Submodularity Property for Facility Locations of Dynamic Flow Networks

Peerawit Suriya

630510500

Department of Mathematics, Faculty of Science

Chiang Mai University

Academic Year 2023

Submodularity Property for Facility Locations of Dynamic Flow Networks

Peerawit Suriya

630510500

This independent research is part of the Bachelor of Science program.

Department of Mathematics, Faculty of Science

Chiang Mai University

Semester 2/2023

Submodularity Property for Facility Locations of Dynamic Flow Networks

Received approval to be part of

the Bachelor of Science program in Mathematics

The Committee for Supervision of Independent Research

..... The Chairman of the Committee

(Asst. Prof. Dr. Suparut Chaidee)

..... The Committee Member

(Asst. Prof. Dr. Piyashat Sripratak)

4 March 2024

Acknowledgments

This independent research is part of the study in the course of Independent Study (206499) according to the Bachelor of Science program in Mathematics, Faculty of Science, Chiang Mai University. The objective is for students to study and apply mathematical theories and concepts in conducting independent research. The research focuses on studying Submodularity Property for Facility Locations of Dynamic Flow Networks.

I would like to express my sincere gratitude to all those who have contributed to the completion of this research.

With deep appreciation, I thank Assistant Professor Dr. Supanut Chaidee, my esteemed adviser, for all of his support, profound insights, and important assistance during this research project. I am sincerely appreciative of his generous sharing of his wealth of knowledge and experience, since his guidance and expertise have been important in determining the course of this research. I have gained a deeper comprehension of the subject matter as well as improved the quality of my work thanks to his insightful criticism.

I extend my sincere gratitude to Associate Professor Vorapong Suppakitpaisarn for his exceptional guidance during my research internship at the University of Tokyo. I am thankful for the opportunity to work under his supervision, which not only contributed to my academic growth but also provided me with valuable insights into the world of research.

I would like to thank Assistant Professor Dr. Piyashat Sripratak for her insightful criticism and helpful ideas, which really helped to improve this work.

I sincerely appreciate all of my family's understanding and constant support. Their confidence in me and their encouragement have always been a source of inspiration.

In conclusion, I am deeply grateful to everyone mentioned above. Your collective support has been instrumental in the successful completion of this academic endeavor.

Peerawit Suriya

Research Title	Submodularity Property for Facility Locations of Dynamic Flow Networks
Researcher	Peerawit Suriya Student ID 630510500
The Chairman of the Committee	Asst. Prof. Dr. Supanut Chaidee
The Committee Member	Asst. Prof. Dr. Piyashat Sripratak

Abstract

This independent study considers facility location problems within dynamic flow networks, shifting the focus from minimizing evacuation time to handling situations with a constrained evacuation timeframe. Our study sets two main goals: 1) Determining a fixed-size set of locations that can maximize the number of evacuees, and 2) Identifying the smallest set of locations capable of accommodating all evacuees within the time constraint. We introduce $\mathsf{flow}_t(S)$ to represent the number of evacuees for given locations S within a fixed time limit t. We prove that flow_t functions is a monotone submodular function, which allows us to apply an approximation algorithm specifically designed for maximizing such functions with size restrictions. For the second objective, we implement an approximation algorithm tailored to solving the submodular cover problem. We conduct experiments on the real datasets of Chiang Mai, and demonstrate that the approximation algorithms give solutions which are close to optimal for all of the experiments we have conducted.

Table of Contents

Chapter 1

Introduction

1.1 Background

In these days, natural disasters are becoming increasingly common and destructive in our modern society. This trend is concerning, as it puts communities at risk and needs urgent help. In such challenging circumstances, it becomes clear how essential it is to prepare for disasters and provide assistance. One of the most vital parts of disaster preparedness is ensuring that evacuation centers are strategically placed in locations that are easily accessible to those in need.

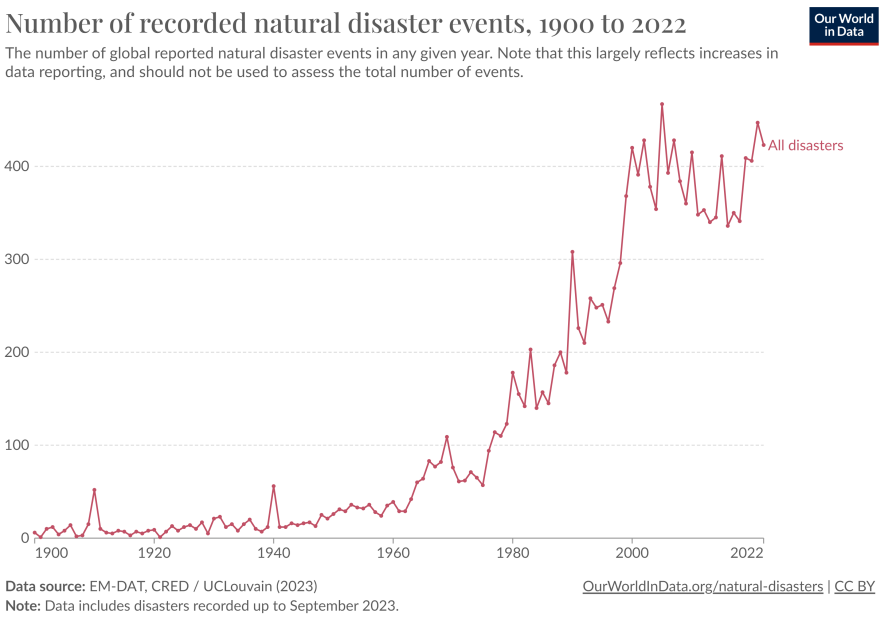

Figure 1.1: The graph illustrates the number of global reported natural disaster events from 1900 to 2022. (Data is acquired from www.ourworldindata.org.)

The facilities location problem [1] is one of the well-known problems for finding the optimal location of facilities that optimizes certain criteria, such as minimizing costs, under the given constraints and considerations. In terms of a graph $G = (V, E)$, the standard problem statement is to identify a subset S from V comprising of k nodes. This subset S should have the property that it minimizes the length of the longest shortest path from any node $v \in V$ to a node within the subset S. There are several approximation algorithms proposed for this standard problem statement such as [2, 3].

The dynamic network [4, 5] is a graph that includes information that changes over time. An illustrated example of a dynamic network is a plan for evacuating people from nodes in a static graph, representing the intersection of roads, to facilities when the edges, representing the roads that connect these nodes' locations. Every node $v \in V$ starts off with a varying number of evacuees at the onset of evacuation. However, each edge $e \in E$ has a capacity limit that restricts the number of people it can accommodate at any given moment. The time-expanded networks approach outlined in [4] can be used to determine the best possible evacuation plan. By leveraging this method, it is feasible to compute the necessary time frame for the complete evacuation of individuals. If there exists a time constraint on the evacuation, this technique can also be used to estimate the maximum number of evacuees that can be transported to safe facilities within the specified time limit.

Extending the problem formulation of facility location to a dynamic network is natural. The goal would be to identify a collection of facilities that could minimize the evacuation time. Several algorithms were proposed for the case of path graph [6], tree [7], and general graphs [8], as summarized in [9].

1.2 Our Contributions

While the majority of prior research has centered around minimizing evacuation time, we argue, based on [10], that real-world evacuations often operate under strong time constraints. This observation has led us to examine the following two variants of facility location problems within dynamic networks:

- **Problem 1:** Given the number of facilities, locate a subset of facilities which can accommodate the maximum number of evacuees in a given time.
- **Problem 2:** Locate a smallest set of facilities which can accommodate all evacuees in a given time.

We may consider that both of the problems are closer to the k-center problem where we aim to minimize the maximum evacuees time.

Our main contribution of this independent study is:

> Define $\mathsf{flow}_t(S)$ as the count of evacuees that can be accommodated by facilities positioned at a set of nodes, S, in time t. We demonstrate that flow_t exhibits the properties of a monotone submodular function.

Consequently, Problem 1 can be reformulated as the maximum submodular function problem subject to size constraints, as discussed in [11]. The greedy algorithm, which carries an

approximation ratio of $1 - 1/e$, which is approximately equal to 0.63, can be deployed, delivering a $(1 - 1/e)$-approximation algorithm for Problem 1. On the other hand, Problem 2 can be reformulated as the minimum submodular cover problem [12]. We can employ the $O(\log n)$-approximation algorithm for the problem to solve Problem 2.

Through numerical experimentation, we demonstrate that the algorithms provide solutions that are closely aligned with optimal solutions. We construct a real dataset from the road network, road capacity, and the resident count in each region of Chiang Mai, Thailand. For Problem 1, we compare the capacity of evacuees accommodated by facilities derived from the greedy algorithm against optimal solutions from an exhaustive search. For Problem 2, we notice that the approximation algorithm can find an optimal solution for our dataset.

Chapter 2

Preliminaries

In this section, we give the definitions of graph and statement of our problem, together with notations, basic concepts, and the Ford-Fulkerson algorithm, which we mainly use in this study.

2.1 Graph Definitions

A graph is a mathematical structure. To represent the relationship between the objects, it consists of vertices, which represent the object, and edges, which represent the relationship.

Definition 2.1 [13] A **graph** G is represented as an ordered pair (V, E), where V is the set of vertices and E is the set of edges. Each edge in E represents a connection between two vertices, forming a 2-element subset of V. Formally, a graph can be expressed as:

$$G = (V, E)$$

where V is the set of vertices, and E is the set of edges.

In directed graphs, edges have a direction, indicating a one-way relationship between vertices. Directed graphs are frequently used to simulate situations in which the direction or sequence of connections is important.

Definition 2.2 [14] A **directed graph** (or **digraph**) G is an ordered pair (V, E), where V is a set of vertices and E is a set of directed edges (arcs). Each directed edge in E is an ordered pair of vertices. Formally, a directed graph can be expressed as:

$$G = (V, E)$$

where V is the set of vertices, and $E \subseteq V \times V$ is the set of directed edges.

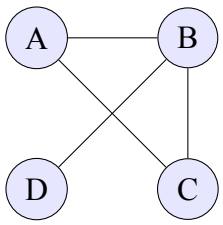

Figure 2.1: Example of a Graph

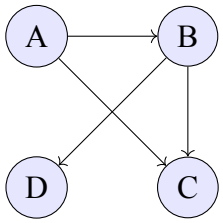

Figure 2.2: Example of a Directed Graph

A path in a graph is a series of connected vertices where one edge connects each pair of vertices in order. The number of edges a path crosses determines the length.

Definition 2.3 [15] Let $G = (V, E)$ be a directed graph. An **undirected path** in G is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that (v_i, v_{i+1}) or (v_{i+1}, v_i) is directed edge in E for $1 \leq i \leq k-1$. The length of the path is the number of edges in the sequence.

A network structure that changes over time is referred to as dynamic. Unlike static networks, which have consistent connections between nodes, dynamic networks change over time in terms of topology, connections, or other attributes.

Definition 2.4 [4] A **dynamic network** is a mathematical model of a system whose nodes, or the connections between them, change over time. Formally, let $G(t) = (V, E(t))$ denote a dynamic network at time t, where V is the set of nodes and $E(t)$ is the set of edges present in the network at time t. The topology, properties, or attributes of nodes and edges in $G(t)$ may change over time.

While the concept of the maximum flow problem and the Ford-Fulkerson algorithm are fundamental to graph theory, one might consider their inclusion unnecessary. However, given their instrumental role in demonstrating our main results in Section 3, we assert their discussion is crucial for maintaining the completeness of this work.

In graph theory, the maximum flow problem is a well-known optimization problem. In the problem, a network, $(G = (V, E), s, t, c)$, is defined as a directed graph, $G = (V, E)$, with the capacity function $c : E \to \mathbb{Z}_+$ such that each edge $(u, v) \in E$ has a capacity $c(u, v) \in \mathbb{Z}_+$ that represents the maximum amount of flow that can be sent through it. $s \in V$ is the source node and $t \in V$ is the sink node. Let $f : E \to \mathbb{Z}_{\geq 0}$ be a function that represents the flow, $f(e)$ be the flow that is sent through the edge e. Finding the maximum flow value from a source node to a sink node under the capacity constraint and flow conservation is the main concept to solve this problem.

Capacity constraint is the limitation of the amount of flow which can be sent through each edge. The amount of flow that can be sent through must be less than or equal to the capacity of that edge which means for all $e \in E$, $f(e) \leq c(e)$.

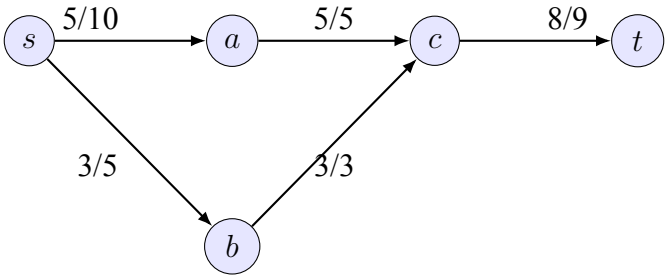

Figure 2.3: An example of network flow that has sent the maximum flow, which is equal to 8, from source node s to sink node t. The m/n on the edge of the graph represents the amount of flow that sent through the edge m and the capacity of the edge n.

The flow conservation is the property that for any vertex that is not a source node or sink node, the incoming and outgoing flows are equal. That is for all $v \in V \backslash \{s, t\}$,

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{u:(v,u)\in E} f(v, u). \tag{2.1}$$

The value of flow which is denoted by $|f|$ is the amount of outgoing flow from a source node which is equal to the amount of incoming flow to a sink node. That is

$$|f| = \sum_{u:(s,u)\in E} f(s, u) = \sum_{v:(v,t)\in E} f(v, t). \tag{2.2}$$

f is a feasible flow if f satisfies capacity constraint and flow conservation. Therefore, the maximum flow problem aims to find a feasible flow f such that $|f|$ is maximized.

The maximum value of flow in a flow network can be calculated using the Ford-Fulkerson algorithm [16]. The algorithm is described as in Algorithm 1.

Definition 2.5 Consider a directed graph $G = (V, E, c)$ such that $\mathsf{s}, \mathsf{t} \in V$, where s is the sink node and t is the source node. An s-t path flow of G, or path flow for short, is represented as p. Let $E(p)$ be the set of directed edges along the path p. Additionally, the flow value of p is symbolized by $\nu(p)$.

Let α be a set of s-t path flows. We say that α is an s-t path flow set of a graph $G = (V, E, c)$ if, for any $e \in E$ such that $\bar{e} \in E$,

$$-c(\bar{e}) \leq \sum_{p\in\alpha:e\in E(p)} \nu(p) - \sum_{p\in\alpha:\bar{e}\in E(p)} \nu(p) \leq c(e), \tag{2.3}$$

and, for $e \in E$ such that $\bar{e} \notin E$,

$$0 \leq \sum_{p\in\alpha:e\in E(p)} \nu(p) - \sum_{p\in\alpha:\bar{e}\in E(p)} \nu(p) \leq c(e), \tag{2.4}$$

We say that α is an s-t maximum path flow set of G if, for any s-t path flow set of G denoted by α', $\sum_{p\in\alpha'} \nu(p) \leq \sum_{p\in\alpha} \nu(p)$.

The algorithm outputs a set of path flows α. For each $p \in \alpha$, let the set of edges of p be $E(p)$. For each $e \in E$, define

$$f_\alpha(u,v) = \max\left\{ \sum_{p\in\alpha:(u,v)\in p} \nu(p) - \sum_{p\in\alpha:(v,u)\in p} \nu(p), 0 \right\}. \tag{2.5}$$

It is known that f_α is a maximum flow of G. We call the graph G_α in Line 4 of the algorithm as a residual graph obtained from G and the path flow set α.

Algorithm 1: Ford-Fulkerson Algorithm [16]

Input: Directed graph $G = (V, E, c)$, source node $s \in V$, sink node $t \in V$

Output: A set of path flows α such that f_α is a maximum flow of G

1 $\alpha \leftarrow \emptyset$

2 $G_\alpha \leftarrow (V, E_0, c_0)$ such that $E_0 = E \cup \{(v,u) : (u,v) \in E\}$ and $c_0(e) = c(e)$ for $e \in E$ and $c_0(e) = 0$ otherwise

3 **while** *there exists an s-t path p with edge sets $E(p)$ and $\nu(p) = \min_{e\in p} c(e) > 0$ in the graph G_α* **do**

4 $\quad \alpha \leftarrow \alpha \cup \{p\}$

5 $\quad$ For each $(u,v) \in E(p)$, $G_\alpha(u,v) \leftarrow G_\alpha(u,v) - \nu(p)$ and $G_\alpha(v,u) \leftarrow G_\alpha(v,u) + \nu(p)$

6 **end**

Let $|E|$ be the number of edges and $|f|$ be the value of maximum flow. Then, the time complexity of the Ford-Fulkerson algorithm in the time-expanded network is $O(|E||f|)$.

It is worth mentioning that the Ford-Fulkerson algorithm is not incorporated into our main algorithms. Instead, we reference it solely for our submodularity proof. In practical scenarios, we prefer using more efficient maximum flow algorithms like the Dinic algorithm [17, 18] or the Edmonds-Karp algorithm [19].

The next fundamental concept we use in our proof is flow decomposition [20]. The concept is introduced in the following theorem:

Theorem 2.6 (Flow Decomposition Theorem [20]) For any flow f of $G = (V, E, c)$, there are feasible path flow set $\alpha = \{p_1, ..., p_k\}$ such that:

1. For all $p \in \alpha$, $E(p) \subseteq E$.
2. $k \leq |E|$.
3. For all $e \in E$, $f(e) = \sum_{p:e\in E(p)} \nu(p)$.

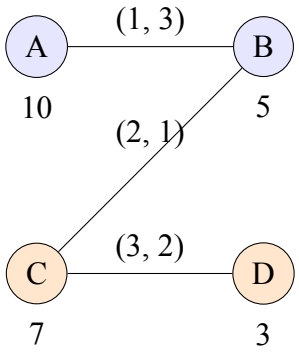

Figure 2.4: An example of a dynamic network in which A, B are non-facility nodes and C, D are facility nodes, with their evacuee number at each node. The pair (a, b) on the edge of the graph represents transit time a with capacity b.

By this theorem, if we have an arbitrary feasible flow, we can decompose it into path flows. The set of path flows will be used at our submodularity proof in the next section.

2.2 Problem Definition

Let $\mathcal{N} = (\mathcal{G}, S, \mathsf{n}, \mathsf{c}, \mathsf{d})$ be the dynamic network such that $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed graph with a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$ where each edge in $\mathcal{E}$ is associated with both directions, meaning that if there is an edge from vertex u to vertex v, there is also a corresponding edge from v to u. Denote $S \subseteq \mathcal{V}$ a set of facilities that are the evacuation centers for evacuees. Let $\mathsf{n} : \mathcal{V} \to \mathbb{Z}_+$ be a function that represents the number of evacuees on each vertex, $\mathsf{c} : \mathcal{E} \to \mathbb{Z}_+$ be a capacity function, and $\mathsf{d} : \mathcal{E} \to \mathbb{Z}_+$ be a transit time function. Let $\mathsf{e} = (\mathsf{v}_1, \mathsf{v}_2)$. The capacity $\mathsf{c}(\mathsf{e})$ represents the maximum number of evacuees which can transit from v_1 to v_2 in one unit time, and the transit time $\mathsf{d}(\mathsf{e})$ represents the amount of time that evacuee transit from v_1 to v_2.

To understand the problem formulation, let us consider an example. Let $\mathcal{V} = \{\mathsf{u}, \mathsf{v}\}$, $\mathcal{E} = \{(\mathsf{u}, \mathsf{v})\}$, and $S = \{\mathsf{u}\}$. To calculate the minimum evacuation time from u to v, we divide the evacuees into $\lceil \mathsf{n}(\mathsf{u})/\mathsf{c}(\mathsf{e}) \rceil$ groups such that each group has the number of evacuees equals to $\mathsf{c}(\mathsf{e})$ except for the last group which has the number of evacuees at most $\mathsf{c}(\mathsf{e})$. After that, we send each group of evacuees from u to v, which means the first group will arrive v at $\mathsf{t} = \mathsf{d}(\mathsf{e})$ and the last group will arrive v at $\mathsf{t} = \mathsf{d}(\mathsf{e}) + \lceil \mathsf{n}(\mathsf{u})/\mathsf{c}(\mathsf{e}) \rceil - 1$.

When there are more nodes in $\mathcal{G}$, there can be a congestion. If there are evacuees from other nodes to u and there are still evacuees in u. The latter evacuees must wait until the earlier evacuees have been evacuated.

Define $\mathsf{flow}_t : 2^{|\mathcal{V}|} \to \mathbb{Z}_+$ as the function that represents the count of evacuees that can be accommodated by facilities positioned at a set of nodes, S, in time t. In this study, we will

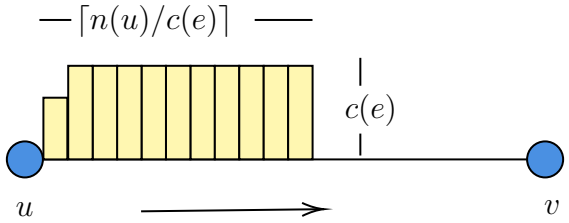

Figure 2.5: This illustration shows how to evacuate the evacuees from u to v.

consider two facility location problems in the dynamic network $\mathcal{N}$.

Problem 1 Consider a facility location problem which aims to evacuate the maximum number of persons in the given amount of time. In particular, given $t, \kappa \in \mathbb{Z}_+$, we give an algorithm which outputs $S \subseteq \mathcal{V}$ such that $|S| = \kappa$ and $\mathsf{flow}_t(S)$ is maximized.

Problem 2 Consider a facility location problem which aims to minimize the number of facilities such that all of the evacuees can evacuate in the given amount of time. In particular, given $t, n \in \mathbb{Z}_+$ when n is the number of evacuees, we give an algorithm which outputs $S \subseteq \mathcal{V}$ such that $\mathsf{flow}_t(S) = n$ and $|S|$ is minimized.

To calculate the maximum number of evacuees whose evacuation time is less than or equal to t, time-expanded network $G_t(S)$, a static flow network for a dynamic flow network, was first proposed by Ford and Fulkerson [16]. The vertices of $G_t(S)$ are divided into three parts. The first part is x^* and ζ^*, which is the source node and the sink node, respectively. The second part is $v(t')$ for $v \in \mathcal{V}$ and $t' \in \{0, 1, 2, ...t\}$, and the last part is u^* for $u \in S$. Furthermore, the edges of $G_t(S)$ are separated into five parts as follows. The first part is (x^*, v) for $v \in \mathcal{V}$ with a capacity $\mathsf{n}(v)$. The second part is $(v(t), v(t+1))$ for $v \in \mathcal{V}$ and $t' \in \{0, 1, 2, ..., t-1\}$ with infinite capacity. If there is edge $e = (v_1, v_2) \in \mathcal{E}$, then there are edges $(v_1(t'), v_2(t' + \mathsf{d}(e)))$ with a capacity $\mathsf{c}(e)$ for $t' \in \{0, 1, 2, ..., t - \mathsf{d}(e)\}$ which is the third part of edges in $G_t(S)$. The fourth part is $(u(t'), u^*)$ with infinite capacity for $u \in S$ and $t' \in \{0, 1, 2, ..., t\}$. The last part is (n^*, ζ^*) for $n \in S$ with infinite capacity. Figure 2.6 shows an example of the correspondence between a given static graph and its time-expanded network.

It is easy to observe the relationship between the maximum number of nodes and the maximum flow, as concluded in the following proposition. The correctness of this proposition is straightforward from [4].

Proposition 2.7 Given a dynamic flow network $G_t(S)$ and a time horizon t, let $\mathsf{flow}_t(S)$ be the value of the maximum flow from x^* to ζ^* in $G_t(S)$. The maximum number of evacuees whose evacuation time is less than or equal to t is equal to $\mathsf{flow}_t(S)$.

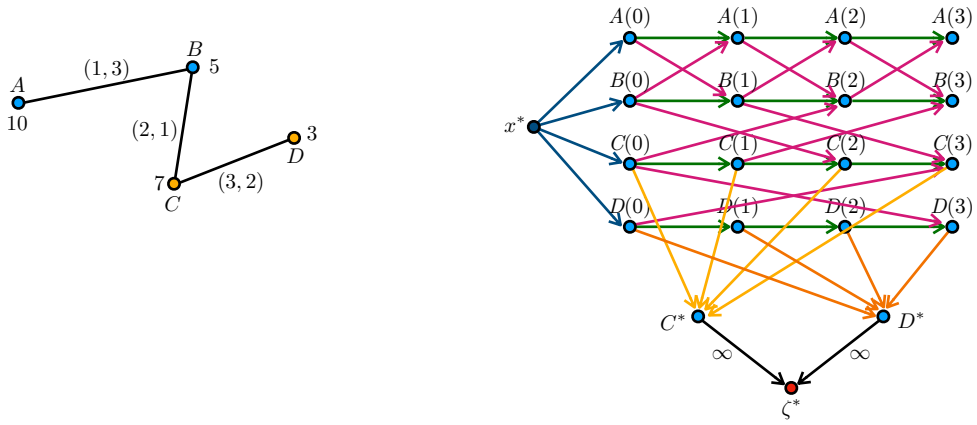

Figure 2.6: (left) An example of a dynamic graph in which A, B are source nodes and C, D are sink nodes, with their evacuee number at each node. The pair (a, b) on the edge of the graph represents transit time a with capacity b. (right) the static network representing each node at the time i, for $i = 1, ..., t$.

2.3 Approximation Algorithms

Approximation algorithms play a crucial role in solving complex optimization problems where finding an exact solution is computationally infeasible or impractical. These algorithms provide efficient and effective ways to approximate optimal solutions, often sacrificing perfect accuracy for computational tractability. In the realm of computer science and mathematics, approximation algorithms have become indispensable tools for addressing computationally intensive problems. The definition of an approximation algorithm is as shown below:

Definition 2.8 [21] An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that produces a solution whose value is within a factor of α of the value of an optimal solution.

For an α-approximation algorithm, α is the approximation ratio of the algorithm. The approximation ratio is a key metric used to evaluate the performance of approximation algorithms. It quantifies how close the solution provided by an approximation algorithm is to the optimal solution for a given problem instance. For example, a 0.5-approximation algorithm for a maximization problem is a polynomial time algorithm that always returns a solution whose value is at least half the optimal value.

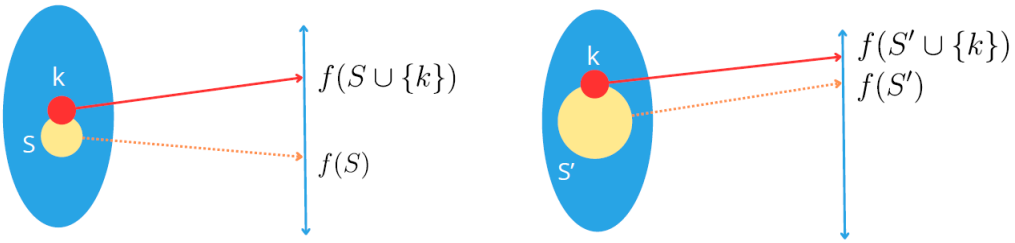

Figure 2.7: The picture shows the examples of sets S and S' that satisfy the inequality of submodular functions.

2.4 Submodularity of Functions flow_t

A submodular function is a mathematical function defined on finite sets satisfying the property that, when adding an element to a smaller set, the difference in value will be greater than or equal to the difference in value when adding it to a larger set, as shown in the following definition [22].

Definition 2.9 If V is a finite set, a function $f : 2^{|V|} \to \mathbb{R}$ is submodular if for every $S, S' \subseteq V$ with $S \subseteq S'$ and for every $k \in V - S'$, $f(S \cup \{k\}) - f(S) \geq f(S' \cup \{k\}) - f(S')$.

Submodular functions can be classified into a class of monotone functions and non-monotone functions. In this study, we will mainly focus on monotone functions, a function in which the value of a smaller set is less than or equal to that of a larger set.

Definition 2.10 A set function f is monotone if for every $S \subseteq S'$, then $f(S) \leq f(S')$.

In optimization, monotone submodular functions have a considerable advantage since their properties can guarantee that the greedy algorithm is an efficient approximation algorithm. In a computationally efficient way, these algorithms can give solutions that are close to the optimal solution with a provable ratio between the solution from the algorithm and the optimal solution.

In this paper, we show that flow_t is a submodular function. It is clear that flow_t is a monotone function since flow_t is the function of maximum flow in a time-expanded network; it is obvious that when the set of sink nodes is larger, the maximum flow will increase. As a result, we can use the algorithm in [11] to give a 0.63-approximation algorithm for Problem 1. The algorithm is shown in Algorithm 2.

Furthermore, we can solve Problem 2 by an $O(\log(n))$-approximation algorithm when n is the total number of evacuees. The algorithm is shown in Algorithm 3.

Algorithm 2: Greedy algorithm for Problem 1 based on the algorithm for the submodular function maximization problem in [11]

Input: The function $\mathsf{flow}_t : 2^{|\mathcal{V}|} \to \mathbb{Z}_{\geq 0}$, the number of facility κ

Output: Set of facility S

$S \leftarrow \emptyset$;

while $|S| < \kappa$ **do**

  $v^* \leftarrow \arg\max_{v \in \mathcal{V}} \mathsf{flow}_t(S \cup \{v\})$

  $S \leftarrow S \cup \{v^*\}$

end

Algorithm 3: Greedy algorithm for Problem 2 based on the algorithm for the submodular cover minimization problem in [12]

Input: The function $\mathsf{flow}_t : 2^{|\mathcal{V}|} \to \mathbb{Z}_{\geq 0}$, the number of evacuees n

Output: Set of facility S

$S \leftarrow \emptyset$;

while $\mathsf{flow}_t(S) < n$ **do**

  $v^* \leftarrow \arg\max_{v \in \mathcal{V}} \mathsf{flow}_t(S \cup \{v\})$

  $S \leftarrow S \cup \{v^*\}$

end

Chapter 3

Main results

3.1 Theoretical Results

We prove the submodularity property of the flow_t function in this section. We denote an inverse of edge $e = (u, v)$ as $\bar{e} = (v, u)$.

Under the earlier defined parameters, it is interesting to note that $E(p)$ might not always be a subset of E, despite p being a path flow of the graph $G = (V, E, c)$. In fact, when applying the Ford-Fulkerson algorithm to determine path flows, the output set of edges may not necessarily be confined within E.

The following definition will focus on a particular instance of path flows that does not incorporate edges in the set $\{\bar{e} : e \in E\}$.

Definition 3.1 Consider a graph $G = (V, E, c)$ such that $\mathsf{s}, \mathsf{t} \in V$. Let α be a set of s-t path flow. We say that α is a **one-sided** s-t path flow set if:

1. for all $p \in \alpha$, $E(p) \subseteq E$, and,
2. for all $e \in \alpha$, $\sum_{p \in \alpha : e \in E(p)} \nu(e) \leq c(e)$.

We can construct a one-sided path flow set from a path flow set using the flow decomposition theorem introduced in the previous section.

Recall the graph $G_t(S)$ defined in the previous section. Let $F(S)$ be the collection of all x^*-ζ^* maximum path flow sets of $G_t(S)$. By the definition, we obtain the following proposition.

Proposition 3.2 For any $S \subseteq S'$ and $\alpha \in F(S)$, there is $\alpha' \in F(S')$ such that $\alpha \subseteq \alpha'$.

Proof. Let $G_\alpha = (V, E_\alpha, c)$ be a residual graph obtained from the graph $G_t(S)$ and the path flow set α. Let $E' = E_\alpha \cup \{(v^*, \zeta^*) : v \in S' \backslash S\}$. Consider a function $c' : E' \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ such that $c'(e) = c(e)$ for all $e \in E_\alpha$ and $c'(e) = \infty$ for all $e \in \{(v^*, \zeta^*) : v \in S' \backslash S\}$. We can apply the Ford-Fulkerson algorithm to the graph $G' = (V, E', c')$. Let β be the set of path flows obtained from the Ford-Fulkerson algorithm. It is straightforward to confirm that $\alpha' = \alpha \cup \beta$ is an x^*-ζ^* maximum path flow set of $G(S')$. Hence, $\alpha' \in F(S')$. □

For each $\alpha \in F(S)$, let $G_\alpha = (V, E_\alpha, c)$ be a residual graph obtained from the graph $G_t(S)$ and the path flow set α. For each $k \in V$, let $E_k^* = E_\alpha \cup \{(k^*, \zeta^*)\}$, $c_k^*(e) = c(e)$ for all

$e \in E_\alpha$, $c_k^*((k^*, \zeta^*)) = \infty$, and $G_k^* = (V, E_k^*, c^*)$. We use the previous proposition to prove the subsequent lemma.

Lemma 3.3 For any $\alpha \in F(S)$, there is $\alpha' \in F(S \cup \{k\})$ such that $\alpha \subseteq \alpha'$, and for all $p \in \alpha' \backslash \alpha$, $k^* \in E(p)$. Furthermore, $\alpha' \backslash \alpha$ is a maximum path flow set of G_k^*.

Proof. Let $\alpha \in F(S)$. By Proposition 3.3, there exists $\alpha' \in F(S \cup \{k\})$ such that $\alpha \subseteq \alpha'$. It is straightforward that $\alpha' \backslash \alpha$ is a maximum path flow set of the residual graph G_k^*.

We then show that, for all $p \in \alpha' \backslash \alpha$, $k^* \in E(p)$ by contradiction. Let assume that $\beta = \alpha' \backslash \alpha$, and there is a path $p \in \beta$ such that $k^* \notin E(p)$. Then there exists $s \in S$ such that $s^* \in E(p)$. Then let us consider β as an x^*-ζ^* maximum path flow set of G_k^*. By the flow decomposition, we can construct a one-side path flow set of G_k^* from β. Let us denote that one-side path flow set by β'. Since there exists $p \in \beta$ such that $s^* \in E(p)$, there exists $p' \in \beta'$ such that $s^* \in E(p')$. We obtain that p' is a path in G_k^*, which means $\alpha \cup \{p'\}$ is a feasible path flow set in $G_t(S)$ with a flow value larger than the flow value of path flow set α. This contradicts to the fact that $\alpha \in F(S)$. □

We are ready to prove the following theorem which confirms the submodularity for the considered function flow_t.

Theorem 3.4 Let $S \subseteq S'$. Then $\mathsf{flow}_t(S \cup \{k\}) - \mathsf{flow}_t(S) \geq \mathsf{flow}_t(S' \cup \{k\}) - \mathsf{flow}_t(S')$.

Proof. Let $\alpha \in F(S)$. By Proposition 3.2, there are $\alpha' \in F(S')$ and $\alpha'' \in F(S' \cup \{k\})$ such that $\alpha \subseteq \alpha' \subseteq \alpha''$. For all $p \in \alpha'' - \alpha'$, we can assume from Lemma 3.3 that $k^* \in p$. By $\mathsf{flow}_t(S' \cup \{k\}) - \mathsf{flow}_t(S') = \sum_{p \in \alpha'' - \alpha'} \nu(p)$, we obtain that $\mathsf{flow}_t(S' \cup \{k\}) - \mathsf{flow}_t(S')$ is equal to the flow value at edge (k^*, ζ^*) in α''.

Let $E_{sk}^* = E_\alpha \cup \{(s^*, \zeta^*) : s \in S' \cup \{k\}\}$, $c_{sk}^*(e) = c(e)$ for all $e \in E_\alpha$, $c_{sk}^*((s^*, \zeta^*)) = \infty$ for all $s \in S' \cup \{k\}$, and $G_{sk}^* = (V, E_{sk}^*, c^*)$. Let $\beta = \alpha'' - \alpha$. We can consider β as an x^*-ζ^* maximum path flow set of G_{sk}^*. By the flow decomposition, we can construct a one-side path flow set of G_{sk}^* from β. Let us denote that one-side path flow set by β'. Also, let us denote β_k' as a set of path flows in β which pass k^*, that is $\beta_k' := \{p \in \beta' : (k^*, \zeta^*) \in E(p)\}$. The flow at the edge (k^*, ζ^*) in α'' is equal to $\sum_{p \in \beta_k'} \nu(p)$, and, by the previous paragraph, $\sum_{p \in \beta_k'} \nu(p) = \mathsf{flow}_t(S' \cup \{k\}) - \mathsf{flow}_t(S')$.

The path flow set β_k' is a path flow set of G_{sk}^* because, for all $e \in G_{sk}^*$, $\sum_{p \in \beta_k' : e \in p} \nu(p) \leq \sum_{p \in \beta' : e \in p} \nu(p) \leq c(e)$. The path flow set $\alpha \cup \beta_k'$ is then a path flow set of $G(S \cup \{k\})$. Hence, the maximum flow value of $G(S \cup \{k\})$ is at least $\sum_{p \in \alpha} \nu(p) + \sum_{p \in \beta_k'} \nu(p)$. We obtain that:

$$\begin{aligned}\mathsf{flow}_t(S \cup \{k\}) - \mathsf{flow}_t(S) &\geq \sum_{p\in\alpha} \nu(p) + \sum_{p\in\beta'_k} \nu(p) - \sum_{p\in\alpha} \nu(p) \\ &= \sum_{p\in\beta'_k} \nu(p) \\ &= \mathsf{flow}_t(S' \cup \{k\}) - \mathsf{flow}_t(S')\end{aligned} \tag{3.1}$$

□

Since flow_t is a monotone function and we show that flow_t is a submodular function, Algorithm 2 and Algorithm 3 becomes approximation algorithms for Problem 1 and Problem 2 with approximation ratios 0.63 and $O(\log(n))$, respectively.

3.2 Experimental Results

3.2.1 Data

An example for verifying the proposed method is derived from a graph of a road network extracted from the city of Chiang Mai, generated by the open data from the project "Urban Observatory and Citizen Engagement by Data-driven and Deliberative Design: A Case Study of Chiang Mai City". The information on the roads (road width and length) and population number is stored as .csv file, which can be opened using QGIS software.

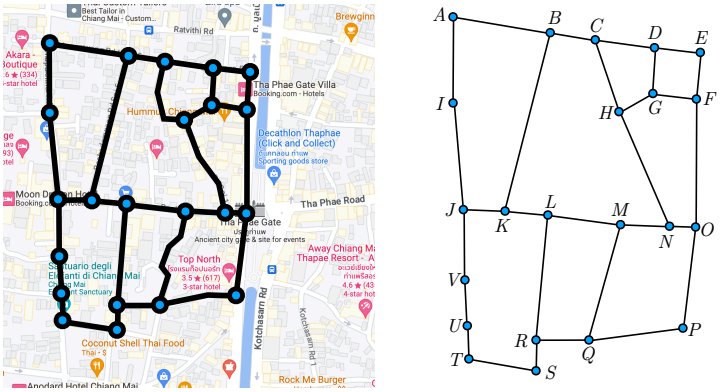

Figure 3.1: (left) An example of a selected network from a map of Chiang Mai (from Google Map) (right) the planar graph generated from the map such that the nodes are derived from the intersections of roads, and edges are roads.

3.2.2 Data Extraction

Based on the provided information, the graph nodes represent the intersection of roads. The capacity of each edge is interpreted as two times the width of the road, and the transit time is

computed from the length of the road.

Since the population number information is stored as the population number per district, the following instruction illustrates the assignment of the population to each node of the graph. Assume that the considered region consisting of $D_1, ..., D_n$ districts with population number $n(D_1), ..., n(D_n)$, respectively.

1. Generate the ordinary Voronoi diagram which generators are the graph nodes over the considered region.

2. For each Voronoi region, consider whether it belongs to district(s). Then compute the area of each district contained within each Voronoi polygon.

   Suppose that $V(v)$ is the Voronoi region of the node v such that $V(v) = V_1(v) \cup ... \cup V_p(v)$ and $V_i(v) \subseteq D_k$ for some k

3. The number of population at node v is computed by

$$n(v) = \sum_i \frac{\text{Area}(V_i(v) \cap D_k)}{\text{Area}(D_k)} \times n(D_k).$$

3.2.3 Results

To set up experiments, we use a graph of a road network with 22 nodes, where the total number of evacuees is 1455, and 30 edges. We assume that unit time in a time-expanded network is 3 seconds; in one meter of road width, two evacuees can evacuate, and in 3 seconds, evacuees can evacuate 2 meters. The experiments were done for both problems as follows.

Experiment Results for Problem 1

The objective of the experiments in problem 1 is to find the facilities' location by Algorithm 2 among the given graph with $2, 3, 4$, and 5 facilities such that the facilities in each case will be located on the node of the given graph. Furthermore, we can find the optimal facility location by considering all possible $C_{n,k}$ cases. In this section, we will show the facility location from Algorithm 2 and the optimal facility location with the number of evacuees whose evacuation time is less than or equal to 3 minutes. The results are shown in Table 3.1 with Figure 3.2 and 3.3.

The experiment result shows that, in the case that the number of facilities is 2, the set of facilities from the greedy algorithm is the same as the optimal solution. On the other hand, when the number of facilities is 3, 4, and 5, the solution from the greedy algorithm is slightly different from the optimal solution because the greedy algorithm may not guarantee that the solution from the algorithm will be the same as the optimal solution.

Table 3.1: The table of the result from Algorithm 2 in Problem 1. The table includes the set of facility locations, and the number of evacuees whose evacuation time is less than 3 minutes, comparing to the optimal facility location set by considering all of the possible $C_{22,k}$

Facilities No. (k)	**Result from Algorithm**		**Result from Enumeration $C_{n,k}$**	
	Set of Nodes	**No. Evacuees**	**Set of Nodes**	**No. Evacuees**
2	$[I, K]$	625	$[I, K]$	625
3	$[I, K, T]$	826	$[I, L, V]$	870
4	$[I, K, T, C]$	983	$[I, L, V, C]$	1027
5	$[I, K, T, C, G]$	1115	$[I, L, V, B, G]$	1159

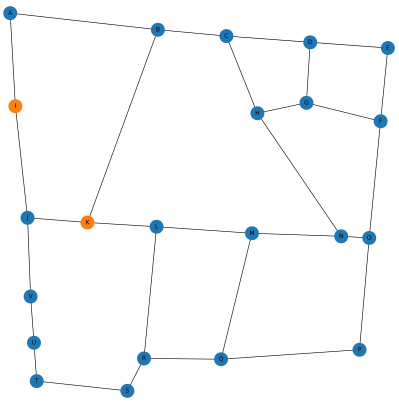

Figure 3.2: Result of facility locations from Algorithm 2 showed by orange nodes, which is the same location with and the optimal facility location by considering all possible locations in the case of two facilities.

Experiment Result for Problem 2

In Problem 2, we use Algorithm 3 with the same data set as Problem 1 to find the set of facilities S such that the number of facilities is minimized and the evacuation time of all evacuees is less than or equal to 5 minutes. It is worth noting that, in the minimization problem, we do not fix the number of facilities but aim to minimize it.

The result from the experiment shows that the number of optimal facilities in this data is equal to 4 when we employ Algorithm 3, in which the set of facility locations is $[B, F, J, Q]$. This satisfies the optimal solution acquired from enumerating all possible locations, as shown in Figure 3.4.

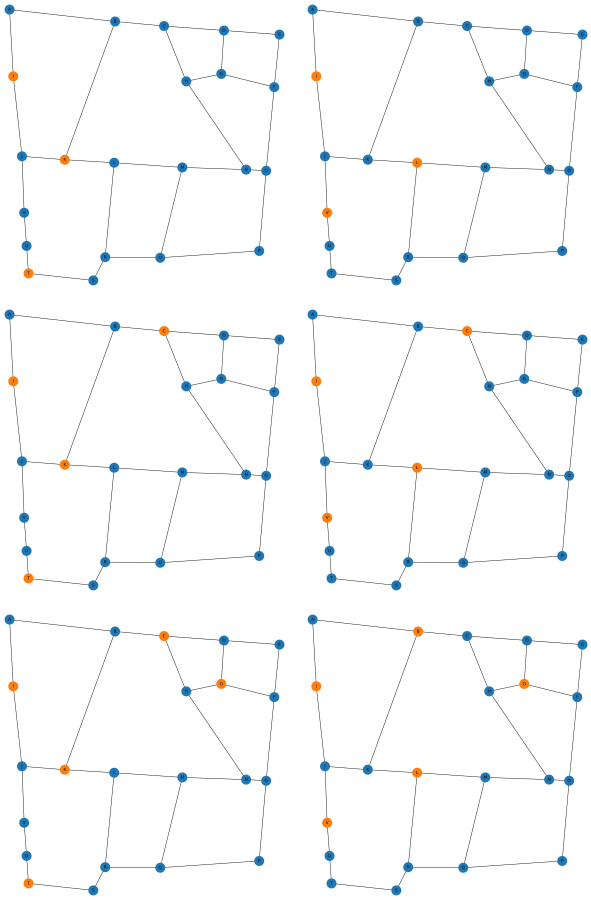

Figure 3.3: (left) Results of facility locations from Algorithm 2 with 3, 4, and 5 facilities (right) results of optimal facility locations by considering all of the possible locations with 3, 4, and 5 facilities. Orange nodes show the locations of facilities.

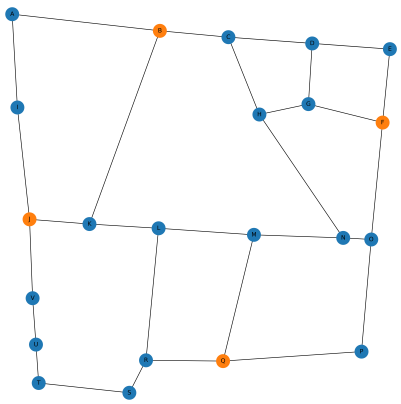

Figure 3.4: The facility locations (orange nodes) from Algorithm 3, which is the same location as the optimal facility location by considering all possible locations

Chapter 4

Conclusion

In this study, we proposed the proof of the submodularity of the function flow_t, which is defined by the maximum flow of a time-expanded network with a given time t from a static graph, which is the function that represents the number of evacuees whose evacuation time is less than or equal to t. This property enables us to apply greedy algorithms for solving the facility location problem of dynamic flow networks by finding the locations that maximize the number of evacuees whose evacuation time is less than or equal to 3 minutes, and the location where the number of them is minimized, making every evacuee evacuate within 5 minutes. We also found the minimum number of facilities such that all evacuees can evacuate within the given time. The experimental results for Problem 1 in the case of 2, 3, 4, and 5 facilities, and Problem 2, showed practical examples with spatial data. Since the effectiveness of greedy algorithms are guaranteed by the submodularity proof, applying the greedy algorithms to real data on larger dynamic flow networks is reasonable.

Based on the proof of submodularity, developing an efficient and robust approximation algorithm for solving the facilities location problem with a larger network is challenging. It would be useful for planning purposes, especially in the evacuation due to disasters in the near future.

We have created a real dataset for evacuation plan in Chiang Mai and have tested with the dataset. However, as a future work, we are planning to conduct more experiments with other datasets including the datasets with larger sizes.

References

[1] Dorit S. Hochbaum, *Heuristics for the fixed cost median problem*, *Mathematical programming*, volume 22, pages 148–162, 1982, Springer.

[2] Maxim Sviridenko, *An improved approximation algorithm for the metric uncapacitated facility location problem*, in *International Conference on Integer Programming and Combinatorial Optimization*, pages 240–257, 2002, Springer.

[3] Fabián A. Chudak, David B. Shmoys, *Improved approximation algorithms for the uncapacitated facility location problem*, *SIAM Journal on Computing*, volume 33, number 1, pages 1–25, 2003, SIAM.

[4] Lester R. Ford Jr, Delbert Ray Fulkerson, *Constructing maximal dynamic flows from static flows*, *Operations research*, volume 6, number 3, pages 419–433, 1958, INFORMS.

[5] Norie Fu, Vorapong Suppakitpaisarn, *Clustering 1-dimensional periodic network using betweenness centrality*, *Computational Social Networks*, volume 3, number 1, pages 1–20, 2016, Springer.

[6] Yuya Higashikawa, Mordecai J. Golin, Naoki Katoh, *Multiple sink location problems in dynamic path networks*, *Theoretical Computer Science*, volume 607, pages 2–15, 2015.

[7] Satoko Mamada, Takeaki Uno, Kazuhisa Makino, Satoru Fujishige, *An $O(n \log^2 n)$ algorithm for the optimal sink location problem in dynamic tree networks*, *Discrete Applied Mathematics*, volume 154, number 16, pages 2387–2401, 2006, Elsevier.

[8] Rémy Belmonte, Yuya Higashikawa, Naoki Katoh, Yoshio Okamoto, *Polynomial-time approximability of the k-sink location problem*, *arXiv preprint arXiv:1503.02835*, 2015.

[9] Yuya Higashikawa, Naoki Katoh, *A survey on facility location problems in dynamic flow networks*, *The Review of Socionetwork Strategies*, volume 13, pages 163–208, 2019, Springer.

[10] Jorge Qüense, Carolina Martínez, Jorge León, Rafael Aránguiz, Simón Inzunza, Nikole Guerrero, Alondra Chamorro, Malcom Bonet, *Land cover and potential for tsunami evacuation in rapidly growing urban areas. The case of Boca Sur (San Pedro de la Paz, Chile)*, *International Journal of Disaster Risk Reduction*, volume 69, page 102747, 2022, Elsevier.

[11] George L. Nemhauser, Laurence A. Wolsey, Marshall L. Fisher, *An analysis of approximations for maximizing submodular set functions—I*, *Mathematical programming*, volume 14, pages 265–294, 1978, Springer.

[12] Laurence A. Wolsey, *An analysis of the greedy algorithm for the submodular set covering problem*, *Combinatorica*, volume 2, number 4, pages 385–393, 1982, Springer.

[13] Narsingh Deo, *Graph Theory with Applications to Engineering and Computer Science*, Dover Publications, 1974.

[14] Robin J. Wilson, *Introduction to Graph Theory*, Prentice Hall, 1996.

[15] Gary Chartrand, Ping Zhang, *A First Course in Graph Theory*, Courier Corporation, 2012.

[16] Lester Randolph Ford, Delbert R. Fulkerson, *Maximal flow through a network*, *Canadian Journal of Mathematics*, volume 8, pages 399–404, 1956, Cambridge University Press.

[17] Yefim A. Dinitz, *Dinitz' algorithm: The original version and Even's version*, in *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 218–240, 2006, Springer.

[18] Yefim A. Dinitz, *An algorithm for the solution of the problem of maximal flow in a network with power estimation*, in *Doklady Akademii nauk*, volume 194, number 4, pages 754–757, 1970, Russian Academy of Sciences.

[19] Jack Edmonds and Richard M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, *Journal of the ACM (JACM)*, volume 19, number 2, pages 248–264, 1972, ACM New York, NY, USA.

[20] David P. Williamson, *Network Flow Algorithms*, Cambridge University Press, 2019.

[21] David P. Williamson, David B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

[22] Satoru Fujishige, *Submodular functions and optimization*, 2005, Elsevier.

Appendix

In this section, we show the Python code that we have used in the experiments.

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import math
```

```
edges_df = pd.read_csv("edges_data.csv")
nodes_df = pd.read_csv("nodes_data.csv")
```

```
def time_expanded(time, G, sink_nodes):
  from numpy import inf
  source_nodes = [g for g in G if g not in sink_nodes]
  te_G = nx.DiGraph()
  for t in range(time+1):
    for i in range(len(list(G))):
      te_G.add_node((str(list(G)[i]) + '(' + str(t) + ')'), pos = (t, len(list(G))-i))

    for k in range(len(list(te_G))-(len(source_nodes) + len(sink_nodes))):
      te_G.add_edge(str(list(te_G)[k]), str(list(te_G)[k + len(source_nodes) + len(sink_nodes)]), capacity = math.inf)
  te_G.add_node('x*', pos = (-1, len(source_nodes)))
  for i in range(len(sink_nodes)):
    te_G.add_node(str(sink_nodes[i]) + '*', pos = ((time+1)*(i+1)/(len(sink_nodes)+1), -len(sink_nodes)))
  for l in range(len(list(G))):
    te_G.add_edge('x*', str(list(te_G)[l]), capacity = nx.get_node_attributes(G, "evacuees")[list(G)[l]])
  te_G.add_node('zeta', pos = ((time+1)/2, -len(sink_nodes)-1))

  for t in range(time+1):
    for e in G.edges():
      weight = nx.get_edge_attributes(G, "weight")[e]
      if t+weight <= time:
        te_G.add_edge(str(list(e)[0]) + '(' + str(t) + ')', str(list(e)[1])
        + '(' + str(t + (nx.get_edge_attributes(G, "weight"))[e]) + ')'
        , capacity = nx.get_edge_attributes(G, "capacity")[e])
        te_G.add_edge(str(list(e)[1]) + '(' + str(t) + ')', str(list(e)[0])
        + '(' + str(t + (nx.get_edge_attributes(G, "weight"))[e]) + ')'
        , capacity = nx.get_edge_attributes(G, "capacity")[e])
    for i in range(len(sink_nodes)):
      te_G.add_edge(str(sink_nodes[i]) + '(' + str(t) + ')', str(sink_nodes[i]) + '*', capacity = math.inf)
  for i in range(len(sink_nodes)):
    te_G.add_edge(str(sink_nodes[i]) + '*', 'zeta', capacity = math.inf)
  return(te_G)
```

```
def flow_te(time, G, sink_nodes):
  flow_value = nx.maximum_flow_value(time_expanded(time, G, sink_nodes), "x*", "zeta")
  return(flow_value)
```

```
G = nx.Graph()

nodes = nodes_df['Node'].to_list()
G.add_nodes_from(nodes)
nodes_x = nodes_df['x'].to_list()
nodes_y = nodes_df['y'].to_list()
evacuees = nodes_df['Evacuees'].to_list()
nodes_attrs = {}
for i in range(len(nodes)):
  nodes_attrs[nodes[i]] = {'pos' : (nodes_x[i], nodes_y[i]), 'evacuees' : evacuees[i]}
nx.set_node_attributes(G, nodes_attrs)

edges = edges_df[['origin_id', 'destination']].values.tolist()
G.add_edges_from(edges)
weight = edges_df['total_cost'].to_list()
capacity = edges_df['width'].to_list()
edges_attrs = {}
for i in range(len(edges)):
  edges_attrs[tuple(edges[i])] = {'weight' : int(round(weight[i]/2,0)), 'capacity' : capacity[i]*2}
nx.set_edge_attributes(G, edges_attrs)
```

```
from itertools import combinations
n = int(input("n = "))
nodes = nodes_df['Node'].to_list()
comb = combinations(nodes, n)
x = []
evacuees_list = []

for c in list(comb):
  x.append(list(c))
  sink_nodes = list(c)
  source_nodes = [g for g in G if g not in sink_nodes]
  max_evacuees = flow_te(60, G, sink_nodes)
  evacuees_list.append(max_evacuees)

pos = nx.get_node_attributes(G,'pos')
sink_nodes = list(x[np.array(evacuees_list).argmax()])
source_nodes = [g for g in G if g not in sink_nodes]
nx.draw_networkx_nodes(G, pos, nodelist = source_nodes, node_size = 500)
nx.draw_networkx_nodes(G, pos, nodelist = sink_nodes, node_size = 500, node_color="tab:orange")
nx.draw_networkx_edges(G, pos, width = 1)
nx.draw_networkx_labels(G, pos, font_size = 10, font_family="sans-serif")
edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos, edge_labels, font_size = 10)
ax = plt.gca()
ax.margins(0)
plt.axis("off")
plt.tight_layout()
plt.show()
print(sink_nodes)
print(flow_te(60, G, sink_nodes))
```

```
n = int(input("n = "))
facility_list = []
while len(facility_list) < n:
  max = 0
  for v in list(G):
    if v not in facility_list:
      if max < flow_te(60, G, facility_list + [v]):
        max = flow_te(60, G, facility_list + [v])
        v_max = v
  facility_list.append(v_max)


pos = nx.get_node_attributes(G,'pos')
sink_nodes = facility_list
source_nodes = [g for g in G if g not in sink_nodes]
nx.draw_networkx_nodes(G, pos, nodelist = source_nodes, node_size = 500)
nx.draw_networkx_nodes(G, pos, nodelist = sink_nodes, node_size = 500, node_color="tab:orange")
nx.draw_networkx_edges(G, pos, width = 1)
nx.draw_networkx_labels(G, pos, font_size = 10, font_family="sans-serif")
edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos, edge_labels, font_size = 10)
ax = plt.gca()
ax.margins(0)
plt.axis("off")
plt.tight_layout()
plt.show()
print(facility_list)
print(flow_te(60, G, facility_list))
```

```
from itertools import combinations
n = int(input("n = "))
nodes = nodes_df['Node'].to_list()
comb = combinations(nodes, n)
x = []
evacuees_list = []

for c in list(comb):
  x.append(list(c))
  sink_nodes = list(c)
  source_nodes = [g for g in G if g not in sink_nodes]
  max_evacuees = flow_te(100, G, sink_nodes)
  evacuees_list.append(max_evacuees)

pos = nx.get_node_attributes(G,'pos')
sink_nodes = list(x[np.array(evacuees_list).argmax()])
source_nodes = [g for g in G if g not in sink_nodes]
nx.draw_networkx_nodes(G, pos, nodelist = source_nodes, node_size = 500)
nx.draw_networkx_nodes(G, pos, nodelist = sink_nodes, node_size = 500, node_color="tab:orange")
nx.draw_networkx_edges(G, pos, width = 1)
nx.draw_networkx_labels(G, pos, font_size = 10, font_family="sans-serif")
edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos, edge_labels, font_size = 10)
ax = plt.gca()
ax.margins(0)
plt.axis("off")
plt.tight_layout()
plt.show()
print(sink_nodes)
print(flow_te(100, G, sink_nodes))
```

```
facility_list = []
while flow_te(100, G, facility_list) < sum(nodes_df['Evacuees']):
  max = 0
  for v in list(G):
    if v not in facility_list:
      if max < flow_te(100, G, facility_list + [v]):
        max = flow_te(100, G, facility_list + [v])
        v_max = v
  facility_list.append(v_max)
  print(facility_list)
  print(max)


pos = nx.get_node_attributes(G,'pos')
sink_nodes = facility_list
source_nodes = [g for g in G if g not in sink_nodes]
nx.draw_networkx_nodes(G, pos, nodelist = source_nodes, node_size = 500)
nx.draw_networkx_nodes(G, pos, nodelist = sink_nodes, node_size = 500, node_color="tab:orange")
nx.draw_networkx_edges(G, pos, width = 1)
nx.draw_networkx_labels(G, pos, font_size = 10, font_family="sans-serif")
edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos, edge_labels, font_size = 10)
ax = plt.gca()
ax.margins(0)
plt.axis("off")
plt.tight_layout()
plt.show()
print(facility_list)
print(flow_te(100, G, facility_list))
```

Submodularity Property for Facility Locations of Dynamic Flow Networks

Peerawit Suriya, and Supanut Chaidee

Department of Mathematics, Faculty of Science, Chiang Mai University

ABSTRACT

This paper considers facility location problems within dynamic flow networks, shifting the focus from minimizing evacuation time to handling situations with a constrained evacuation timeframe. Our study sets two main goals: 1) Determining a fixed-size set of locations that can maximize the number of evacuees, and 2) Identifying the smallest set of locations capable of accommodating all evacuees within the time constraint. We introduce $\mathsf{flow}_t(S)$ to represent the number of evacuees for given locations S within a fixed time limit t. We prove that flow_t functions is a monotone submodular function, which allows us to apply an approximation algorithm specifically designed for maximizing such functions with size restrictions. For the second objective, we implement an approximation algorithm tailored to solving the submodular cover problem. We conduct experiments on the real datasets of Chiang Mai, and demonstrate that the approximation algorithms give solutions which are close to optimal for all of the experiments we have conducted.

INTRODUCTION

The facilities location problem is one of the well-known problems for finding the optimal location of facilities that optimizes certain criteria under the given constraints and considerations. Extending the problem formulation of facility location to a dynamic network is natural. While the majority of prior research has centered around minimizing evacuation time, we argue that real-world evacuations often operate under strong time constraints.

PROBLEM DEFINITION

Let $\mathcal{N} = (\mathcal{G}, S, \mathsf{n}, \mathsf{c}, \mathsf{d})$ be the network where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph with a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$.

- $S \subseteq \mathcal{V}$ is a set of facilities that are the evacuation centers.
- Each vertex v has the number of evacuees $\mathsf{n}(\mathsf{v})$.
- Each edge $\mathsf{e} = (\mathsf{v}_1, \mathsf{v}_2)$ consists of a capacity $\mathsf{c}(\mathsf{e})$ and a transit time $\mathsf{d}(\mathsf{e})$.
- The transit time $\mathsf{d}(\mathsf{e})$ represents the amount of time that evacuee transit from v_1 to v_2.
- The capacity $\mathsf{c}(\mathsf{e})$ represents the maximum number of evacuees which can transit from v_1 to v_2 in one unit time.

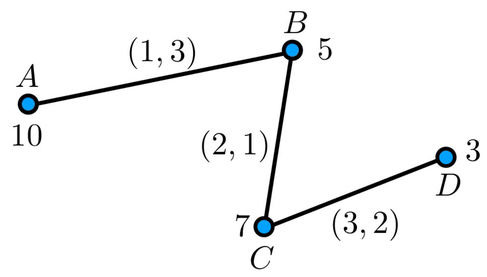

Figure 1: *An example of a dynamic graph with its evacuee number at each node. The pair (a, b) on the edge represents transit time a with capacity b.*

Define $\mathsf{flow}_t(S)$ as the count of evacuees that can be accommodated by facilities positioned at a set of nodes, S, in time t.

In this study, we will consider two facility location problems in the dynamic network $\mathcal{N}$.

Problem 1 Consider a facility location problem which aims to evacuate the maximum number of persons in the given amount of time.

Problem 2 Consider a facility location problem which aims to minimize the number of facilities such that all of the evacuees can evacuate in the given amount of time.

SUBMODULARITY OF FUNCTIONS

A submodular function is a mathematical function defined on finite sets satisfying the property that, when adding an element to a smaller set, the difference in value will be greater than or equal to the difference in value when adding it to a larger set.

Definition 1. *If V is a finite set, a function $f : 2^V \to \mathbb{R}$ is submodular if every $S, S' \subseteq V$ with $S \subseteq S'$ and every $k \in V - S'$ then $f(S \cup \{k\}) - f(S) \geq f(S' \cup \{k\}) - f(S')$.*

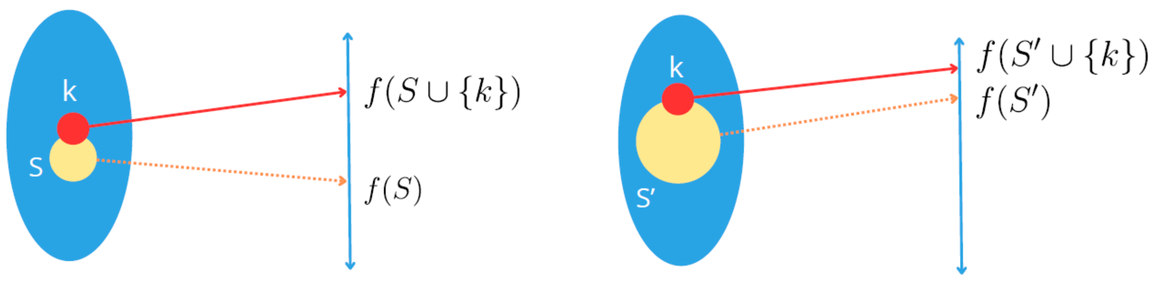

Figure 2: *Sets S and S' that satisfy the inequation.*

To calculate the flow_t, time-expanded network $G_t(S)$, a static flow network for a dynamic flow network, was first proposed by Ford and Fulkerson. It is easy to observe the relationship between the maximum number of nodes and the maximum flow, as concluded in the following proposition.

Proposition 1. *Given a dynamic flow network $G_t(S)$ and a time horizon t, let $\mathsf{flow}_t(S)$ be the value of the maximum flow from x^* to ζ^* in $G_t(S)$. The maximum number of evacuees whose evacuation time is less than or equal to t is equal to $\mathsf{flow}_t(S)$.*

We show that flow_t is a submodular function. It is clear that flow_t is a monotone function. As a result, the greedy algorithm can be deployed, delivering a $(1 - 1/e)$-approximation algorithm for Problem 1. On the other hand, we can employ the $O(\log n)$-approximation algorithm for the problem to solve Problem 2.

Theorem Let $S \subseteq S'$, then

$$\mathsf{flow}_t(S \cup \{k\}) - \mathsf{flow}_t(S) \geq \mathsf{flow}_t(S' \cup \{k\}) - \mathsf{flow}_t(S')$$

To prove this theorem, we first formulate the dynamic network into a time-expanded network. Then, from proposition 1, we consider the maximum flow of $G_t(S)$, $G_t(S')$, $G_t(S \cup k\})$, and $G_t(S' \cup k\})$. Finally, we use the main idea of the Ford-Fulkerson algorithm and the flow decomposition theorem to prove that flow_t is a submodular function.

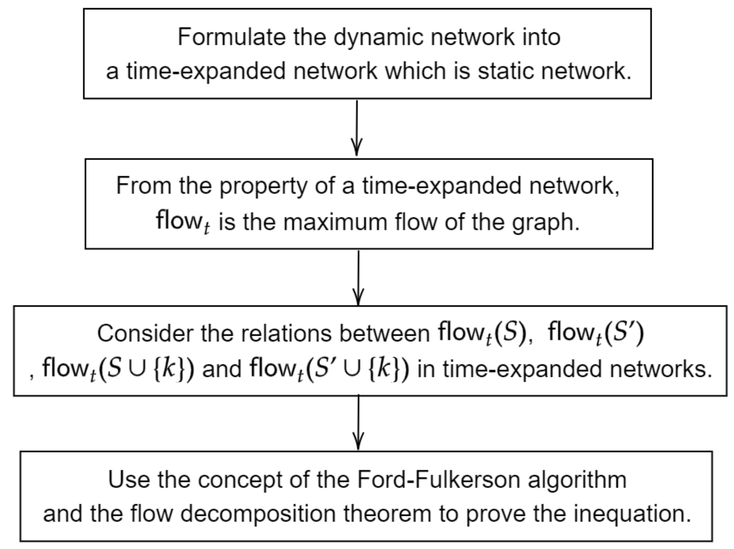

Figure 3: *The diagram shows the process of the proof.*

EXPERIMENTAL RESULTS

An example for verifying the proposed method is derived from a graph of a road network extracted from the city of Chiang Mai.

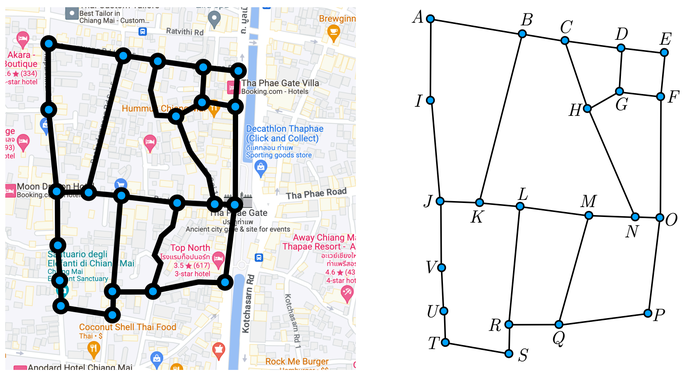

Figure 4: *(left) An example of a selected network from a map of Chiang Mai (right) the planar graph generated from the map.*

Problem 1 We will show the facility location from greedy algorithm and the optimal facility location with the number of evacuees whose evacuation time is less than or equal to 3 minutes.

Table 1: *The table below shows the result from greedy algorithm in Problem 1, compared to the optimal facility location set*

Facilities No.	Result from Algorithm		Result from Enumeration	
	Set of Nodes	No. Evacuees	Set of Nodes	No. Evacuees
2	$[I, K]$	625	$[I, K]$	625
3	$[I, K, T]$	826	$[I, L, V]$	870
4	$[I, K, T, C]$	983	$[I, L, V, C]$	1027
5	$[I, K, T, C, G]$	1115	$[I, L, V, B, G]$	1159

Problem 2 We use greedy algorithm to find the set of facilities S such that the number of facilities is minimized and the evacuation time of all evacuees is less than or equal to 5 minutes.

The result from the experiment shows that the number of optimal facilities in this data is equal to 4 when we employ greedy algorithm, in which the set of facility locations is $[B, F, J, Q]$. This satisfies the optimal solution acquired from enumerating all possible locations.

CONCLUSION

In this study, we proposed the proof of the submodularity of the function flow_t. This property enables us to apply the greedy algorithm to solving problems 1 and 2. Through numerical experimentation, we demonstrate that the algorithms provide solutions that are closely aligned with optimal solutions.

REFERENCES

[1]Higashikawa, Y., & Katoh, N. (2019). A survey on facility location problems in dynamic flow networks. The Review of Socionetwork Strategies, 13, 163-208. Springer.

[2]Hochbaum, D. S. (1982). Heuristics for the fixed cost median problem. Mathematical Programming, 22, 148-162. Springer.

[3]Nemhauser, G. L., Wolsey, L. A., & Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions—I. Mathematical Programming, 14, 265-294. Springer.

[4]Wolsey, L. A. (1982). An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica, 2(4), 385-393. Springer.