys provide a 4-factor approximation using lagrangian relaxation for the uniform requirement case on a euclidean metric. [4]

2.2 Motivation

The previous work closely related to this problem focuses on linear programs. Solving a Linear Program is computationally expensive. Moreover, the LP solution does not generalise to other metrics. We try to find a combinatorial solution to the problem so that we get insights to extend the solution to Hierarchically Separated Trees (HST).

Chapter 3

Preliminaries

3.1 Notations

Consider a finite set of clients C. The input space is the real line. Let us denote the ordered set $C = \{c_1, c_2, ...c_n\}$, ordered by non-decreasing magnitude. Similarly, we consider a finite set of facilities $F = \{f_1, f_2, ...f_m\}$ ordered by non-decreasing magnitude.

3.2 Basic Definitions

Definition 3.1 (**Solution**)**.** A solution to the fault tolerant facility location problem is a set of facilities to be opened $F_{open} \subseteq F$. An optimal solution is that solution that minimises the total cost of opening the facilities and serving all the clients' requirement

Lemma 3.1. *In a solution, given an open facility f, if it serves clients c_i and c_j for $i \leq j$, then it serves all clients c_k such that $i \leq k \leq j$.*

Proof. By way of contradiction, suppose that for an open facility f, it serves c_i and c_j $(i \leq j)$ and there exists a facility c_k, $i \leq k \leq j$ such that f does not serve c_k.

Without loss of generality assume that c_k lies to the right of f.

`Case 1`: $\exists f^{'} \in F_{open}$ to the right of c_k and the right of f such that serves $f^{'}$ serves c_k. Since this is an optimal solution, $D(c_k, f^{'}) < D(c_k, f)$. Since $j \geq k$, it means that $D(c_j, f) > D(c_j, f^{'})$. This is a contradiction to the solution being optimal.
`Case 2`: $\exists f^{'} \in F_{open}$ to the left of c_k and the right of f such that serves $f^{'}$ serves c_k. Again, this means $D(c_j, f^{'}) < D(c_j, f)$ which is a contradiction.
`Case 3`: $\exists f^{'} \in F_{open}$ to the left of c_k and the left of f such that serves $f^{'}$ serves c_k. But this is not possible since $D(c_k, f^{'}) > D(c_k, f)$, which is a contradiction.

We observe that these cases are exhaustive and lead to a contradiction. Thus, we say that the claim is true. □

The above claim allows us to define an interval associated with an open facility.

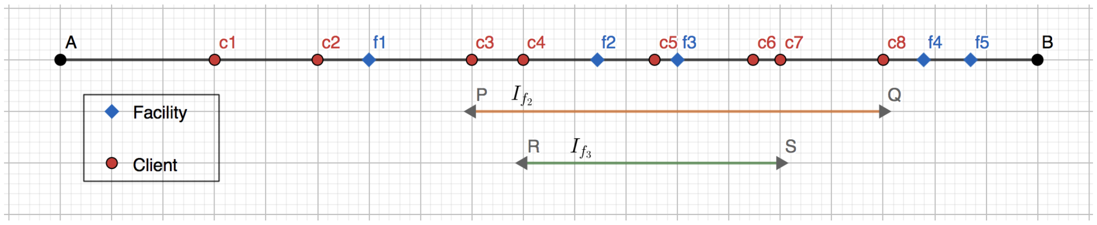

Figure 3.1: Containment

Definition 3.2 (Interval of an open facility). The interval $I_f = (L_f, R_f)$ of a facility $f \in F_{open}$ is the smallest segment that contains all the clients served by the f in a solution. Any client c served by f follows $L_f \leq c \leq R_f$. We associate a set of clients C_{I_f} with the interval I_f where $C_{I_f} = \{c \in C : L_f \leq c \leq R_f\}$.

Remark. Notice that I_f is the smallest segment that contains all clients served f. This means that there is a client at L_f and R_f. We denote these clients by L_f and R_f respectively.

Definition 3.3 (Containment). Given facilities f_1 and f_2 where $I_{f_1} = (L_{f_1}, R_{f_1})$ and $I_{f_2} = (L_{f_2}, R_{f_2})$, f_2 is contained in f_1 if $L_{f_1} < L_{f_2}$ and $R_{f_2} < R_{f_1}$, as shown in Fig 3.1

Lemma 3.2. *$\nexists f_1, f_2 \in F_{open}$ such that f_2 is contained in f_1.*

Proof. By way of contradiction, suppose Lemma 3.2 does not hold. Without loss of generality, let $f_1 < f_2$ (f_1 lies to the left of f_2). Since, $R_{f_1} > R_{f_2}$, implies $D(R_{f_1}, f_2) < D(R_{f_1}, f_1)$. Thus R_{f_1} is not connected to the nearest facilities. This is a contradiction. $\square$

Chapter 4

Fault Tolerant Facility Location

4.1 The Structure Theorem

4.1.1 The Theorem

In the previous section, we showed that every facility $f \in F_{open}$ is associated with an interval $I_f = (L_f, R_f)$ and f serves all the clients in C_{I_f}. We define a collection of these intervals $I_{F_{open}} = \{I_f : f \in F_{open}\}$, where $|I_{F_{open}}|$=$|F_{open}|$.

Theorem 1 (**Structure Theorem**)**.** $I_{F_{open}}$ can be partitioned into r subsets $\{S_i$: $1 \leq i \leq r\}$ such that $\forall i$, $\{C_{I_f} : I_f \in S_i\}$ is a partition of C.

4.1.2 Intuition

Let $f_i(c)$ denote the i^{th} facility that serves the client c, ordered by its location on the real line. There exists a one to many mapping from every client c to r distinct intervals $I_{f_i(c)}$, $1 \leq i \leq r$ that serve c. Since every client has exactly r intervals associated with it, it gives us an intuition that I_F can possibly be partitioned into r different sets or requirement layers, where the intervals in each requirement layer collectively serve each client exactly once.

4.1.3 Proof

We will present a constructive proof of the structure theorem. We start with r empty subsets, i.e, $\forall i$ $S_i = \{\phi\}$ and incrementally add intervals to the subsets such that the structure theorem holds, when the procedure ends.

Definition 4.1. An interval I_f is said to satisfy a client c if f serves a requirement of c in the solution. In other words, $c \in C_{I_f}$

We define a function Least Unsatisfied Client $LUC : \mathbb{P}(I_{F_{open}}) \rightarrow C$ that returns the leftmost client $c \in C$ in the input space whose requirement is not satisfied even once, by the intervals in the domain, "None" otherwise. $FindInterval : \mathbb{P}(I_{F_{open}})XC \rightarrow F_{open}$ takes in a set of intervals and a client as input and returns any interval in the set that serves the client, "None" otherwise. The constructive procedure is as follows:

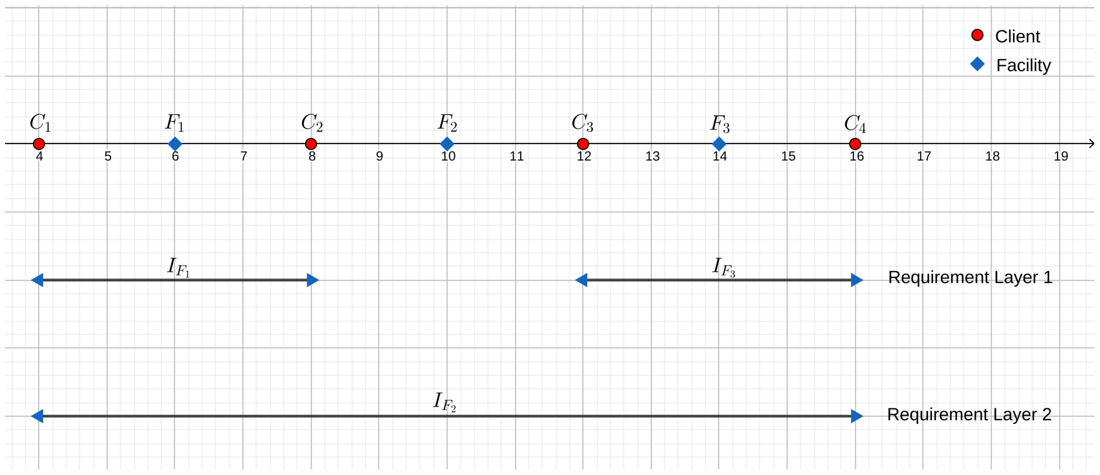

Figure 4.1: Visualisation of structure theorem

Algorithm 1 Structure Generation Algorithm

```
1: procedure PARTITION($I_{F_{open}}$)                          ▷ Returns the partitioned set S
2:     $IntervalList \leftarrow I_{F_{open}}$
3:     $S[1..r] \leftarrow \phi$                                 ▷ Initialise the partitions to $\phi$
4:     $SetToFill = \arg\min_i LUC(S[i])$                        ▷ Find the set with global LUC
5:     while $SetToFill! = None$ do                              ▷ Run the loop till there is an unsatisfied client
6:         $SmallestClient \leftarrow LUC(S[set])$               ▷ Client that has to be satisfied
7:         $IntervalToAdd \leftarrow FindInterval(IntervalList, SmallestClient)$
8:         $IntervalList.pop(IntervalToAdd)$                     ▷ Remove interval from current list of intervals
9:         $S[SetToFill].add(IntervalToAdd)$                     ▷ Add interval to the set
10:        $LUCset = \arg\min_i LUC(S[i])$                       ▷ Find the set with next LUC
11:    end while
12:    return $S$                                                ▷ S is the required partition
13: end procedure
```

The procedure assigns every client to r intervals because the procedure runs till there is no unsatisfied client. The procedure assigns every interval in the IntervalList to exactly one set (line 8 and 9) and since all the clients are satisfied, it is guaranteed that there are no more remaining intervals in *IntervalList*, therefore, this procedure is partitioning $I_{F_{open}}$ into r subsets. Now, we will show that the intervals in a subset satisfies all the clients exactly once.

Let c be the smallest unsatisfied client at any time step. This implies that any client $c' < c$ has all its r requirements satisfied. Since c is not satisfied yet, there must exist an interval I_f containing c, that FindInterval returns.

Claim. $L_f = c$

Proof. Suppose by the way of contradiction, $L_f \neq c$. $L_f < c$ because c is contained in I_f. In that case, there exists $c' < c$ such that f serves c'. However, we know that c' has its requirements satisfied. Adding the new interval will satisfy $r + 1$ requirements of c'. This is a contradiction because every client is satisfied exactly r times, in any solution. This proves that $L_f = c$. □

Since the claim holds, we can conclude that at any iteration, no previously satisfied client is satisfied again, in any subset. Hence, the intervals in a subset satisfies all the clients exactly once. This concludes that the procedure generates a valid partition of $I_{F_{open}}$ and $\forall i$, $\{C_{I_f} : I_f \in S_i\}$ is a partition of C, thereby proving the theorem.

4.2 The Flow Model

4.2.1 Notations

Consider two open facilities f_1 and f_2 such that there is no open facility between f_1 and f_2. We define the function $cost(f_1, f_2)$ as the minimum cost of serving all clients c ($f_1 \leq c \leq f_2$) exactly once, by f_1 or f_2.

Remark. Given a facility f, cost of serving all clients c ($c \leq f$) by f is defined as $cost(-\infty, f)$ while cost of serving all clients c ($f \leq c$) by f is defined as $cost(f, \infty)$.

4.2.2 Cost Function Computation

Lemma 4.1. *The function $cost(f_1, f_2)$ is calculable in $O(|C|)$ time.*

Proof. The following algorithm gives us the value of the function $cost(f_1, f_2)$.

Algorithm 2 Cost Calculation for adjacent facilities

```
1: procedure COST($f_1, f_2$)                                ▷ Returns the value of $cost(f_1, f_2)$
2:     $result \leftarrow 0$                                 ▷ Initialise the result with zero
3:     for $c$ in $C$ do                                     ▷ Iterate over all clients
4:         $result \leftarrow result + max(0, min(c - f_1, f_2 - c))$ ▷ Add cost of serving client by nearer facility if it is between $f_1$ and $f_2$
5:     end for
6: return $result$
7: end procedure
```

The above algorithm iterates over the set of clients C and in each iteration it computes the cost of serving the client once, in $O(1)$ time. Thus, the runtime of the algorithm is $O(|C|)$.

$\square$

4.2.3 Flow Network

Discussion

The structure theorem discussed in chapter 3 enables us to model the fault tolerant facility location as a minimum cost maximum flow problem that runs in polynomial time. We try to model the flow network in such a way that every feasible flow corresponds to the cost incurred in a requirement layer. Suppose if facilities $(f_{i_1}, f_{i_2}, ...f_{i_p})$ are opened in any requirement layer, the cost of serving all the clients in that layer breaks down into $cost(-\infty, f_{i_1}) + \sum_{j=1}^{p-1} cost(f_{i_j}, f_{i_{j+1}})$ $+$ $cost(f_{i_p}, \infty)$. This gives us a hint that the cost can be captured in different edges, summed up and minimised using the standard min cost max flow framework.

Flow Network

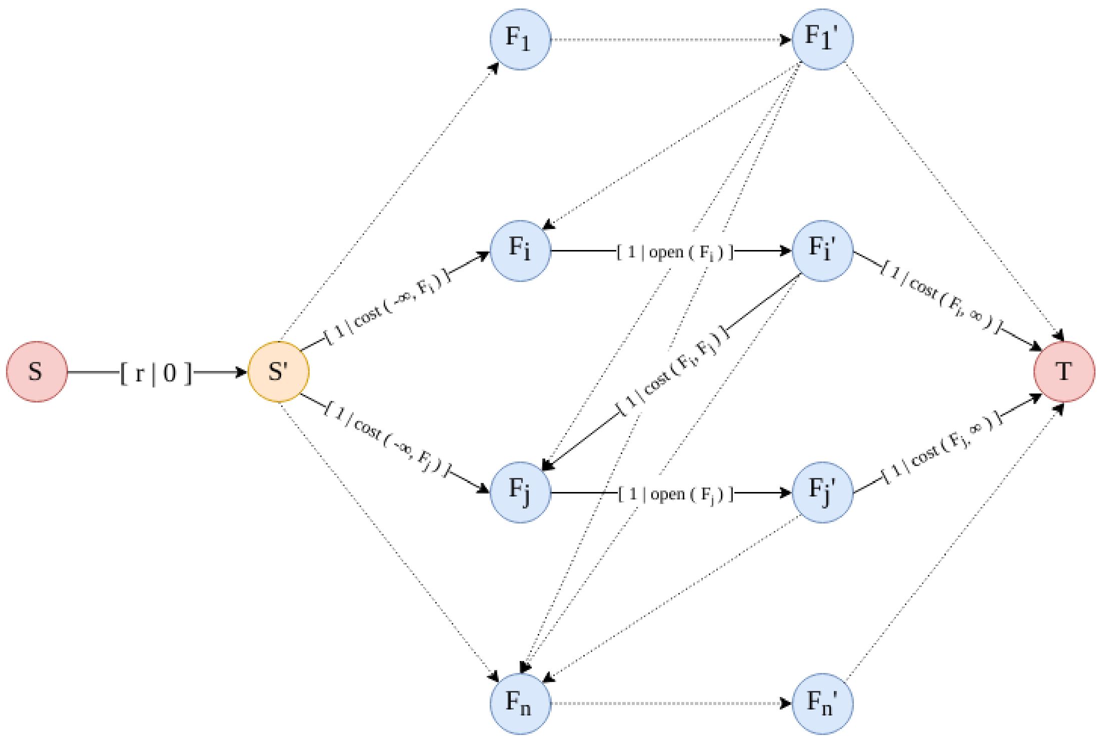

Figure 4.2: Flow Network

Let $n = |F|$ and $m = |C|$. The graph is modelled as shown in Fig 4.2.

- Source vertex s.
- Destination vertex t.
- Add an edge from s to s' with $cost = 0$ and $capacity = r$.
- $\forall i : 1 \leq i \leq n$, add an edge from s' to f_i with $cost = cost(-\infty, f_i)$ and $capacity = 1$.
- $\forall i : 1 \leq i \leq n$, add an edge from f_i to f'_i with $cost = open(f_i)$ and $capacity = 1$.

- $\forall i : 1 \leq i \leq n$ and $\forall j : i+1 \leq j \leq n$, add an edge from f'_i to f_j with $cost = cost(f_i, f_j)$ and $capacity = 1$.
- $\forall i : 1 \leq i \leq n$, add an edge from f'_i to t with $cost = cost(f_i, \infty)$ and $capacity = 1$.

Theorem 2. Min-cost Max-flow from s to t gives the minimum cost solution to the uniform fault tolerant facility location on a line metric, and the facilities opened are the facilities through which the flow passes.

4.2.4 Proof of Correctness

Idea

We will prove that the min cost flow in the above described flow network gives an optimal solution to the Fault tolerant Facility Location problem. First, we say that any feasible flow in the above network is a solution to Facility Location problem. Next, we say that any solution to Facility Location can be modelled as a feasible flow. Now, since every solution is a feasible flow and every feasible flow is a solution, we can say that the optimal solution (that minimizes the cost) is the min-cost max flow.

Proof

Consider a flow of unit 1 being sent from source s. The augmenting path is the following $ss'f_{i_1}f'_{i_1}f_{i_2}...f_{i_k}f'_{i_k}t$ for some facilities f_{i_j}, $1 \leq j \leq k$.

Claim. $C_{I_{f_{i_1}}}$, $C_{I_{f_{i_2}}}$, ... $C_{I_{f_{i_k}}}$ is a partition of C.

Proof. First we prove that

$$\forall c \in C, c \in \bigcup_{j=1}^{k} C_{I_{f_{i_j}}}$$

Notice that the graph is constructed in such a way if there is an edge between f'_x and f_y, f_y lies to the right of f_x. Consider f'_{i_x} and f_{i_y} in the augmenting path such that there is an edge between the two. This implies there is no facility f in the augmenting path such that $f_{i_x} < f < f_{i_y}$. Therefore, the augmented path is a sequence of adjacent opened facilities in order. Consider the facility f_{i_i} and last facility f_{i_k} in the augmenting path. Since every edge between adjacent facilities sums the cost of serving clients between the two facilities, we say that all c such that $f_{i_1} \leq c \leq f_{i_k}$ are served by their nearest facility in the augmenting path. Moreover, the edge (s, f_{i_1}) sums the cost of serving all clients c such that $c \leq f_{i_1}$. Similarly, the edge (f'_{i_k}, t) sums the cost of serving all clients c such that $f_{i_k} \leq c$. Hence, we can say

$$\forall c \in C, c \in \bigcup_{j=1}^{k} C_{I_{f_{i_j}}}$$

Now, we prove that $C_{I_{f_{i_p}}} \cap C_{I_{f_{i_q}}} = \emptyset$, $\forall p, q \in \{1...k\}$.
In the algorithm given to calculate the cost on each edge, we sum up the cost of serving client c by its nearest of the adjacent open facilities. Thus, in each augmenting path, every client c is served by exactly 1 facility. Thus, each client c lies in exactly one $C_{I_{f_{i_j}}} \Rightarrow C_{I_{f_{i_p}}} \cap C_{I_{f_{i_q}}} = \emptyset$

Hence, the claim is true.

$\square$

The claim indicates that the augmenting path with flow 1 accounts for the cost of one of the requirement layers in the solution according to the structure theorem.

We can say that a feasible flow is a solution if and only if every augmenting path is a requirement layer such that there are exactly r augmenting paths, and that each facility f lies in at most 1 augmenting path.

It is easy to see that the max-flow in the flow network is r. Since the requirement for each client is r, we know that there are at least r facilities in the network. Now, for each facility f, the node f' is connected to source s ($s \rightarrow s' \rightarrow f \rightarrow f'$). Also, $f' \rightarrow t$. Thus, there are at least r in incoming edges on t from s each with capacity r. Thus, the sink can collect at least r units of flow. Since, the flow pushed by source s is r, the max flow is r.

For every facility f, maximum incoming flow can be capacity of (s', f) which is equal to 1. Since we consider integral flows, once a facility f is in an augmenting path, the residual incoming capacity is zero. Thus, f cannot be a part of any other augmenting path. So, a facility lies in at most 1 augmenting path.

This proves that every feasible flow is a solution to the facility location problem.

We complete the proof by saying that a solution to facility location problem is a feasible flow in the flow network.

Using the structure theorem, we say that $I_{F_{open}}$ can be partitioned into r subsets $\{S_i$: $1 \leq i \leq r\}$ such that $\forall i$, $\{C_{I_f} : I_f \in S_i\}$ is a partition of C. We have already proved that there will be exactly r augmenting paths with flow 1 in a feasible flow. That means, if we are able to show that there is a one to one mapping from each requirement layer to an augmenting path, we are done.

Consider, a requirement layer L such that it contains facilities $f_{i_1}, f_{i_2}...f_{i_k}$. All clients are served by their nearest facility in this requirement layer. Consider an augmenting path $ss'f_{i_1}f'_{i_1}f_{i_2}...f_{i_k}f'_{i_k}t$. A flow of unit 1 in this augmenting path gives a cost exactly equal to the cost of serving each client c by its nearest facility in the set $\{f_{i_1}, f_{i_2}, f_{i_k}\}$. We know that each open facility is present is at most one requirement layer. We say that a facility is open if it is present in an augmenting path. Since, no facility can be present in more than 1 augmenting path, there is a one to one mapping from requirement layer to augmenting path such that the cost of the augmenting path is equal to the cost incurred in the requirement layer. Thus, every solution is a feasible flow.

This completes the proof of correctness of Theorem 2.

4.2.5 A note on complexity

It is known that the min-cost max flow can be solved in $\mathcal{O}(|E|^2 log|V| + |E||V|log^2|V|)$ time using Orlin's algorithm [3] where E and V are the set of edges and vertices in the graph, respectively. Our network has $\mathcal{O}(n^2)$ edges and $\mathcal{O}(n)$ vertices, therefore the min cost max flow algorithm runs in $\mathcal{O}(n^4 logn + n^3 log^2 n)$ Also, the cost function can be computed in $\mathcal{O}(m)$ time (**Lemma 4.1**), hence the graph can be generated in $\mathcal{O}(n^2 m)$ time. Summing up all these factors, the total running time is $\mathcal{O}(n^4 logn + n^3 log^2 n + n^2 m)$.

Chapter 5

Fault Tolerant K-Median

5.1 FTFL to FTM Adaptation

This section deals with extending the approach used in solving FTFL to solve FTM.

5.1.1 Intuition

In FTM, the facilities do not have an opening cost but rather the constraint is to open atmost k facilties. Thus, the opening cost of a facility does not affect the solution. In the setting of FTFL, suppose the opening cost of all facilities is some constant (say p). Now, the only differentiating factor between two facilities is the total distance from the clients it serves. Notice that if $p = 0$, we will open all the facilities and if $p = \infty$, we will open only r facility that are nearest to the median of the clients. This gives an idea that increasing p reduces the number of opened facilities.
Claim. Let the number of opening facilities for an opening cost p be $N(p)$. For $p_1 \leq p_2$, $N(p_1) \geq N(p_2)$.

Proof. By contradiction, let us suppose that $N(p_1) < N(p_2)$.

Now, two cases arise:

- Case 1 : $TotalCost(p_1) \geq TotalCost(p_2)$

$$\Rightarrow N(p_1)p_1 + ServingCost(p_1) \geq N(p_2)p_2 + ServingCost(p_2)$$

  This is a contradiction to $TotalCost(p_1)$ being optimal, as we can open the facilities we opened for cost p_2, and the total cost will become $N(p_2)p_1 + ServingCost(p_2) < TotalCost(p_2) \leq TotalCost(p_1)$.

- Case 2 : $TotalCost(p_1) < TotalCost(p_2)$

$$\Rightarrow N(p_1)p_1 + ServingCost(p_1) < N(p_2)p_2 + ServingCost(p_2)$$

  This is a contradiction to $TotalCost(p_2)$ being optimal, as we can open the facilities we opened for cost p_1, and the total cost will become $N(p_1)p_2 + CostServing(p_1) < TotalCost(p_2)$

$\square$

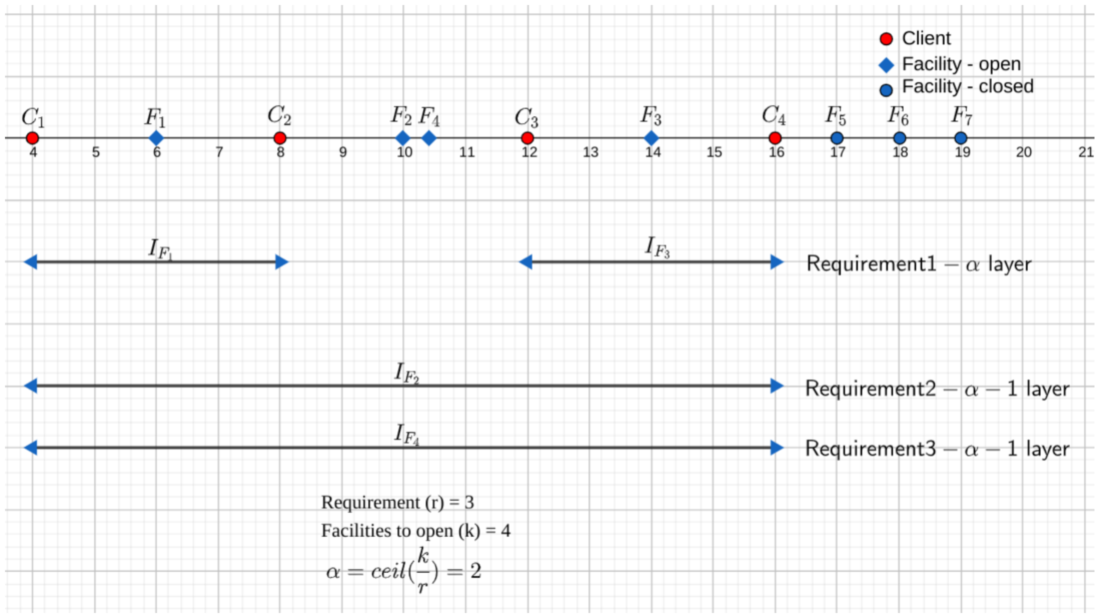

Figure 5.1: Visualisation of modified structure theorem

5.1.2 Proposed Solution

The previous claim opens the avenue for binary search. We binary search over the search space $[0, \infty)$ of opening cost p, to find an optimal cost $p*$ such that the number of opened facilities, $N(p*) = k$.

Over here, ∞ is defined as the opening cost p_∞ such that $N(p_\infty) = 1$. We define

$$d_{max} = \max_{c \in C, f \in F} D(c, f)$$

Claim. $p_\infty = d_{max} * |C|$

Proof. To prove this, it is sufficient to prove that the total cost of opening r facilities is always less than that of opening $r+1$ facilities. Now, total cost of opening r facilities, $TC(r) = r * d_{max} * |C| + cost_{serving}$. For any client c and facility f, $D(c, f) \leq d_{max}$. Thus, $cost_{serving} \leq d_{max} * |C|$. Hence, $TC(r) \leq (r + 1) * d_{max} * |C| \leq TC(r + 1)$ □

Now, the search space reduces to $[0, p_\infty]$. We binary search over this space to find $p*$ such that $N(p*) = k$ in polynomial time.

5.1.3 Issues

The assumption for the above algorithm to work is that $\forall j \in [r, |F|], \exists p \in [0, p_\infty]$ such that $N(p) = j$. However, this is not true. We found a counter example by simulating the procedure on a computer.

5.2 Modified Structure Theorem

We have shown that the structure theorem partitions the facilities into different requirement layers. If the size of a requirement layer is the number of facilities serving in it, how large can

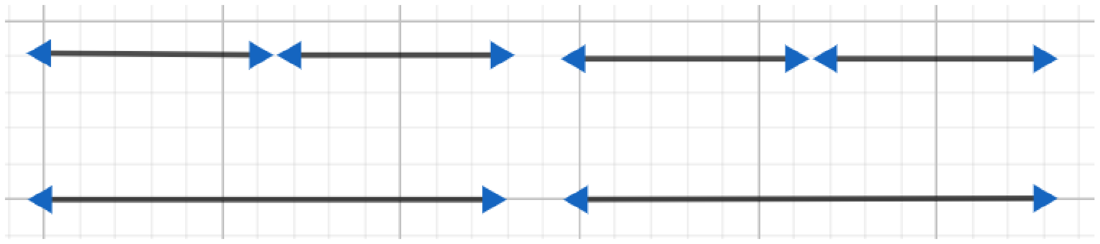
Figure 5.2: Interval swapping argument

a requirement layer be? Intuitively, it seems that it is not possible to have just 1 facility in one requirement layer and 10 facilities in another. If it happens, one can always try to push facilities from one requirement layer to the other, and decrease the cost. This is a consequence of Lemma 3.2. Keeping this fact in mind, can we say that the number of facilities in every requirement layer in any solution should roughly be the same? This thought leads to an enhancement in the structure theorem proposed earlier. Formally, the theorem is stated as follows.

Theorem 3 (**Modified Structure Theorem**). Let $\alpha = \lceil \frac{k}{r} \rceil$. $I_{F_{open}}$ can be partitioned into r subsets $\{S_i\colon\ \alpha - 1 \leq |S_i| \leq \alpha\}$ such that $\forall i$, $\{C_{I_f} : I_f \in S_i\}$ is a partition of C.

Proof. Recall Lemma 3.2. The lemma states that the interval of any facility cannot be strictly contained inside another. This implies that for any two requirement layers i and j, $||S_i| - |S_j||$ cannot be too large. If it was too large, it'll lead to containment of intervals. How large can this difference be?

Claim. $||S_i| - |S_j|| \leq min(|S_i|, |S_j|)$

This claim can be easily verified by looking at the worst case scenario, as shown in Figure 5.2. We can easily see that every interval in the second layer has two intervals in the corresponding range of the previous interval. Also, note that this scenario is valid as the intervals are not completely contained. The left end point for both the intervals in the two layers are the same, which ensures that the containment property (Lemma 3.2) is not violated. Here, if we try to squeeze in an extra layer, the Lemma 3.2 will be violated. Therefore, the number of intervals in the layer with larger facilities is at most twice the number of facilities in the layer with lesser facilities.

This idea shows us that the difference between the number of facilities in any two layers can be high only when the intervals share common end points. If no two intervals in two requirement layers share an end point, the difference in the number of facilities will be at most 1. Therefore, if there are two requirement layers that differ by two or more facilities, we can find an interval with a common end point, and swap the set of facilities to one side of the end point in both the requirement layers (In figure 5.2, the first two intervals in the first requirement layer can be swapped with the first interval in the second requirement layer, thereby decreasing the difference in the number of facilities to 0. However, in certain cases, we might not be able to decrease the diffence below 1, because k mod r is not 0. This scenario is possible and that leads to layers with $\alpha - 1$ facilities, which will be discussed later). This swapping will not affect the cost of the solution because both the intervals are serving up to the same client and we are just trying to move the facilities to different requirement layers. Such swapping will decrease the difference between the number of facilities in the two requirement layers. Therefore, we can keep doing this procedure recursively until there are no two requirement layers such that the difference between the number of facilities is greater than or equal to 2.

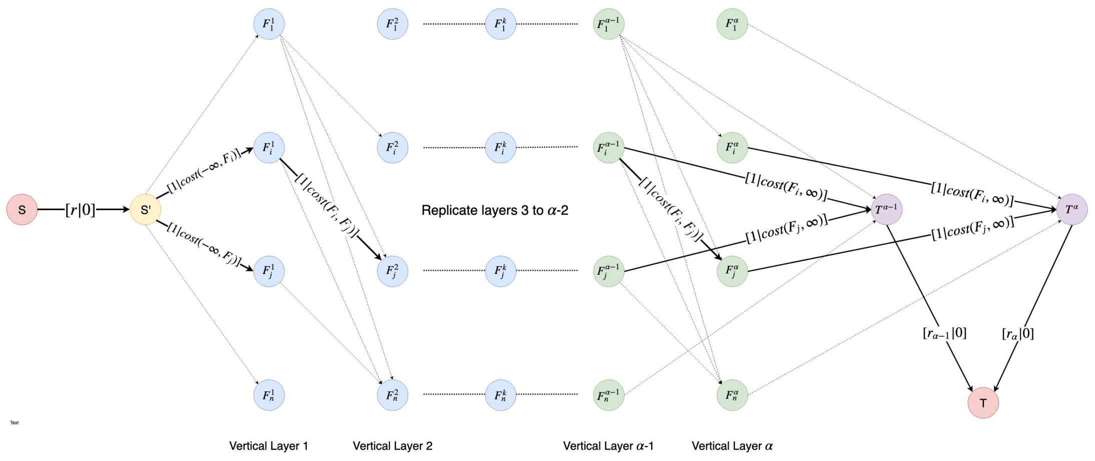

Figure 5.3: Proposed flow model for k-median

When the procedure described above ends, it's guaranteed that each requirement layer will have exactly α or $\alpha - 1$ ($\alpha - 1$ is possible because the difference is at most 1) facilities. Since there are exactly k intervals in total, $\alpha = \lceil \frac{k}{r} \rceil$. □

Corollary 3.1. Let $r_{\alpha-1}$ equal to the number of requirement layers S_i of size $\alpha - 1$ and r_α equal to the number of requirement layers S_i of size α. Then, $r_\alpha = k - r(\alpha - 1)$ and $r_{\alpha-1} = r - r_\alpha$

Proof. Since there are exactly r requirement layers in total, $r_{\alpha-1} + r_\alpha = r$. Also, from Theorem 3, we know that $(\alpha)r_\alpha + (\alpha - 1)r_{\alpha-1} = k$. Solving the two equations, we get $r_\alpha = k - r(\alpha - 1)$. Hence, the corollary holds true. □

5.3 K-Median Flow Model

5.3.1 Discussion

We follow a similar idea as used in FTFL, where each unit of flow represents a requirement layer, and cost over the edges represents the cost of serving clients. Using Theorem 3, we know that in each requirement layer there will be either α or $\alpha - 1$ intervals. Thus, our flow model needs to make sure of that. To incorporate this, we create α copies of each facility. An edge from copy i to copy $i + 1$ represents interval $i + 1$ in the requirement layer. Now, there are exactly α copies and the last and second last copy have an edge to the sink. Thus, in a requirement layer there will be either α or $\alpha - 1$ intervals. Now, we can figure out how many α layers are there and thus how many $\alpha - 1$ layers are there. Using this information, we create a super sink and connect α sink with a flow of r_α and $\alpha - 1$ sink with a flow of $r_{\alpha-1}$ where r_i is the number of requirement layers with exactly i intervals.

5.3.2 Flow structure

The graph is modelled as shown in Fig 5.3.

- Source vertex s
- Destination vertex t

- Add an edge from s to s' with capacity r and cost 0 (representing number of requirement layers.
- $\forall i \in [n]$, add an edge from s' to F_i^1 with capacity 1 and cost $= cost(-\infty, F_i)$
- $\forall i \in [n], \forall j in [i+1, n]$ and $\forall k in [\alpha - 2]$, add an edge from F_i^k to F_j^{k+1} with cost $= cost(F_i, F_j)$ and capacity 1).
- $\forall i \in [n]$, add an edge from $F_i^{\alpha-1}$ to $T^{\alpha-1}$ with cost $= cost(F_i, \infty)$ and capacity 1.
- $\forall i \in [n]$, add an edge from F_i^{α} to T^{α} with cost $= cost(F_i, \infty)$ and capacity 1.
- Add an edge from $T^{\alpha-1}$ with cost $= r_{\alpha-1}$ and capacity 0.
- Add an edge from T^{α} with cost $= r_{\alpha}$ and capacity 0.

5.3.3 Issues

The issue with above structure is that it does not guarantee that a facility will be opened at most once, because of the presence of multiple copies of the same facility in multiple vertical layers or groups. That means, if a facility can serve the same client multiple times to achieve a smaller cost, the above flow model will allow that. There is a potential solution to this problem, which will be discussed in the next section.

5.4 Potential Workaround

The issue in the above flow model can potentially be solved using the idea of vertical groups.

Definition 5.1 (**Vertical Group**)**.** A vertical group, V_i is a set of all facilities f_i such that they serve the i^{th} interval for each requirement layer.

Lemma 5.1. *$\forall f \in V_i$, $\forall f' \in V_j$ such that $j > i$, $f < f'$. That means all the facilities in the i^{th} vertical group are to the left of all the facilities in the j^{th} vertical group such that $j > i$.*

5.4.1 Implications

Since there is a strict ordering of the vertical groups, if we can somehow find the underlying partition of the vertical groups in polynomial time, we can place the facilities corresponding to the i^{th} vertical group in the i^{th} vertical layer in the flow model proposed in Fig 5.3. This ensures that there is exactly one copy of all the facilities over all the vertical groups, thereby solving the issue that posed a challenge in the previous section. However, as of now, no polynomial time approach is known to find the partitions.

Chapter 6

Future Work

The structure theorem is a powerful result that has enabled us to model the FTFL problem using flows. However, retrofitting gadgets to this flow model to solve FTM is not very trivial due to the hard constraint that the maximum number of facilities opened is at most k. In the second half of the semester, we proposed the modified structure theorem which defined the number of facilities in each requirement layer. However, due to the presence of multiple copies of the same facility in multiple vertical groups, the k-median flow model fails. We know that there exists a total order on the vertical groups which enables us to remove unwanted copies of facilities in the flow model. The next step would be to try and figure out a polynomial time algorithm that can identify these vertical group boundaries, either by retrofitting a gadget in the existing flow model, or by exploiting the inherent properties and constraints in the problem.

References

1. Mohammad Taghi Hajiaghayi, Wei Hu, Jian Li, Shi Li, and Barna Saha. A constant factor approximation algorithm for fault-tolerant k-median. *ACM Trans. Algorithms*, 12(3):36:1–36:19, 2016.
2. Allen Holder. Understanding and using linear programming by jiří matoušek; bernd gärtner; introduction to optimization by pablo pedregal. *The American Mathematical Monthly*, 116(5):471–476, 2009.
3. James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.
4. Chaitanya Swamy and David B. Shmoys. Fault-tolerant facility location. *ACM Trans. Algorithms*, 4(4):51:1–51:27, 2008.