UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Gravitational potential method and its application to network optimization

Author:
Oleksa HRYNIV

Supervisor:
Prof. Yurii GOLOVATY

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Gravitational potential method and its application to network optimization

by Oleksa HRYNIV

Abstract

The primary goal of this research is to propose a novel spectral-based method of network optimization, called the method of the gravitational potentials. The problem under discussion is to locate several server-type nodes to optimize the service area distribution and to minimize the total shortest path length for the network nodes. We introduce the quantitative criteria of optimization, explain the intuition behind the method, describe in detail all involved steps and justify the functionality of the algorithm. We also demonstrate the performance of the algorithm on various graphs and compare it with the optimal results. Complete implementation of the algorithm, along with the graph models used in the experiments, can be found on the GitHub repository [1].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's data-driven world, network models and their graph representations have become indispensable tools for analyzing complex systems and predicting their behaviour. Such complex systems are ubiquitous in everyday life, ranging from social or collaboration networks to transportation or computer networks. Although diverse in their nature or origin, all such networks describe some actors and interactions between them and can be modelled by graphs consisting of nodes (also called vertices) connected by edges. In turn, these graph representations provide convenient language to foster interdisciplinary collaboration between researchers from diverse fields such as mathematics, computer science, physics, sociology, biology etc. and allow them to share ideas and transfer methodologies between different domains.

Graph representations of networks also offer a powerful mathematical framework for understanding the internal structure of these networks, studying their evolution, or even designing networks with required properties. Graph theory has developed special tools and concepts to analyze and uncover fundamental properties and patterns in networks such as connectivity, paths, cycles, centrality etc. This, in turn, allows researchers to design efficient algorithms for solving complex problems, optimize resource allocation or performance of the networked systems. In view of the vast contextual diversity of networks, there are numerous aspects that can be optimized. For instance, one of the most important questions is finding the shortest path in the network connecting two given nodes or detecting the edges that contribute most to preserving network connectivity.

The main aim of this thesis is to study a class of network optimization problems and to suggest a novel approach to their solution. In the context of computer networks, the task we address can be described as follows. Each node of the network can be a server or a client; we need to place several servers and correctly allocate the remaining clients between servers so that the server loads become balanced and bottlenecks of client-server routes minimized, which in turn results in a boost of the server's efficiency of data processing, its overall speed and general performance. To give a better understanding of the diversity and importance of the problems under consideration, we list several further illustrative examples.

1. **Location of the post offices.** Post companies often face non-uniform distribution of clients among their departments, leading to significant overload in some areas and underload in others. To efficiently handle the flow of clients within given department throughput constraints, it is essential to determine the appropriate number of post offices, assign responsible service areas within the region, and identify optimal locations within each designated area.

2. **Petrol station location.** The optimal location of petrol stations plays a significant role in business and road safety. Additional rest places, where the driver

could take a nap and have a bite, and check the safety of his vehicle, will certainly decrease the number of road accidents. However, randomly locating new petrol stations will only overburden some stations, while others will remain unused. Consequently, determining the optimal number of additional stations, allocating service regions along the highways, and identifying suitable locations within these regions become significant questions.

3. **Safeness of the ski resorts.** The number of traumatic cases significantly increased in large ski resorts recently. In order to prevent fatal consequences, providing first aid on time and transportation of the injured to the closest hospital is mandatory. Given the ski routes map and all possible passages between them, it becomes essential to identify optimal locations for emergency rescue posts, assign routes to each post, and determine the number of posts required to ensure adequate safety measures within the resort.

4. **Placement of the garbage cans.** With the constant growth of the number and size of megalopolises, efficient management of garbage disposal becomes a major challenge. Merely increasing the number of garbage cans is not a good approach, since it would result in delayed garbage collection, potential tank overflow, and ecological issues. Optimal garbage can placement requires determining the appropriate number of reservoirs that lead to efficient collection, assigning specific regions to each garbage location based on resident distribution, and identifying suitable locations within each region to minimize total distance from citizens' houses.

5. **Car rental companies.** When travelling abroad, individuals may encounter inconveniences with public transport or taxi services, such as limited routes, long waiting times, unavailability during certain hours, etc. Car rental services offer a flexible alternative to the traveller; however, a limited number of car return places presents a major disadvantage. Suppose a car rental company would like to expand their service. In that case, the natural questions are about the optimal number and strategic locations of new facilities and the estimated service area for clients.

6. **Logistic problems.** Consider a vast production network encompassing the manufacturing of raw materials and the construction of finalized production-ready items. Given the natural manufacturing regions (the location of which is determined by the minerals, water resources, etc.), the questions are, what the optimal number of factories is that would make the production stable and secure, where should these factories be located, and from which manufacturers the raw materials should be supplied?

Since network models may be of such a diverse structure, no algorithm can exist that would solve the optimization problems for all such models exactly and determine the best server locations and their areas of service. Also, real-world networks can be enormous in size, so the brute-force approach of running through all possible locations of the servers is extremely inefficient. A good alternative approach may be to use approximation algorithms. Even though they do not always give optimal results, their computational efficiency makes them worthwhile.

The aim of this thesis is to suggest new spectral approaches to the above network optimization problem. Our primary motivation stems from the recent paper by Steinerberger [10], in which eigenvectors for Laplacian matrices on graphs were used to find efficiently the approximate shortest paths to a given vertex. We generalized that idea to the case of several distinguished nodes called *servers* and constructed the *gravitational potential* related to the eigenvectors of the corresponding restricted Laplacian matrices. The network optimization algorithm we developed essentially uses the gravitational potentials on each of the following steps.

1. Construct the *gradient flow* of the network, which is an analogue of the gradient descent field for functions of many variables. The gradient flow helps to find a path from a given network node to the closest server, which is nearly the shortest one (Sec. 4.3).

2. The gradient flow for a fixed server placement determines the allocation of the nodes between these servers. This allocation is typically unbalanced and leads to sub-optimal server loads. We use the gravitational potential in the iterative greedy-type method that improves client-server allocation and, in several steps, leads to almost balanced clusters (Sec. 4.4).

3. Given the acceptable network clustering, we need to find the optimal placement of a server within each cluster that minimizes the total path length from all other cluster nodes. We suggest an iterative method that first initializes the server placement near the cluster centroid and then updates it using the corresponding gravitational potential and the gradient flow paths. To locate the centroid, we suggest an effective method to construct the dissimilarity matrix between all nodes and then use Multidimensional scaling (MDS) to embed the graph into $\mathbb{R}^d$ for $d = 2$ or $d = 3$ so that dissimilarities and obtained Euclidean distances become as close as possible (Sec. 5.2).

As we demonstrated in extensive experiments, the developed network optimization method performs well on diverse graph types.

The rest of the thesis is organized as follows. In the next chapter, we review approaches and algorithms on graphs that address similar optimization tasks, then formulate the main optimization problem of this work in more detail and introduce the measures to use in objective functions. In Chapter 3, we first introduce the basic notions of graph theory and fix notations, establish several properties of graph Laplacians used throughout the thesis, and then explain the main ideas and constructions of the paper [10]. In Chapter 4, the gravitational potential for a given set of server locations is constructed and its main properties are established. This gravitational potential defines the gradient flow on the network and thus performs clusterisation of the graph into the service areas for these servers. We next introduce the greedy-type algorithm that uses the gravitational potential to iteratively update server placement to achieve well-balanced distribution of these clusters. Chapter 5 addresses the problem of optimal server placement within the given cluster to minimize the total path lengths from all other nodes (clients). We explain the basic construction behind the MDS approach and then suggest the method that efficiently replaces zero values in the adjacency matrix by positive dissimilarity values that are close to the shortest path lengths. After identifying the initial server placement via the MDS embedding centroid, we describe the iterative updating method that is based on the gravitational potential and achieves a local optimum in a few iterations. The whole pipeline is summarized in Chapter 6, which also presents the experimental results of algorithm validation on graphs of diverse structures and properties. A summary of the whole thesis is given in the Conclusions, and, finally, the Appendix contains the mathematical derivation of the norming constants for the J_1 score.

Chapter 2

Network optimization problem

2.1 Overview of related approaches

Probably the most famous example of a network and the graph representing it dates back to Euler's solution of the problem known as the seven bridges of Königsberg. Since then, the graph theory has developed into a separate branch of mathematics and has proven very efficient in modelling real-world networks and analysing their functioning, see e.g. the books [2, 3].

Many natural global or local characteristics of networks and graphs describe and affect the global or local behaviour of processes in these network models. For instance, maximal s-t flow determines the maximal traffic between the source s and sink t in a network. *Betweenness* centrality [4] characterises the nodes that strongly influence information spread in a network.

One of the most important tasks of graph theory that is related to the aims of this thesis is that of clustering, i.e., division of the network into several parts that are highly homogeneous. Spectral clustering is of the most popular and efficient graph clustering methods; it is based on eigenvectors of the related graph Laplacian matrix and is also used in dimensionality reduction tasks [5, 6]. For sparse networks (i.e. those in which the vertex degrees grow slower than the total number N of vertices), leading eigenvectors can be calculated in $O(N^2)$ steps, which makes the method computationally attractive. Some other clustering methods are k-means [7] and DBSCAN [8] but there objectives are different to those we need in this study and the resulting clustering is often very inbalanced.

One of the most important tasks in network science is that of finding the shortest path between two nodes. A classic Dijkstra algorithm [9, Sec. 24.3] finds shortest paths from a given start node to all other nodes and results in the so-called shortest-path tree. Its complexity is $O(N^2)$ in the worst case of unstructured and non-sparse trees. However, for the task under study in this work, where K server-type nodes are fixed, and for every node the closest server should be located and the shortest path to it identified, one would need to run a Dijkstra algorithm for each server separately, which is inefficient. The method of the paper [10], in which spectral approach to the shortest path problem was suggested could be generalized to the case of multiple starting points (servers), and that is the approach we adopted in this research work.

The principal problem we address in this thesis requires splitting the network into several clusters of equal sizes and then determining optimal locations of server-type nodes within each cluster to minimize the total shortest path lengths. We were unable to find a single existing algorithm that would solve this task. Inspired by [10], we developed an approach that is based on the spectral properties of the graph and demonstrated its efficiency by extensive experiments.

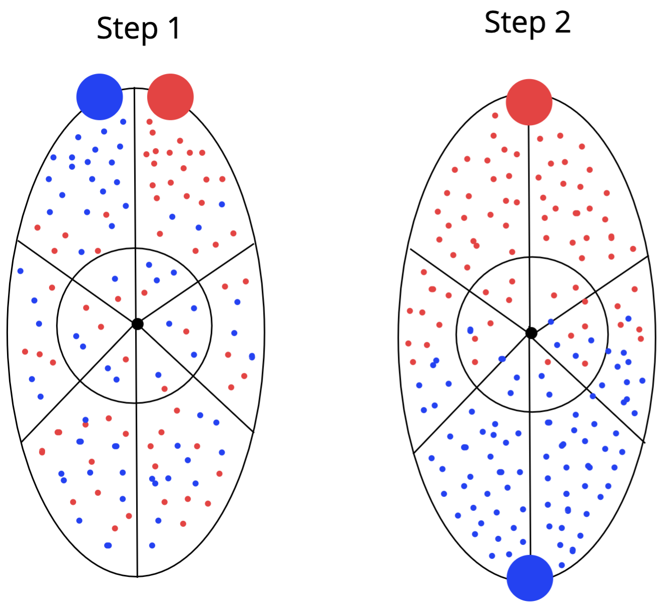


FIGURE 2.1: City model with two server nodes and client allocations

2.2 Problem statement

After thoroughly examining the apparently different at-first-sight cases, described in Section 1, one may observe some similarities. All the networks can be represented as a structure that is connected in some way. Among the elements of the structure, two main types can be distinguished: the *client* and *server* types. Furthermore, the concept of a *distance* between elements can be introduced, and for each server-type node, the notion of its *load* can be defined.

Assuming that all servers have similar throughput, the criteria following which a client will choose the server would be the distance between the client and server. Therefore, after determining the clients for a particular server, that server should be located in a place which minimizes the total distance to all its clients.

Let us introduce the idea in a more concrete example. Consider a model of a city on Fig. 2.1, where an ellipse represents borders, the city centre is represented by an inner circle, and lines represent the main roads. The task is to find the optimal locations of public services and determine the regions which should correspond to them.

The first image shows the importance of services' locations: if both of the service centres were located on one side of the semi-major axis, then people from all parts of the city should travel to the side, which would increase the road load and would make the services load unpredictable, as for people who travel from distant parts of the city there is no apparent reason to chose one service centre apart from another.

The second image demonstrates the first-step solution: determine locations of the service centres that will minimize the centre load. By determining the closest centre to each client, we can set the group of clients handled by each service centre.

The last image in Fig. 2.1 shows the optimization, which should be made at the last step: for the clients of each service centre, determine the location of the service centre, which will minimize the distance among the clients, i.e., the centre of a clients group. To summarize, the main problem considered in this thesis reads as follows.

Given a complex network, determine the number of servers and their placement that would optimize the network performance. This requires solving the following tasks:

(T1) Introduce the metrics to measure efficiency of network performance.

(T2) For each client-type element, determine the closest corresponding server.

(T3) Determine the optimal locations of server-type elements.

(T4) Establish the optimal number of servers, leading to the most efficient performance.

The first task is discussed in the next subsection. In Chapter 4, we develop the gravitational potential method inspired by [10] that allows to identify the server that is typically the closest one, along with the corresponding path, thus providing a solution to task (T2). Task (T3) on the optimal location of a server within a given cluster of nodes is discussed in Chapter 5, and problem of determining the optimal number of servers is experimentally studied in Chapter 6.

2.3 Optimization criteria and metrics

The measure of efficient distribution of client-type elements among server-type is the load of the servers. The best theoretical allocation of clients is the uniform one when each server receives an equal amount of clients (however, this distribution may not be achieved, depending on the actual structure). Thus, we can compare the load of each server with the optimal load and define the measure of goodness as the maximum of the normed difference.

Although this criterion seems realistic and intuitive, for some cases (one of which will be demonstrated in the next section), there are several locations with the same values of load, but different client distributions. In some cases, different servers have many clients in common, making client disposal inefficient. Therefore, an additional criterion which shows how many clients different servers have in common, i.e., server-clients group intersection, must be added.

After locating each server and determining specific clients for each, which will form so-called hubs around the servers, we can optimize the distance for each hub individually. As the clients are deterministically assigned for each server, moving the server between the hub's elements will not change the business or the server-clients group intersection criteria value.

The most convenient representation of a network is via *graphs* (see Section 3). A graph $G = (V, E)$ consists of the set E of *nodes* (vertices), some of which are connected by *edges* (elements of E) with prescribed positive *weights*. Each node can be a client or a server, edges give possible connections between the nodes, and the weights reflects how close the nodes are (e.g., in terms of the physical distance or the travel time).

Among the $N := |V|$ vertices of the graph, M are declared *clients* and K declared *servers*, $M + K = N$. The set of all clients is denoted by $V_0 = \{v_1, v_2, \dots, v_M\}$, and the set of all servers is denoted by $R = \{r_1, r_2, \dots, r_K\}$; then $V = V_0 \cup R$.

In the ideal situation, for every client, there is a single closest server; however, there could be cases when this is not the case and clients can not be assigned deterministically to one server. In this case, it is natural to introduce the probability p_{ij} that the i-th client v_i is handled by the j-th server r_j. This way, we define the *allocation* *probability matrix*

$$\boldsymbol{P} = \begin{pmatrix} p_{11} & \cdots & p_{1K} \\ \vdots & \ddots & \vdots \\ p_{M1} & \cdots & p_{MK} \end{pmatrix}.$$

We note that the sum in each row of the matrix $\boldsymbol{P}$ is equal to 1, $\sum\limits_{j=1}^{K} p_{ij} = 1$, and the sum in each column is equal to the load of the j-th server; in particular, $\sum\limits_{i=1}^{M} \sum\limits_{j=1}^{K} p_{ij} = M$.

The optimal server loads, with no regard for the graph architecture, will be achieved in the case of uniform client distribution among all servers, i.e., when all server loads are equal to M/K. Since distinct graphs may significantly differ in size, it is more natural to compare relative server loads, i.e., fractions of clients handled by each server. Therefore, we use the mean squared error (MSE) between the actual and optimal fractions of allocated clients as the measure of allocation efficacy:

$$\boldsymbol{J}_1 = \frac{K}{K-1} \sum_{i=1}^{K} \left(\frac{1}{M} \sum_{j=1}^{M} p_{ij} - \frac{1}{K} \right)^2 .$$

The norming factor $\frac{K}{K-1}$ was chosen for $\boldsymbol{J}_1$ to lie in the range $[0,1]$; the proof is given in Appendix A. The value of $\boldsymbol{J}_1$ shows how far from uniform is the network's current client-server distribution.

However, the load metric $\boldsymbol{J}_1$ itself does not always offer a good optimal criterion: in networks with symmetric patterns, many clients may be assigned to more than one server, see Fig. 2.1. Therefore, we must also minimize intersections of client clusters allocated to different servers, i.e., the number of common clients.

We say that the node i can be handled by server j iff $p_{ij} > 0$. Denote by $P_i = (p_{i1}, p_{i2}, \dots, p_{iK})$ the probability distribution for server allocation for node i. Then $\|P_i\|_1 := p_{i1} + p_{i2} + \dots + p_{iK} = 1$. Let also

$$\mathcal{P}_0 = \{(1,0,0,\dots,0), (0,1,0,\dots,0), \dots (0,0,0,\dots,1)\} = \{\pi_1, \pi_2, \dots, \pi_K\}$$

be the set of singleton distributions.

A simple visualization in the 3D case is shown on the figure 2.2. The closest probability distribution for node i is to the singleton one; in other words, we must determine which of the elements of $\mathcal{P}_0$ is the closest one to the corresponding row P_i of the matrix $\boldsymbol{P}$. Therefore, the problem of minimizing the clients allocation intersections (an analogue of the intersection-over-union, IOU) can be formulated as that of minimizing

$$\boldsymbol{J}_2 = \frac{1}{M} \sum_{i=1}^{M} \operatorname{dist}(P_i, \mathcal{P}_0),$$

where

$$\operatorname{dist}(P_i, \mathcal{P}_0) = \min_{j=1,\dots,K} \frac{K}{(K-1)} \|P_i - \pi_j\|_1,$$

and the factors are chosen to keep the values in the $[0,1]$ range.

Thus, we set the servers' optimality measure $\boldsymbol{J}$ to be the average of the load $\boldsymbol{J}_1$ (MSE-type) and intersection $\boldsymbol{J}_2$ (IOU-type) measures,

$$\boldsymbol{J} = \tfrac{1}{2}(\boldsymbol{J}_1 + \boldsymbol{J}_2),$$

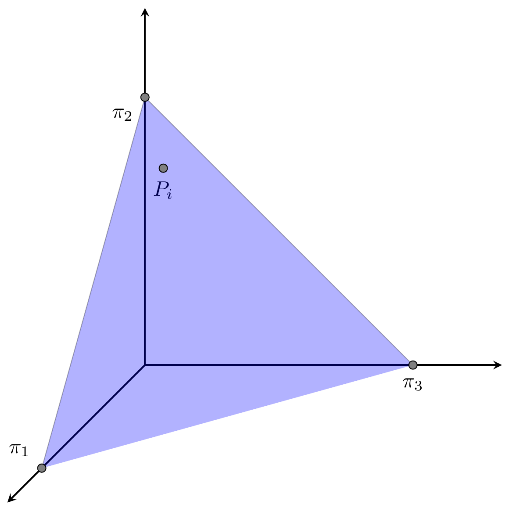


FIGURE 2.2: Optimization criteria

with values in the $[0, 1]$ range, and the value 0 corresponding to the ideal distribution: all clusters are of equal size and not intersecting.

After determining the optimal clients' distribution, i.e., determining K clusters allocated to particular servers, there is a possibility to optimize the location of the servers within their clusters. The optimal location should be somewhat close to the "centroid" of the cluster to minimize the total travel cost from all allocated clients. In other words, if we denote by $\{G_1, G_2, \dots, G_K\}$ the set of all server-clients groups, we need to find the locations of the servers $r_1, r_2, \dots, r_K$ in respective groups that minimize the sum of all path lengths from clients to the server in each group, i.e,

$$\mathcal{L}_i(r_i) := \sum_{v_j \in G_i} \mathrm{dist}(r_i, v_j),$$

where $\mathrm{dist}(r_i, v_j)$ is the length of the shortest path between the client v_j and the server r_i.

In summary, we need to find a vector $Y = (v_1, v_2, \dots, v_K)$, at which the function $\mathcal{L}(r_1, r_2, \dots, r_K) := \mathcal{L}_1(r_1) + \mathcal{L}_2(r_2) + \cdots + \mathcal{L}_K(r_K)$ assumes its minimum, i.e.,

$$Y = \arg\min_{r_i \in G_i} \mathcal{L}(r_1, r_2, \dots, r_K).$$

The problem of optimal server location in the given cluster is discussed in Section 5.

As a measure of the optimal clients' distribution accuracy, we compare the optimal sum of all path lengths inside every cluster to the one found by the algorithm using the mean absolute percentage error (MAPE) metrics:

$$MAPE = \sum_{i=1}^{K} \frac{\mathcal{L}_{i,act} - \mathcal{L}_{i,opt}}{\mathcal{L}_{i,act}}.$$

Chapter 3

Spectral Methods for Complex Networks

3.1 Basic notions of the graph theory

In this section, we introduce the basic notion related to graphs and fix notations to be used throughout the thesis.

A *directed graph* G is an ordered pair $G = (V, E)$, with $V = \{v_1, v_2, \dots, v_n\}$ being a set of *nodes* (*vertices*) ($|V| = n$), and $E = \{e_1, e_2, \dots, e_m\} \subseteq V \times V$ being a set of *edges* ($|E| = m$). An edge $e = (v_i, v_j)$ is said to connect node v_i to v_j, in which case v_i is connected (or adjacent) to v_j; this will be denoted $v_i \sim v_j$.

Graph G is *simple* if there are no loops or multiple edges connecting the same nodes. G is called *undirected* if $v_i \sim v_j$ implies $v_j \sim v_j$; edges are then identified with two-element subsets $\{v_i, v_j\}$ of V, and we write that $\{v_i, v_j\} \in E$. To simplify notations, we will often denote by e_{ij} the edge connecting v_i and v_j and refer to a vertex v_i or v_j as i or j, respectively.

In what follows, we will consider simple undirected graphs $G = (V, E)$.

A function $\omega : E \to \mathbb{R}_+$ is called the *weight*; we denote by ω_{ij} the weight of an edge $e_{ij} \in E$. The graph G with the weight function ω defined on its edges is called *weighted*. Observe that an unweighted graph is a particular case of a weighted one, with the constant weight $\omega \equiv 1$; therefore, all graphs will be assumed weighted.

The *adjacency* matrix A of the (weighted) graph $G = (V, E)$ is in an $n \times n$ symmetric matrix of the form

$$A_{ij} = \begin{cases} \omega_{ij}, & \text{if } \{v_i, v_j\} \in E, \\ 0, & \text{otherwise.} \end{cases}$$

The *degree* d_i of the node v_i is defined as the sum of the weights over all edges outgoing from it: $d_i := \sum\limits_{j\,:\,v_i \sim v_j} \omega_{ij}$. We denote by D the *degree matrix* of G, which is a diagonal $n \times n$ matrix defined by:

$$D_{ij} = \begin{cases} d_i, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

The weighted *Laplacian matrix* of the graph G is defined as $\boldsymbol{L} := D - A$; thus

$$\boldsymbol{L}_{ij} = \begin{cases} d_i, & \text{if } i = j, \\ -\omega_{ij}, & \text{if } v_i \sim v_j, \\ 0, & \text{otherwise.} \end{cases}$$

By definition, $\boldsymbol{L}$ is a symmetric matrix whose rows sum to zero; therefore, $\boldsymbol{L}$ is singular and $\lambda = 0$ is its eigenvalue. It turns out that the Laplacian matrix has many interesting and useful properties; in particular, it is positive and semi-definite, and the dimension of its null-space of (i.e., the multiplicity of the eigenvalue $\lambda = 0$) is equal to the number of connected components of the graph G. Proofs of these statements are given in the next section.

3.2 Spectral properties of graph Laplacian

In this section, we discuss the main spectral properties of the graph Laplacian to be used in subsequent sections and give arguments justifying them.

Proposition 1. *The Laplacian matrix* $\boldsymbol{L}$ *always has a zero eigenvalue.*

Proof. Let G be a graph of n vertices; we set $\mathbf{1} := (1, 1, \dots, 1)^\top \in \mathbb{R}^n$. Recalling that $d_i = \sum_{j:i\sim j} A_{ij} = \sum_j A_{ij}$, we conclude that

$$(\boldsymbol{L}\mathbf{1})_i = (D\mathbf{1})_i - (A\mathbf{1})_i = d_i - \sum_{j:i\sim j} A_{ij} = 0, \qquad i = 1, 2, \dots, n. \tag{3.1}$$

Therefore, $\boldsymbol{L}\mathbf{1} = \mathbf{0}$, i.e., $\mathbf{1}$ is an eigenvector of $\boldsymbol{L}$ with eigenvalue 0. □

Proposition 2. *The Laplacian matrix* $\boldsymbol{L}$ *is positive semidefinite.*

Proof. Take any $\phi \in \mathbb{R}^n$; then, keeping in mind that A is symmetric and $d_i = \sum_j A_{ij}$, we find that

$$\begin{aligned}\phi^\top \boldsymbol{L}\phi &= \sum_{i,j} L_{ij}\phi_i\phi_j = \sum_{i,j}(d_i\delta_{ij} - A_{ij})\phi_i\phi_j \\ &= \sum_i d_i\phi_i^2 - \sum_{i,j} A_{ij}\phi_i\phi_j = \sum_{i,j} A_{ij}\phi_i^2 - \sum_{i,j} A_{ij}\phi_i\phi_j \\ &= \frac{1}{2}\sum_{i,j} A_{ij}(\phi_i^2 - 2\phi_i\phi_j + \phi_j^2) = \frac{1}{2}\sum_{i,j} A_{ij}(\phi_i - \phi_j)^2 \geq 0,\end{aligned} \tag{3.2}$$

thus completing the proof. Here, δ_{ij} is the Kronecker delta. □

Proposition 3. *The multiplicity of eigenvalue zero of the Laplacian matrix* $\boldsymbol{L}$ *is* 1 *if and only if the graph* G *is connected.*

Proof. Suppose that G is connected and let ϕ be any eigenvector corresponding to the eigenvalue zero. Then, by (3.2),

$$0 = \phi^\top \boldsymbol{L}\phi = \frac{1}{2}\sum_{i,j} A_{ij}(\phi_i - \phi_j)^2$$

showing that $\phi_i = \phi_j$ for every pair of connected nodes $i \sim j$. By connectedness of G, we get $\phi \equiv \text{const}$, so that the nullspace of $\boldsymbol{L}$ is of dimension 1.

Assume now that the graph G is disconnected and $G_1 \neq G$ is a maximal connected component of G. Consider the vector $\mathbf{1}_{V_1}$, the indicator function of the set $V_1 \subset V$ of vertices in G_1, and let $i \in V_1$. Since $i \sim j$ is only possible when $j \in V_1$, we find that, by analogy with (3.1),

$$(\boldsymbol{L}\mathbf{1}_{V_1})_i = d_i - \sum_{j\in V_1 : i\sim j} A_{ij} = d_i - \sum_{j : i\sim j} A_{ij} = 0.$$

Similar arguments show that $(\boldsymbol{L}\mathbf{1}_{V_1})_i = 0$ for every $i \in V \setminus V_1$; as a result, $\mathbf{1}_{V_1}$ is an eigenvector of $\boldsymbol{L}$ with eigenvalue zero. Therefore, the nullspace of $\boldsymbol{L}$ contains two linearly independent vectors $\mathbf{1}$ and $\mathbf{1}_{V_1}$ and thus is of dimension at least two. □

Proposition 4. *The number of connected components of the graph G is equal to the multiplicity of the eigenvalue zero of the Laplacian $\boldsymbol{L}$.*

Proof. Denote by $k > 1$ the number of connected components $G_1, G_2, \dots, G_k$ of the graph G and by $V_1, V_2, \dots, V_k$ the corresponding subsets of nodes; let also n_p be the cardinality of V_p. Since $v_i \sim v_j$ is only possible if the nodes v_i and v_j are from the same subset V_p, by rearranging the nodes $v_1, \dots, v_n$ in proper order, we make the Laplacian $\boldsymbol{L}$ block diagonal with blocks $L_1, L_2, \dots, L_k$. Here, L_p is the Laplacian matrix of the connected component G_p; therefore, L_p has eigenvalue zero of multiplicity one due to Proposition 3. Recall that for a symmetric matrix S, the multiplicity of an eigenvalue λ_0 equals the multiplicity of $\lambda = \lambda_0$ as a zero of the characteristic polynomial $\det(\lambda I - S)$. Since

$$\det(\lambda I_n - \boldsymbol{L}) = \det(\lambda I_{n_1} - L_1) \cdots \det(\lambda I_{n_k} - L_k),$$

we conclude that the multiplicity of the eigenvalue zero of the graph Laplacian $\boldsymbol{L}$ is equal to k. □

3.3 The shortest path problem

The main idea of our research is based on the paper of *Steinerberger* [10], where a new spectral approach to the task of constructing the shortest path between vertices of an undirected unweighted connected graph was proposed. We explain here the main idea of the algorithm and briefly discuss the obtained results.

Consider a graph $G = (V, E)$ and two vertices i and j, the shortest path between which should be found. As the graph is undirected, the shortest path from i to j travelled in the opposite direction gives the shortest path from j to i; we will regard i as a destination node.

The algorithm suggested in [10] proceeds as follows. We construct the graph Laplacian matrix $\boldsymbol{L}$ and then delete the i-th row and column from it, resulting in a matrix denoted $\boldsymbol{L}_i$. Find an eigenvector ϕ associated with the smallest eigenvalue of $\boldsymbol{L}_i$ and consider this eigenvector as a function $f : (V \setminus \{i\}) \to \mathbb{R}$ defined on all graph vertices except for the destination vertex i. Setting $f(i) = 0$, we extend f to a function $V \to \mathbb{R}$. This function has several properties: it is sign-constant (and thus can be taken positive), vanishes only at the destination vertex i, and for every node k there exists an adjacent node l such that $f(l) < f(k)$.

The path between any node j and the fixed destination node i can be constructed as follows. Starting from j, select a node adjacent to it where the function f assumes the smallest value. Continue from that node until we reach the node i. It follows from the properties of f that the algorithm cannot get stuck and will always produce a path between the given vertices j and i.

As explained in [10], the constructed path from j to i may not always be the shortest. However, there are many different types of graphs on which all pairwise shortest paths are constructed by that spectral method: the Folkman Graph, the Gray graph, the Harries-Wong Graph, the Ljubljana Graph, the Petersen Graph, the Line Petersen Graph, Wells Graph etc [10].

Gardner's Map is an example of a graph where the algorithm does not produce the shortest path between every two vertices (see Fig. 3.1). The shortest path (shown

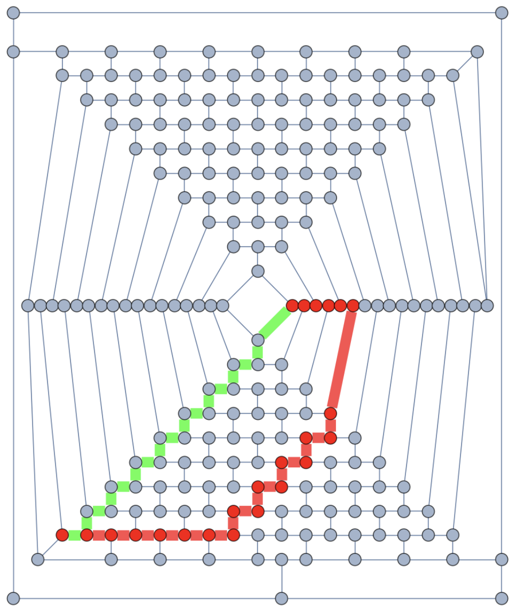


FIGURE 3.1: Gardner's Map graph: shortest path is green, spectral method path is red

in green) has length 17, and the spectral method leads to the path (shown in red) of length 22. Nevertheless, the mean value of the path's difference between two randomly chosen vertices is only 0.08 steps longer than the shortest one. For a random graph of 300 points sampled uniformly at random in the rectangular domain $[0,3] \times [0,1]$, the shortest spectral path is, on average, only 0.05 steps longer.

We also want to emphasize that constructing the path on a graph according to this spectral method by selecting the next vertex with the smallest value of the function f can be considered as a discrete analogue of the gradient descent method, the main idea of which is to search for a local minimum of an objective function f in the direction opposite to the gradient of f at the current point, i.e., in the direction of the steepest descent. Approximating the function's derivative by the scaled difference of its values on the vertices, we select the next node with the biggest difference. Further on, we will refer to this spectral path construction as to the construction using gradient flows.

Inspired by the above spectral approach for the shortest path problem, we developed a spectral method for network optimization, which seeks the optimal placement of several nodes (regarded as servers) that minimizes total travel costs for the remaining network nodes (regarded as clients). It is based on the notion of *gravitational potential*, which we introduce in the next chapter.

Chapter 4

Gravitational Potential

In this chapter, we introduce the notion of gravitational potential, on which our network optimisation method is based. In particular, the gravitational potential constructed for the given number K of server-type nodes determines the partitioning of the network into K clusters. It will be used in an iterative procedure to optimize initial partitioning and then to determine the optimal placement of the servers to minimize the total paths length in the next chapters.

4.1 Constructing the gravitational potential on a graph

Let $G = (V, E)$ be a connected graph. Denote by $\boldsymbol{L}$ the corresponding weighted graph Laplacian. In this section, we explain the construction of the gravitational potential for a set of servers placed at vertices of the subset $R \subset V$, $|R| = K$.

For every server $r_i \in R$, we remove the corresponding row and column of the matrix $\boldsymbol{L}$. The reduced matrix $\boldsymbol{L}_R$ contains only entries of $\boldsymbol{L}$ for the client vertices V_0. To explain the main idea of the gravitational potential construction, we assume first that the graph G remains connected after removing the server nodes R. In that case, we denote by $\phi \in \mathbb{R}^M$ the eigenvector associated with the smallest eigenvalue μ_1 of $\boldsymbol{L}_R$. We next extend ϕ to R by setting $\phi(r_i) = 0$ for each server $r_i \in R$; so defined ϕ can be interpreted as a function $\phi : V \to \mathbb{R}$ on the whole vertex set V. If the case when removing the server vertices makes the graph disconnected, we apply a similar procedure to each connected component separately, see Section 4.2.2.

The function ϕ has several useful properties:

(P1) ϕ can be chosen sign-constant (thus, we can assume it to be non-negative).

(P2) ϕ vanishes only on vertices of the server set R.

(P3) For any vertex $v \notin R$ there always exists a neighbouring vertex ω, $\{v, \omega\} \in E$, such that $\phi(\omega) < \phi(v)$.

We will prove that such a function ϕ is unique under condition (P1). Consequently, it is natural to think of this function ϕ as a potential of a gravitational field with field sources at the server vertices, which exerts a force on client vertices. Therefore, considering clients as elements with a mass, we expect each client to choose the closest server, as the force exerted on it by the closest server will be the greatest.

A visual interpretation is given in Fig. 4.1, where balls represent servers, and the grid height represents the value of the gravitational potential at client vertices.

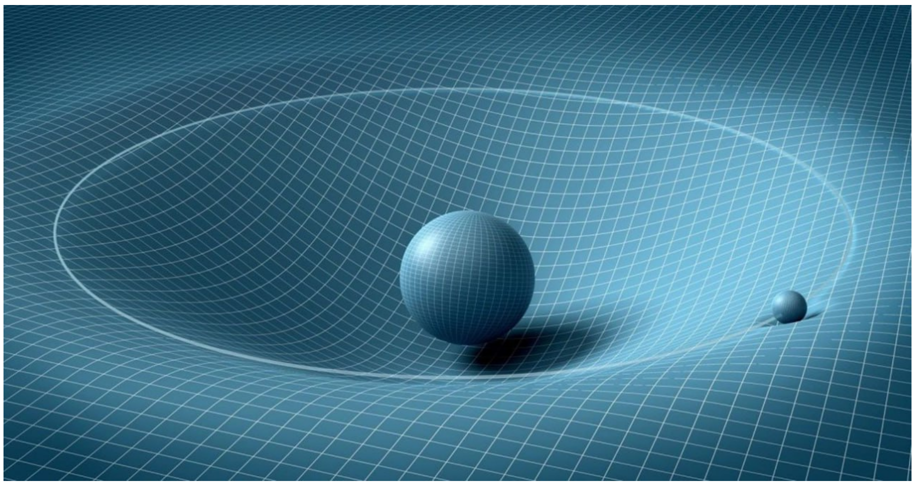


FIGURE 4.1: Gravitational potential model

4.2 Proof of Properties (P1)–(P3)

The reduced Laplacian matrix $\boldsymbol{L}_R$ is symmetric and positive semi-definite. Symmetry of the matrix $\boldsymbol{L}_R$ is clear from the construction and definition. We next prove its positive semi-definiteness using variational principle [11]. Denote by $\mathbf{x} \in \mathbb{R}^{M+N}$ and $\mathbf{y} \in \mathbb{R}^{M}$ any non-zero vectors, which we interpret as functions defined on the vertices in V and V_0, respectively. Then

$$\lambda_1 := \min_{\mathbf{x}\neq\mathbf{0}} \frac{\langle \mathbf{x}, \boldsymbol{L}\mathbf{x}\rangle}{\|\mathbf{x}\|^2} \leq \min_{\mathbf{x}\neq\mathbf{0}\,:\,\mathbf{x}(R)=0} \frac{\langle \mathbf{x}, \boldsymbol{L}\mathbf{x}\rangle}{\|\mathbf{x}\|^2} = \min_{\mathbf{y}\neq\mathbf{0}} \frac{\langle \mathbf{y}, \boldsymbol{L}_R\mathbf{y}\rangle}{\|\mathbf{y}\|^2} =: \mu_1.$$

By the minmax property for eigenvalues, λ_1 is the smallest eigenvalue of $\boldsymbol{L}$ and μ_1 is the smallest eigenvalue of $\boldsymbol{L}_R$; as the full Laplacian $\boldsymbol{L}$ is positive semi-definite, we get $\mu_1 \geq \lambda_1 = 0$.

We next show that $\mu_1 > 0$; thus $\boldsymbol{L}_R$ is a positive definite matrix. Assume, on the contrary, that $\mu_1 = 0$ and let $\mathbf{y}$ be a vector of norm one, $\|\mathbf{y}\| = 1$, on which the minimum of the quadratic form $\langle \mathbf{y}, \boldsymbol{L}_R\mathbf{y}\rangle$ is achieved. Extend $\mathbf{y}$ to a vector $\mathbf{x}$ on the whole vertex set V by setting $\mathbf{x}(i) = 0$ for each server $i \in R$; then, by (3.2),

$$0 = \langle \mathbf{y}, \boldsymbol{L}_R\mathbf{y}\rangle = \langle \mathbf{x}, \boldsymbol{L}\mathbf{x}\rangle = \tfrac{1}{2}\sum_{i\sim j} \omega_{ij}\big(\mathbf{x}(i) - \mathbf{x}(j)\big)^2.$$

This implies that $\mathbf{x}$ is constant on every connected component of G. Since G is assumed connected, $\mathbf{x}$ is constant on the whole vertex set V, and because $\mathbf{x}$ vanishes at server vertices in R by construction, we conclude that $\mathbf{x} \equiv 0$. This contradicts the assumption that $\mathbf{y}$ is a vector of norm one and thus completes the proof that $\mu_1 > 0$.

We next construct the gravitational potential corresponding to the server set $R \subset V$. We recall that the quadratic form for the Laplacian matrix $\boldsymbol{L}$ is

$$\langle \mathbf{x}, \boldsymbol{L}\mathbf{x}\rangle = \frac{1}{2}\sum_{i\sim j} \omega_{ij}\big(\mathbf{x}(i) - \mathbf{x}(j)\big)^2.$$

Consider the variational problem of minimizing the above quadratic form over all functions f on V such that $f(R)=0$ and $\|f\|^2=1$:

$$\begin{aligned} \phi &= \operatorname*{arg\,min}_{f\,:\,f(R)=0} \sum_{v_i\sim v_j} \omega_{ij}\big(f(v_i)-f(v_j)\big)^2, \\ &\text{subject to } \sum_{v\in V} |f(v)|^2 = 1. \end{aligned} \tag{4.1}$$

Here and hereafter, $f(R)=0$ is used as a shorthand notation for the property that f vanishes on R, i.e., that $f(r)=0$ for every $r\in R$. The existence of the minimizer follows from the compactness of the unit ball $\|\mathbf{x}\|=1$ in $\mathbb{R}^n$ and the continuity of the function to be minimized.

Equivalently to (4.1), we can restrict the function f to the client vertices $v\in V_0$ only forming the function f_0 and then minimize the quadratic form $\langle f_0, \boldsymbol{L}_R f_0\rangle$ subject to $\|f_0\|=1$. Recall that the matrix $\boldsymbol{L}_R$ is obtained from $\boldsymbol{L}$ by removing the i^{th} row and column for each $v_i\in R$; in terms of the graph G, this is equivalent to removing from G every vertex $v_i\in R$ and all edges outgoing from it. After such a removal, the graph either remains connected or becomes disconnected; the number of disconnected clusters can be found using Proposition 4. Let us first investigate the former case.

4.2.1 Connected case

In this subsection, we construct the gravitational potential in the case when the graph G remains connected after removing the server nodes of R and establish properties (P1)–(P3).

For a function f on the vertex set V, we denote by f_0 its restriction onto the clients' vertices in V_0. Assume also that $f(R)=0$ (i.e., that f vanishes on R); then

$$\langle f_0, \boldsymbol{L}_R f_0\rangle = \langle f, \boldsymbol{L} f\rangle = \tfrac{1}{2}\sum_{v_i\sim v_j} \omega_{ij}\big(f(v_i)-f(v_j)\big)^2.$$

Since $\boldsymbol{L}_R$ is a symmetric matrix, by the variational principle the function f_0 minimizing $\langle f_0, \boldsymbol{L}_R f_0\rangle$ over f_0 satisfying $\|f_0\|=1$ is an eigenvector corresponding to the smallest eigenvalue $\mu_1>0$ of $\boldsymbol{L}_R$. We take such a minimizing function f_0 and extend it by zero on the server nodes $v_i\in R$ to form the function f; then f is the solution to the optimization problem (4.1). Set now $g(\cdot)=|f(\cdot)|$; since the value of g on every element of the set R equals 0, the ℓ^2-norm of g remains equal to 1.

Noting that for all real numbers a and b it holds that $|a-b|\ge ||a|-|b||$, we conclude that

$$\begin{aligned} \sum_{v_i\sim v_j} \omega_{ij}\big(g(v_i)-g(v_j)\big)^2 &= \sum_{v_i\sim v_j} \omega_{ij}\big(|f(v_i)|-|f(v_j)|\big)^2 \\ &\le \sum_{v_i\sim v_j} \omega_{ij}\big(f(v_i)-f(v_j)\big)^2. \end{aligned}$$

Since f was a function minimizing the above expression, the equality in the last inequality must hold, whence

$$|f(v_i)-f(v_j)| = ||f(v_i)|-|f(v_j)||$$

for all pairs of connected vertices $v_i\sim v_j$. Therefore, on every pair of connected vertices $v_i\sim v_j$, the function f must have the same sign (or be zero). The only possibility for f to be of different signs at some non-adjacent vertices v_i and v_j is that every path $v_i \sim u_1 \sim \cdots \sim u_k \sim v_j$ from v_i to v_j must include a vertex u_l on which f vanishes. Since G remains connected after removing all server nodes $r_i \in R$, the above path can always be chosen to avoid the servers, whence such a u_l can be assumed not in R.

Assume now that a function f solving (4.1) changes its sign on V. As explained above, f must vanish at some vertex u_l not in R. By connectedness of G, we can assume that u_l has at least one neighbouring vertex on which f assumes a non-zero value; otherwise, we can simply change u_l to the last vertex on some path from u_l to some v with $f(v) \neq 0$ on which f vanishes. We now prove that existence of such a u_l contradicts the assumption that g also solves the problem (4.1).

To this end, we observe that for any set of numbers $x, x_1, x_2, \dots, x_k \in \mathbb{R}$ and any positive numbers $\omega_1, \omega_2, \dots, \omega_k$, we get

$$\sum_{j=1}^{k} \omega_j (x_j - x)^2 \leq \sum_{j=1}^{n} \omega_j (x_j - \bar{x})^2,$$

with the equality achieved if and only if x is equal to the weighted mean $\bar{x} := \sum_j \omega_j x_j / \sum_j \omega_j$ of x_j. Therefore, by replacing the value of the function g at the node u_l by the (positive) weighted mean of its neighbours, we decrease the value of the quadratic form in (4.1) and increase the ℓ^2 norm of g. Taking the modified function g and scaling it to ℓ_2-norm one, we get a function $\tilde{g}$ delivering a smaller value of the functional in (4.1) than g, thus contradicting the fact that the function g is a minimizer.

To summarize, we proved that the function f solving the optimization problem (4.1) does not vanish outside R and thus has a constant sign on the client vertices V_0. We can multiply f by -1 if necessary to make it non-negative on V, thus completing the proof of (P1) and (P2).

Next we prove uniqueness of the solution to (4.1). Let f and h be two linearly independent minimizers; by construction, they vanish on R. Since the restrictions f_0 and h_0 of f and h onto the clients vertices V_0 are eigenvectors of the symmetric matrix $\boldsymbol{L}_R$ corresponding to its smallest eigenvalue μ_1, any their linear combination is also an eigenvector of $\boldsymbol{L}_R$. Thus f and h can be chosen orthogonal; as they both are sign-constant on V_0, their inner product can not be zero. This contradiction shows that a minimizer f of (4.1) that is non-negative on V is unique. We denote this minimizer by ϕ.

It remains to prove the third property (P3). Take $v \notin R$; then there are two cases: either v is connected to some server or not. In the first case, $v \sim r$ for some $r \in R$, and we have $\phi(v) > 0 = \phi(r)$ as claimed. In the second case, we use the fact that the restriction ϕ_0 of ϕ to the client vertices V_0 is an eigenvector of $\boldsymbol{L}_R$ for the eigenvalue $\mu_1 > 0$, so that

$$(\boldsymbol{L}_R \phi_0)_v = \mu_1 \phi_0(v) > 0. \tag{4.2}$$

Assume now that $\phi(v_i) \geq \phi(v)$ for every v_i adjacent to v; observe that all v_i are client nodes and thus present in the matrix $\boldsymbol{L}_R$. Therefore,

$$(\boldsymbol{L}_R \phi_0)_v = d_v \phi(v) - \sum_{v_i \sim v} w_{v,v_i} \phi(v_i) \leq \phi(v) \big[d_v - \sum_{v_i \sim v} w_{v,v_i} \big] = 0,$$

which contradicts inequality (4.2). The derived contradiction shows that there must be at least one vertex ψ adjacent to v such that $\phi(\psi) < \phi(v)$, thus finishing the proof

Algorithm 1: Construction of the gravitational potential ϕ and detection of the components G_i in a general case

```
Data: The truncated Laplacian matrix $\boldsymbol{L}_R$
Result: Components $G_j$, gravitational potential $\phi$
1 initialize $i := 1$ and $\boldsymbol{L}_1 := \boldsymbol{L}_R$
2 while $\boldsymbol{L}_i$ is non-empty do
3     calculate the smallest eigenvalue of $\boldsymbol{L}_i$ and the corresponding normalized
        eigenfunction $f_i$;
4     set $V_i$ to be the support of $f_i$ and $G_i$ the corresponding subgraph of $G$;
5     set $\phi(v) = |f_i(v)|$ for every $v \in V_i$;
6     remove from $\boldsymbol{L}_i$ rows and columns corresponding to the vectices in $V_i$
        and call the resulting matrix $\boldsymbol{L}_{i+1}$;
7     set $i := i + 1$
```

of (P3).

4.2.2 Disconnected case

Consider now the case when the graph becomes disconnected after removing the server vertices r_i and all edges from them. The resulting graph is therefore split into m connected components $G_1, G_2, \dots, G_m$, $m > 1$, with $n_1, n_2, \dots, n_m$ vertices respectively. We can rearrange the graph vertices so that n_1 vertices from G_1 are listed first, then n_2 vertices from G_2 and so on. After such a rearrangement, the matrix $\boldsymbol{L}_R$ becomes block-diagonal with submatrices $\boldsymbol{L}_1, \boldsymbol{L}_2, \dots, \boldsymbol{L}_m$ of sizes $n_1, n_2, \dots, n_m$ respectively. We construct the gravitational potential on each component G_j separately. To this end, we take the eigenfunction ϕ_j for the matrix $\boldsymbol{L}_j$ corresponding to the smallest positive eigenvalue; by above, it can be chosen positive on the vertices of G_j. Denote now by ϕ the function on V defined in the following way: $\phi(v) = 0$ for every server $v \in R$ and $\phi(v) = \phi_j(v)$ for every vertex v belonging to the component G_j. The constructed function satisfies properties (P1) – (P3); in particular, it is positive on V_0 and zero on R.

In the considered case of a disconnected graph G_R, a natural question may arise about detecting the blocks of the matrix $\boldsymbol{L}_R$. The above analysis suggests Algorithm 1, which on the way also constructs the gravitational potential ϕ. This algorithm succeeds because, on each step i, the smallest eigenvalue μ_i is an eigenvalue of some of the remaining blocks in $\boldsymbol{L}_R$. As proved in the previous subsection, the corresponding eigenfunction f_i does not vanish on the component G_i of the graph and can be taken as gravitational potential (after flipping the signs in f_i if required). The function f_i must be zero on the remaining blocks, thus completely determining G_i. Observe that even if μ_i is a multiple eigenvalue, this can only hold if two distinct components G_i and G_j of the graph G generate the same smallest eigenvalue μ_i; the eigenfunction f will keep constant sign on G_i and G_j or can vanish on one of them. In the latter case, the algorithm detects one of the components G_i or G_j and the gravitational potential on it; in the former case, we add $G_i \cup G_j$ and determine the gravitational potential ϕ on the union $G_i \cup G_j$. Since the ultimate goal on this step is to construct ϕ, this case creates no complications.

4.3 Gradient flows and closest servers

In the previous section, we explained how to construct the gravitational potential ϕ for a graph with fixed server locations. As in [10], we can use this function to determine the gradient flow from each node to one of the servers. Namely, take an arbitrary client node v, consider all nodes connected to it, choose $\omega \sim v$ with the smallest value of $\phi(\omega)$, and point the gradient flow from v to ω. In the rare case of several vertices with the smallest value of ϕ, we include in the gradient flow directions from v to all of them.

According to Property (P3), at least one node connected to v has the value of ϕ smaller than $\phi(v)$; thus we guarantee that $\phi(\omega) < \phi(v)$. Therefore, the gradient flow from the node v corresponds to the steepest descent in the continuous case. In several steps, this gradient flow reaches one of the servers. As experimentally established in [10], in the case of a single server, this gradient flow chooses almost the shortest path to it. This fact is also experimentally confirmed in Chapter 6 for the case of several servers. Thus almost all gradient flows point to the closest server from the given client node and produce almost the shortest paths to these servers.

The paths from each node to one of the servers constructed using gradient flow generate clustering of the graph into K components consisting of the nodes corresponding to respective servers. As this clustering may be unbalanced, we optimize it in the next section with a greedy-type algorithm we developed based on gravitational potentials.

4.4 Clustering optimization

We begin with the motivational example of a well-known problem that can be easily solved using the greedy algorithm.

Suppose that hosting a radio show efficiently requires reaching listeners in k country regions. A decision must be made on what stations to play it on to reach all or most of those listeners. As it costs money to broadcast the show on each station, the goal is to minimize the number of hosting stations, given a list of stations and regions each station covers.

A greedy algorithm paradigm can be applied to this problem as follows:

1. Pick the station covering the largest number of regions that have not been covered yet.
2. Add it to the station lists and then continue with step 1 if not all regions have been covered.

A straightforward analogy to servers' efficiency optimization can be drawn as follows. Assume we have K servers, which must be located at some of N nodes. On each step, we pick the node with the largest value of the gravitational potential among all nodes; if the graph is disconnected, we pick the node with the largest value of the gravitational potential on the biggest connected component. According to the property (P3) of the gravitational potential (4.1), vertices closer to the current server have smaller values of the gravitational potential than the more distant ones. Therefore, applying the greedy paradigm, we pay attention to a region with high values of gravitational potential since it consists of nodes which are far from the current server locations; we then move one of the servers in that region so that the new server location takes over the largest portion of the clients that are currently too far from the existing servers, thus levelling out the cluster sizes.

The exact solution to this problem for fixed k is NP-hard, and applying a greedy approach reduces complexity to a polynomial one.

In other words, we propose the following iterative algorithm of servers' clustering optimization using the gravitational potential method.

1. Firstly, we randomly initialize the server's location and construct the corresponding gravitational potential on the graph.

2. The gradient flow method of Section 4.3 then determines the (nearly) closest servers for every client, thus performing initial clusterisation. We then form the probability allocation matrix $\boldsymbol{P}$ and calculate the clustering scores $\boldsymbol{J}_1$, $\boldsymbol{J}_2$, and their average $\boldsymbol{J}$.

3. On the iterative step of the greedy algorithm, we take the server with the smallest cluster size and move it to the node with the largest value of the gravitational potential.

4. Recalculate the gravitational potential for new server locations and repeat steps 2.–4. until the optimization score $\boldsymbol{J}$ no longer decreases.

5. Return the location of the servers with the lowest score and the corresponding clustering of the clients.

After completing this greedy clustering algorithm, we resolve server assignment conflicts for the clients supported by several servers. The probability matrix $\boldsymbol{P}$ immediately detects such clients since the corresponding rows include values different from $\{0, 1\}$. We assign such clients deterministically to a server (or one of the servers) with maximal allocation probability. After that, every client is assigned to exactly one server.

Chapter 5

Server location optimization

5.1 General idea

Finding a "central" vertex v in a graph that minimizes the total paths' length to v from all other vertices is a problem not possessing an efficient algorithm that returns the exact solution. The naive brute force approach suggests to iterate over all nodes, for each node v calculate the shortest paths from v to all other nodes using Dijkstra or any other shortest path algorithm, and finally to select the node v with minimal total paths' length. However, this approach is computationally very inefficient and requires $O(n^3)$ operations. A natural improvement comes from the intuitive observation that "outer" nodes, which are on the "boundary" of the graph, can not be optimal and thus can be discarded from the iteration process. Therefore, a natural question arises, if there is a method to effectively determine such an "inner" part of a graph containing the optimal "central" node.

However, many real-world networks (and the graphs representing them) have very complicated internal structure and thus no natural or easily determined boundary. Although in the Euclidean space the point minimizing the sum of squared distances to a fixed set of points $\mathbf{x}_1, \dots, \mathbf{x}_N$ is their mean $\bar{\mathbf{x}} = (\mathbf{x}_1 + \dots, \mathbf{x}_N)/N$, even if the network nodes are physically located in $\mathbb{R}^3$ or some higher-dimensional Euclidean space, the weights of the edges often do not represent the actual distance in that space and thus these locations cannot be used to determine the "central" vertex. Also, the actual objective function is the total distance from a given point $\mathbf{x}$ to $\mathbf{x}_1, \dots, \mathbf{x}_N$,

$$f(\mathbf{x}) = \sum_{j=1}^{N} \operatorname{dist}(\mathbf{x}, \mathbf{x}_j);$$

the point minimizing f is known as the Fermat–Torricelli point, and there is no closed-form formula to construct it from $\mathbf{x}_1, \dots, \mathbf{x}_N$ explicitly, see [14]. In the context of graphs, the additional obstacle appears that the edge weights may violate the triangle inequality and thus be incompatible with any distance.

In this section, we suggest an approach giving a good initial approximation to the central vertex; after several iterations, we then get a local minimum of the total paths' length that is a good candidate for the "central" point.

Namely, we use the Multidimensional Scaling (*MDS*) technique to find an embedding Φ of the vertex set V in $\mathbb{R}^d$ some $d \geq 2$ such that, for each pair of adjacent vertices v_i and v_j, the distance between $\Phi(v_i)$ and $\Phi(v_j)$ is as close as possible to w_{ij}. We observe that, usually, $d = 2$ or $d = 3$ is insufficient for complex graphs, and that the MDS results in higher dimensions can not be visually interpreted; however, as shown in [15], the algorithm's outcomes are very satisfactory even for relatively small d. Moreover, we suggest a simple way of converting G into a complete weighted graph $\tilde{G}$ so that, for every pair of i and j, w_{ij} is close to the shortest path length between v_i and v_j and, in turn, to the distance between $\Phi(v_i)$ and $\Phi(v_j)$.

After a satisfactory embedding of a graph G has been found, we calculate the mean $\bar{\mathbf{x}}$ of the points $\mathbf{x}_j = \Phi(v_j)$, determine the point $\mathbf{x}_j$ that is the closest one to $\bar{\mathbf{x}}$, and take the respective vertex v_j as the initial approximation for the central point of the graph G. We then test the nodes adjacent to v_j for better candidates and stop whenever no further improvements of the objective function is possible. As the experiments show, the process typically stops after two or three iterations.

In the next two subsections, we give details of the initialization of the central point and the iterative improvement method.

5.2 Formulation of the approach

The node that minimises the shortest paths' sum to all other nodes in a graph should be close to the graph's centre in a geometrical sense as offered by the multidimensional scaling. The multidimensional scaling (MDS) refers to a family of dimensionality reduction techniques used to represent high-dimensional data in low-dimensional space while approximately preserving the original distances [16]. The classical multidimensional scaling approach is a useful alternative for graph drawing [16].

Let $\Delta \in \mathbb{R}^{n\times n}$ denote a symmetric matrix of metric dissimilarities or distances δ_{ij} between some abstract elements $i, j \in \{1, \dots, n\}$. The goal of multidimensional scaling is to find positions $\mathbf{x}_i \in \mathbb{R}^d$ in the d-dimensional space with $d \ll n$ such that $\|\mathbf{x}_i - \mathbf{x}_j\| \approx \delta_{ij}$, i.e., such that the distances are well preserved in this low-dimensional space.

The graph's adjacency matrix can not be taken as a matrix of dissimilarities, because it has non-zero values only between vertices connected by an edge and is often quite sparse. In graph drawing, the matrix Δ usually consists of shortest-path distances [15], computing which is costly for large graphs. We propose a simpler approximation of the graph's shortest-distance matrix, which we describe next.

5.2.1 Construction of the dissimilarity matrix Δ

Given a weighted graph $G = (V, E)$, denote by $A^\dagger$ an unweighted adjacency matrix, i.e., a matrix with values 1 in entries where the original adjacency matrix A is non-zero, and values 0 otherwise. We note the well-known fact from the graph theory that the element $((A^\dagger)^k)_{ij}$ of the unweighted adjacency matrix $A^\dagger$ raised to power k gives the number of walks between the nodes i and j of length k. Consider now the matrix $C := AA^\dagger + A^\dagger A$; the matrix multiplication shows that

$$c_{ij} = \sum_{l\,:\,i\sim l\sim j} (a_{il}a^\dagger_{lj} + a^\dagger_{il}a_{lj}) = \sum_{l\,:\,i\sim l\sim j} (a_{il} + a_{lj})$$

is the total length of all two-step paths between i and j.

Thus we propose the following iterative method of constructing an approximate dissimilarity matrix Δ for the graph G:

1. Construct the matrix $C = AA^\dagger + A^\dagger A$
2. Divide non-zero elements of the matrix C by corresponding elements of the matrix $(A^\dagger)^2$ and denote the resulting matrix by $\tilde{A}$. The non-zero entries $\tilde{a}_{ij}$ of $\tilde{A}$ are the average lengths of two-step paths between i and j

3. Fill the non-diagonal zero entries of the matrix A with the corresponding elements of the matrix $\tilde{A}$

4. If the re-filled matrix A remains too sparse, we repeat the steps (1)–(3) to the re-filled matrix A, inserting the average lengths of walks between the nodes of the graph G up to four steps long on places with zero entries

5. For each non-diagonal entry in the resulting matrix that is still zero, replace it with the maximum entry of the corresponding row and column of the current matrix A, thus estimating the distance between i and j by the so-far longest path from i or j

6. The resulting matrix is denoted Δ and used as a dissimilarity matrix for the MDS algorithm.

5.2.2 The MDS embedding

After constructing the dissimilarity matrix Δ, our task remains to find matrix $\mathbb{X} \in \mathbb{R}^{n\times d}$ with $\mathbb{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^T$, such that $\delta_{ij} \approx \|\mathbf{x}_i - \mathbf{x}_j\|$. Observe that then

$$\delta_{ij}^2 \approx \|\mathbf{x}_i - \mathbf{x}_j\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = \|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\mathbf{x}_j + \|\mathbf{x}_j\|^2,$$

whence it is natural to introduce the Gram matrix $\mathbb{B} = \mathbb{X}\mathbb{X}^T$ of inner products $b_{ij} = \mathbf{x}_i^T\mathbf{x}_j$. Assuming that the above relations are exact equalities, it can be shown that

$$b_{ij} = -\frac{1}{2}\left(\delta_{ij}^2 - \frac{1}{n}\sum_{r=1}^{n}\delta_{rj}^2 - \frac{1}{n}\sum_{s=1}^{n}\delta_{is}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n}\delta_{rs}^2\right),$$

so that $\mathbb{B}$ can also be obtained by double-centering the matrix of squared dissimilarities $\Delta^{(2)}$.

The Gram matrix $\mathbb{B}$ is positive semidefinite and thus possesses the spectral decomposition $\mathbb{B} = U\Lambda U^T$, where Λ is the diagonal matrix of the eigenvalues of $\mathbb{B}$ and U is the orthonormal matrix of its eigenvectors. Then the optimal choice of the matrix $\mathbb{X}$ in the MDS algorithm is

$$\mathbb{X} = U_{(d)}\Lambda_{(d)}^{1/2},$$

where $\Lambda_{(d)} \in \mathbb{R}^{d\times d}$ is the diagonal matrix of the d largest eigenvalues of $\mathbb{B}$ and $U_{(d)} \in \mathbb{R}^{n\times d}$ is an $n \times d$ matrix of respective eigenvectors.

5.2.3 Iterative improvement

After finding the MDS embedding $V \to \mathbb{R}^d$ of the graph's vertices v_j as points $\mathbf{x}_j$ in d-dimensional space, we determine the centroid $\bar{\mathbf{x}}$ of these points and find the point $\mathbf{x}_j$ that is the closest to the centroid. The corresponding node v_j is considered as an approximate central graph node c. To determine the actual node minimizing the total paths' length, we propose a simple greedy-type algorithm of checking the adjacent nodes to the previously suggested centre c.

We begin with calculating all shortest paths from the centre node c using the gravitational potential with source c. Next, we check if a node i adjacent to the current centre c may have smaller total paths length. This can be done directly by

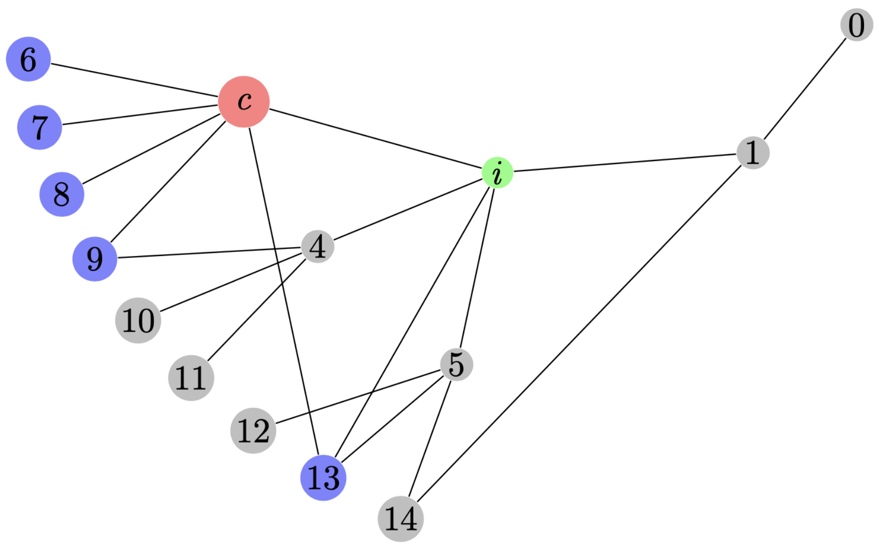


FIGURE 5.1: Path optimization example

re-calculating all shortest paths on the graph from i; however, we propose a simple and less computationally expensive method instead.

Let i be one of the nodes adjacent to the current centre candidate c. All vertices $v \in V \setminus \{c, i\}$ can be split into two groups: v belongs to the first group, if the shortest path from v to c passes through i; for v in the second group, the shortest path from v to c avoids i. By moving the centre from c to i, we decrease the shortest paths to the centre from v in the first group by w_{ic}. For v in the second group, the shortest path to the new centre can increase by at most w_{ic}. Therefore, the cardinalities of the two groups predict if moving the current centre c to i can decrease the total shortest paths length and by what amount, thus suggesting the next iteration.

For a better understanding of the general idea, consider the unweighted graph in Fig. 5.1. Moving the centre from c to i will decrease the length of the shortest paths to the new centre for the vertices coloured grey by no less than one and will increase the length of the shortest path to the new centre by no more than one for the vertices coloured blue. Therefore, the total path sum will decrease by no less than three.

After selecting the vertex j having the minimum approximated total shortest paths' length among all vertices adjacent to c, we consider this vertex j the new current centre c. The procedure described above should be continued until no further improvement is possible.

To sum up, we proposed an algorithm for optimizing the total shortest paths length, which consists of two main steps, finding the approximate centre of the graph based on the MDS algorithm and iterative improvement of the current centre candidate. The iterative part of the algorithm can not make more than $d/2$ iterations (where d is the diameter of the graph) in the highly unlikely case of very poor initial centre approximation. The extensive experiments we conducted for various graph configurations demonstrate that it typically takes no more than three iterations to find the local optimum.

Chapter 6

Formulation and performance of the algorithm

In the previous chapters, we described the two main steps of the general network optimization pipeline. One is that of finding nearly the shortest paths to the closest servers using the gravitational potential approach; this automatically suggests the allocation of the clients to the closest servers. The second step discussed above is that of finding the nearly optimal server locations within the corresponding clusters using the initial guess suggested by the MDS and its iterative improvement. In this chapter, we want to combine all of the network optimization steps into one algorithm and evaluate its performance on different graphs.

After performing the deterministic assignment of clients to the servers in Section 4.4, the only final step left is to choose the optimal locations of servers. Every server with the clients allocated to it is considered a separate subgraph (whose adjacency matrix is formed by taking the corresponding rows and columns from the original one), and we perform the procedure of path optimization described in detail in Chapter 5 for every such subgraph separately.

The entire algorithm of network optimization can be described in the following pseudocode:

Algorithm 2: Network optimization algorithm

```
  Data: Adjacency matrix of the graph, number of servers ($k$)
  Result: Indices of servers' locations
1 Random initialization of server;
2 while criteria minimum is not achieved do
3     construct gravitational potential;
4     construct gradient flows;
5     build probability matrix and calculate criteria;
6     interchange the node with the biggest criteria value with the node, where
        gravitational potential assumes its local maximum on the largest
        disconnected part of the graph
7 Perform path optimization
```

6.1 Performance of the algorithm

In this section, we demonstrate the performance of the developed network optimisation algorithm and compare its performance with that of the optimal solution obtained by the brute-force method. The comparison is with respect to the two main tasks.

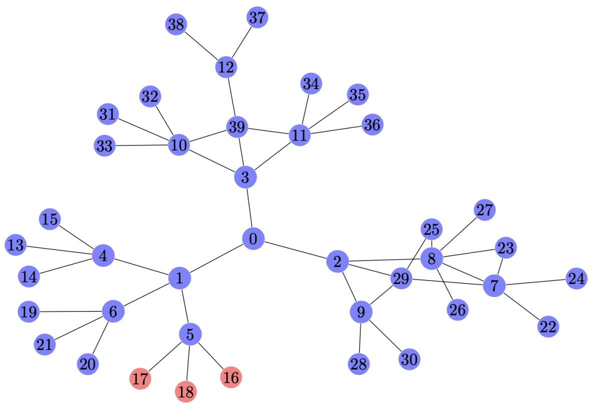


FIGURE 6.1: Initial server location, $\boldsymbol{J} = 0.520$

Firstly, from all possible servers' locations on the graph, we select the one with the lowest value $\boldsymbol{J}$ and compare it to the $\boldsymbol{J}$ score of our optimization algorithm.

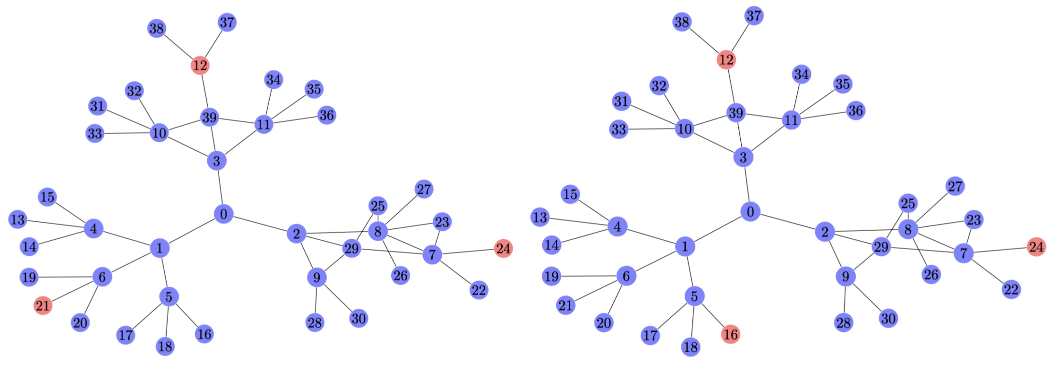
Afterwards, for the fixed splitting into subgroups, we find the optimal locations of the servers. This is done separately for each group, by running over all nodes in the group and choosing the node v with the smallest total paths length to v from all other nodes in the group; this value is compared to the one obtained by the path optimisation procedure. We report the mean shortest path length as it offers better interpretability of the results.

We want to emphasize that due to the present randomness of the algorithm, the results may vary depending on the starting conditions; thus, it is natural (especially for large graphs) to run this algorithm several times with different initial conditions and take the average value as the performance score.

As was observed in the experiments, the weights of the graph only slightly influence the server-client group distribution but may influence the the path optimization procedure, final server localizations, and mean shortest path length. This is well seen on the examples of tree-structured graphs 6.1.1.

6.1.1 Application of the algorithm on trees

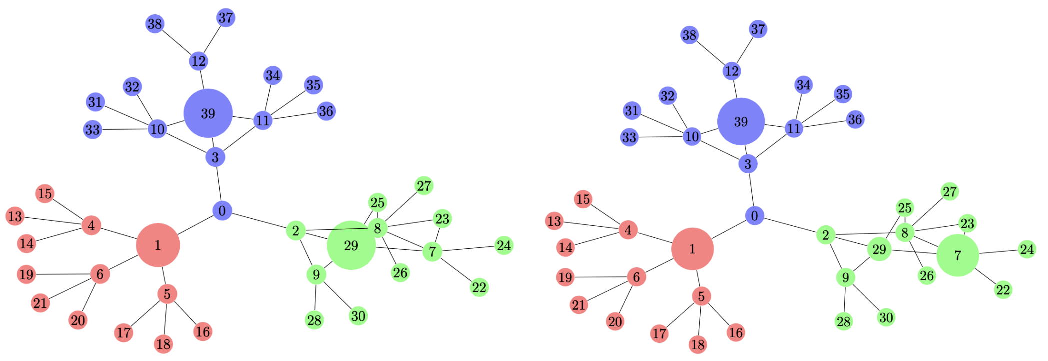
Consider a graph created from a balanced ternary tree with a slight modification of the edges 6.1. We locate the servers initially at the nodes 16, 17, 18 (marked red), which are far from the optimal distribution. Let us compare the performance of the algorithm on the same graph with identical initial conditions, but different edges weights. For the weighted graph, we randomly select weights from range $[0.5, 1]$.



(A) Unweighted tree, $J = 0.010$ (B) Weighted tree, $J = 0.010$

FIGURE 6.2: Client-server optimization step

As can be seen from Fig. 6.2 above, the values of the J-score for both weighted (Fig. 6.2a) and unweighted (Fig. 6.2b) graphs are the same, but the server locations differ. The visualization of the final step of path optimization can be seen on Fig. 6.3. As expected, adding graph weights did not change the server-client groups' distribution, but did change the allocation of the servers.



(A) Unweighted graph (B) Weighted graph

FIGURE 6.3: Path optimization step

Table below contains the results obtained by our algorithm and the optimal brute-force search:

	Unweighted graph		**Weighted graph**	
	Obtained	Optimal	Obtained	Optimal
J	0.010	0.010	0.010	0.010
Blue cluster mean path	1.571	1.571	1.097	1.020
Red cluster mean path	1.615	1.615	1.157	1.138
Green cluster mean path	1.692	1.538	1.228	1.220

TABLE 6.1: Scores for the near-ternary tree graph

For this graph, our algorithm achieved optimal result of server efficiency optimization and does not change the score after including the weights. However, in the latter case the performance of the path optimization algorithm got slightly lower. The **MAPE** score between the clusters' mean path, obtained by the algorithm and optimal ones for the unweighted graph was 0.0303, and for weighted one it became 0.0308.

6.1.2 Application of the algorithm on general graphs

In this subsection, we want to demonstrate the algorithm efficiency on graphs of diverse structures.

Random Powerlaw Tree

The graph is generated from a power law degree sequence, where elements are swapped with new elements from a powerlaw distribution until the sequence makes a tree [17]. Initially, the servers are located at the nodes 5, 10, 20, 34 (marked red). As could be predicted, due to the tree-structure of the graph, the path optimization achieved the optimal result. The obtained result for the server efficiency optimization is greater from the optimal one by 7.9%.

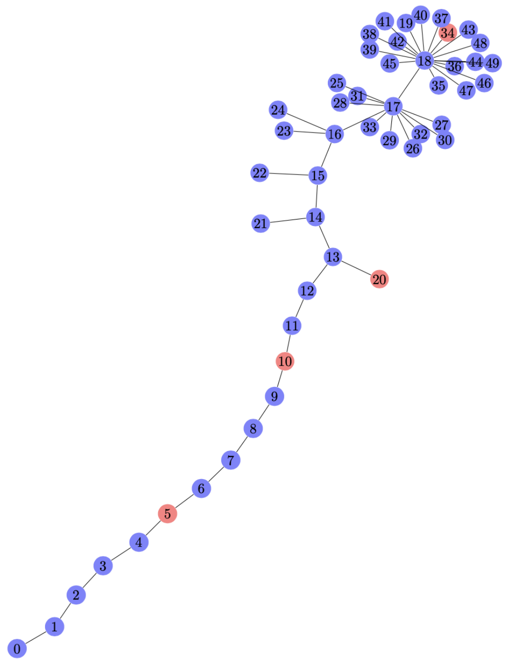
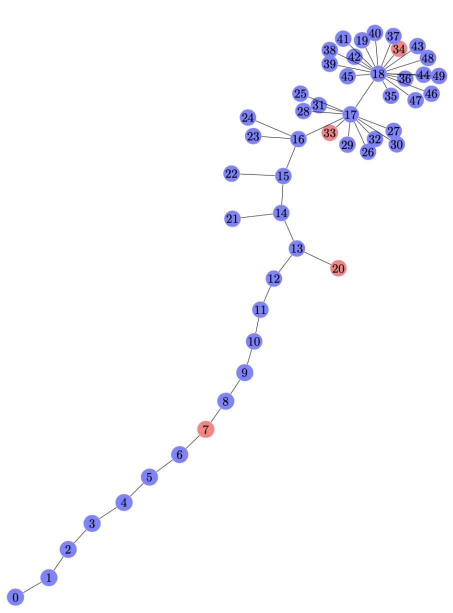


FIGURE 6.4: The Powerlaw graph: initial server location, $J = 0.263$


(A) $J = 0.051$

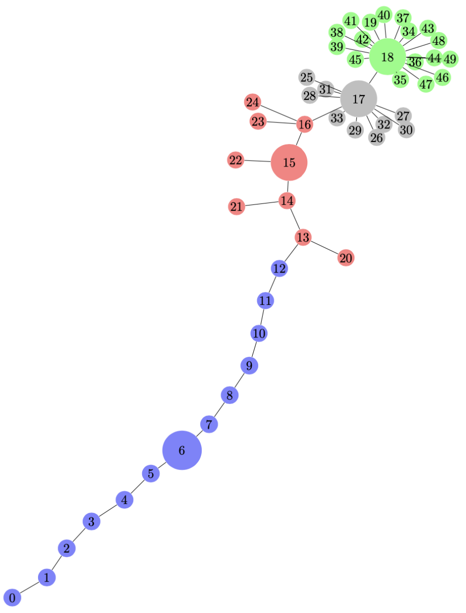

(B) Server location

FIGURE 6.5: Client-sever and path optimization steps

Performance of the algorithm is summarized in Table 6.2 below.

	Random Powerlaw Tree	
	Obtained	Optimal
J	0.051	0.047
Blue cluster mean path	3.231	3.231
Red cluster mean path	1.556	1.556
Green cluster mean path	0.944	0.944
Gray cluster mean path	0.900	0.900

TABLE 6.2: Scores for the Powerlaw graph

Tutte Graph

The Tutte graph is a cubic polyhedral, non-Hamiltonian graph, which has 46 nodes and 69 edges. [18]

We initially located the servers randomly at the nodes $1, 2, 20, 41$ (marked red). The algorithm achieved the optimal result for server efficiency optimization, while the path optimization step attained a 0.105 **MAPE** score in comparison with the optimal one.

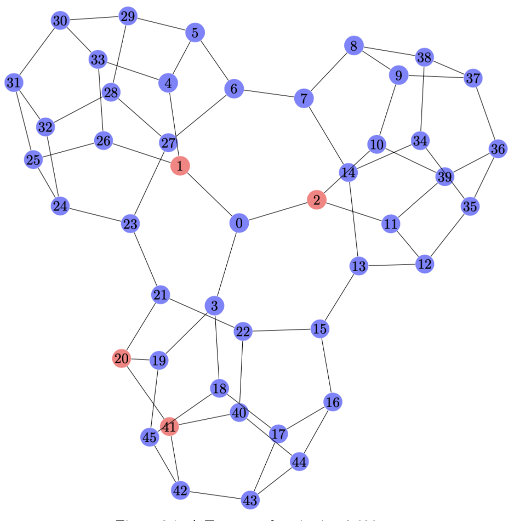
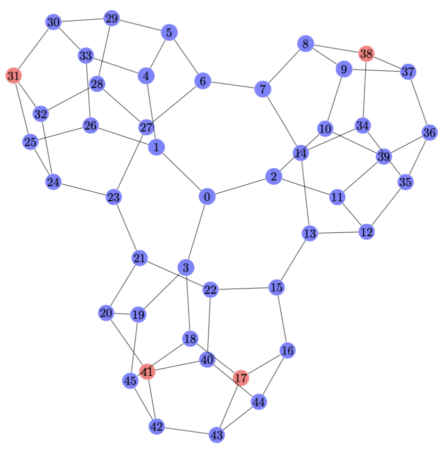

(A) $J = 0.343$

FIGURE 6.6: Initial server location, $J = 0.686$


(A) $J = 0.014$

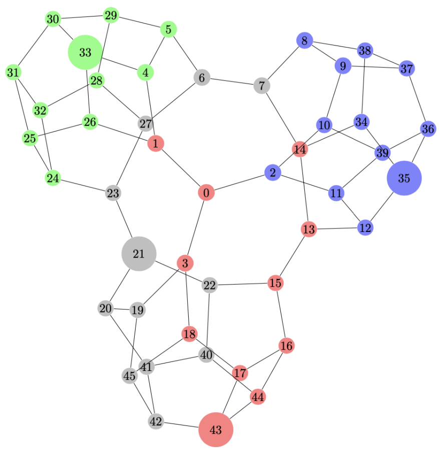

(B)

FIGURE 6.7: Client-sever and path optimization steps

Summary of the performance is described in Table 6.4.

	Tutte Graph	
	Obtained	Optimal
J	0.014	0.014
Blue cluster mean path	1.917	1.750
Red cluster mean path	2.727	2.091
Green cluster mean path	1.818	1.636
Gray cluster mean path	2.0	2.0

TABLE 6.3: Scores for the Tutte graph

Chordal Cycle Graph

The Chordal Cycle graph is a cycle graph on n nodes with chords joining each vertex to its inverse modulo n, where n is a prime number [19].

We initially located the servers randomly at the nodes 3, 5, 23 (marked red). The algorithm achieved optimal result for server efficiency optimization and the path optimization part attained a 0.061 **MAPE** score in comparison to the optimal one.

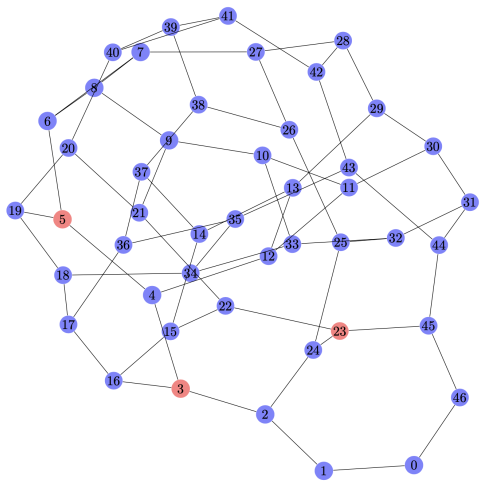
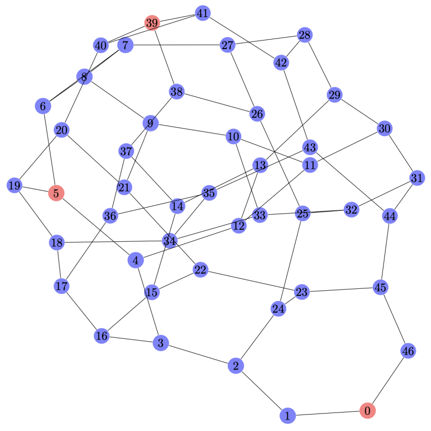

(A) $J = 0.366$

FIGURE 6.8: Initial server location, $J = 0.366$


(A) $J = 0.008$

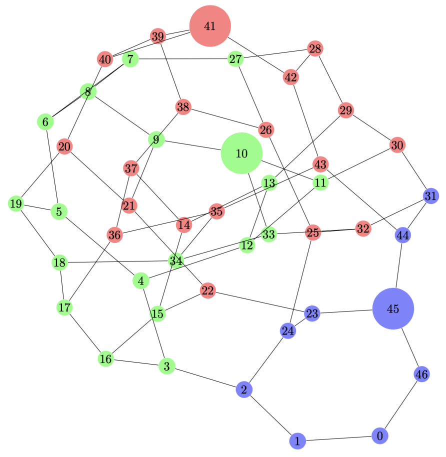

(B)

FIGURE 6.9: Client-sever and path optimization steps

Summary of the performance is described in the table:

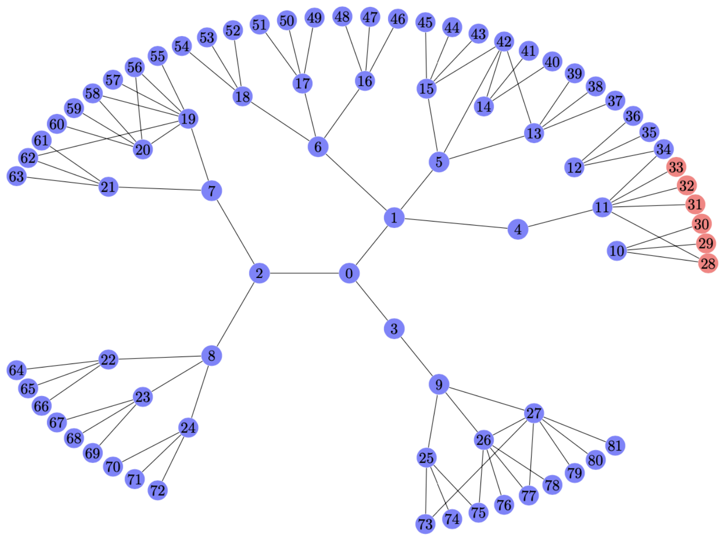
	Chordal Cycle Graph	
	Obtained	Optimal
J	0.008	0.008
Blue cluster mean path	1.667	1.667
Red cluster mean path	2.684	2.632
Green cluster mean path	2.895	2.421

TABLE 6.4: Scores for the Chordal Cycle Graph

6.1.3 Application of the algorithm on large graphs

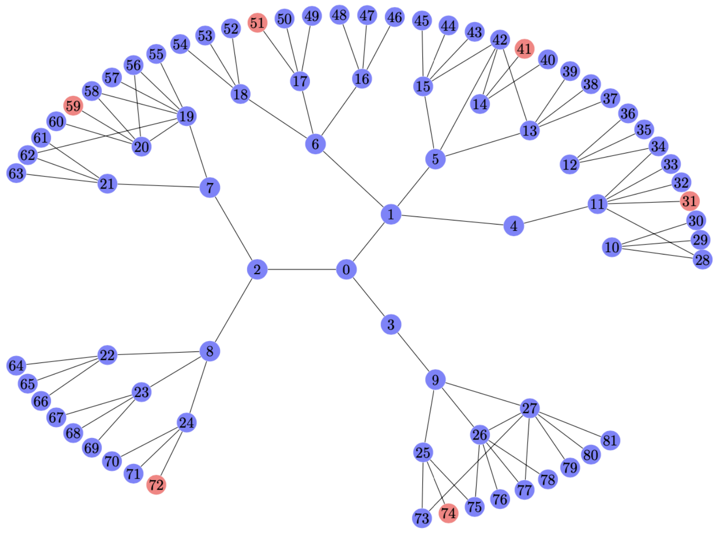
Another exciting demonstration of the performance of our algorithm is its application to large graphs. As for those graphs, we can not compute the optimal result; it is wise to construct such graphs with known structure so that we can predict the optimal allocations of the servers. Thus, in this chapter, we will illustrate the performance of the algorithm's first part (server efficiency optimization).

For example, consider an unbalanced tree-based graph with a fraction of branches power approximately 3:2:1 (6.10a). It is evident that for six servers, every one of them should be allocated to a different branch. As can be seen from the figure 6.11a, the algorithm does so.



(A) $J = 0.525$

FIGURE 6.10: Initial locations of the servers



(A) $J = 0.006$

FIGURE 6.11: Client-server optimization step

After examining the algorithm's performance, a natural question arises if we can use the values of criteria in order to determine the possible number of servers for efficient optimization. In many cases, adding extra servers does not improve the network performance, as due to the network architecture, some servers will not be used effectively. Also, after exceeding some maximum number of servers, expanding their number will only worsen the overall functioning of the network.

Consider the same unbalanced graph as in the previous example. After testing a different number of servers, we obtain the following dependency, see Fig. 6.12.

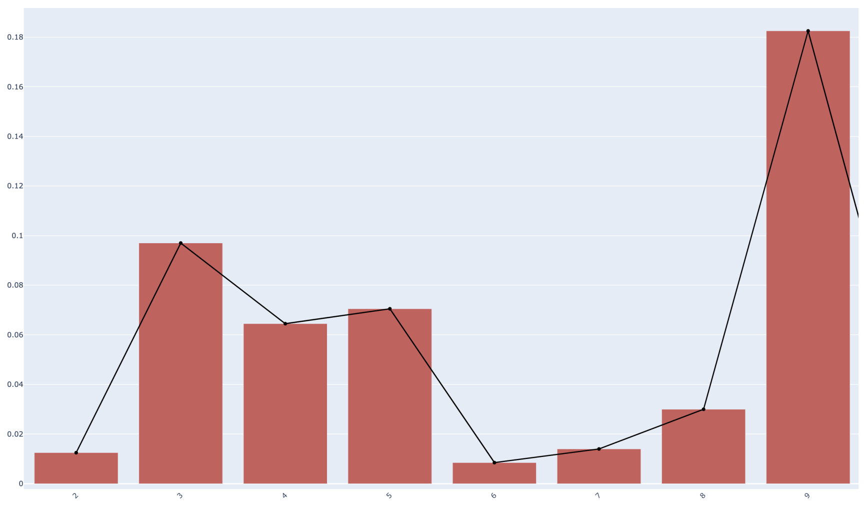


FIGURE 6.12: Dependence between values of J and K for 3-r unbalanced tree graph

As we can see, the optimum is achieved for $K = 6$ and increasing the server number only makes the J value bigger. As the graph has 82 vertices, making a significant number of servers will increase the number of common clients between the servers and, therefore, will increase the J value.

Consider now a large graph with 1000 vertices based on a 3-r balanced tree. It is natural to assume that optimum should be achieved for values of K, which are multiples of three.

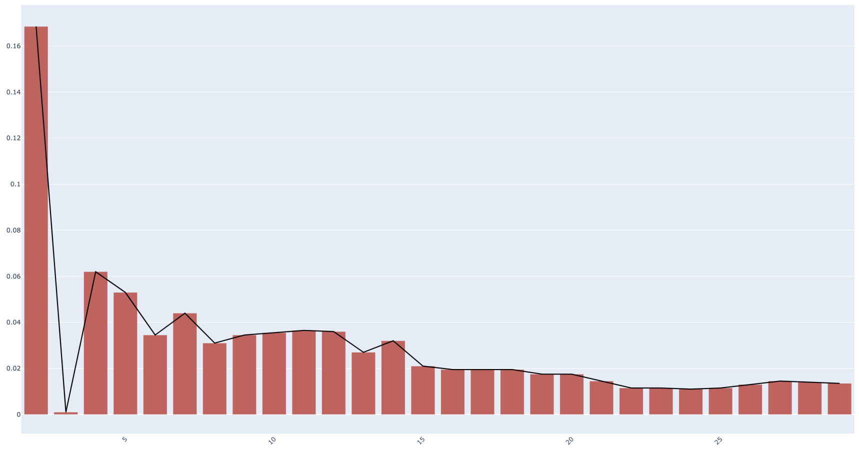


FIGURE 6.13: Dependence between values of J and K for 3-r balanced tree graph

As we can see from Fig. 6.13, the first optimum is achieved for $K = 3$, then for $K = 6$, $K = 15$, $K = 21$ and $K = 24$, which agrees with the prediction that K must be multiple of 3.

Chapter 7

Conclusions

The main aim of this thesis work was to study the network optimization task to find the best locations for several servers in the nodes of the network. The ultimate goal is to minimize the total path lengths that the clients should cover to reach the service point and reach uniform server loads. This task involves several optimization subtasks of different nature, and we are aware of no single algorithm that could efficiently solve them.

In this thesis, we developed a novel approach for solving this problem, which we called the gravitational potential method. It is based on the spectral properties of the related graph Laplacian and is inspired by the paper [10], where a spectral approach to the shortest path problem for unweighted graphs was suggested. We showed how the gravitational potential helps to determine a balanced clustering of the network into service areas and then to place the server in each in the nearly optimal way. Although the suggested approach is not guaranteed to result in the optimal solution, finding the latter would require the brute-force search at least on some stages and is doomed inefficient. On the contrary, we showed that the information carried by gravitational potential helps to achieve local minima just in a few iteration steps and produce solutions that are nearly optimal. The experiments conducted on networks of various types and sizes confirm efficiency of the developed algorithm.

Appendix A

Exact value of the norming coefficient for the load criterion

We need to find the maximum value of the function $f(\mathbf{x}) = \sum\limits_{i=1}^{k} \left(x_i - \frac{1}{k}\right)^2$, subect to the constraints $x_i \geq 0$, $i = 1, 2, \ldots, k$, and $x_1 + \cdots + x_k = 1$.

Applying the Lagrange multipliers method, we construct the Lagrangian:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \lambda_0 \frac{1}{2} \sum_{i=1}^{k} \left(x_i - \frac{1}{2}\right)^2 + \lambda_1 \left(\sum_{i=1}^{k} x_i - 1\right) + \sum_{i=1}^{k} \mu_i x_i.$$

The necessary condition for the maximum reads:

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0} \implies \begin{cases} \lambda_0 \left(x_1 - \frac{1}{k}\right) + \lambda_1 + \mu_1 = 0 \\ \quad \vdots \\ \lambda_0 \left(x_k - \frac{1}{k}\right) + \lambda_1 + \mu_k = 0 \end{cases}$$

$$\sum_{i=1}^{k} x_i = 1,$$

$$\mu_i x_i = 0, \qquad i = 1, 2, \ldots, k.$$

It is clear that $\lambda_0 > 0$; otherwise, $-\lambda_1 = \mu_1 = \cdots = \mu_k$ are either zero, in which case the Lagrangian is identically zero (which makes no sense), or $x_1 = x_2 = \cdots = x_k = 0$, and the above system is inconsistent. For convenience, set $\lambda_0 = 1$; then we obtain the following system:

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = 0 \implies \begin{cases} x_1 - \frac{1}{k} + \lambda_1 + \mu_1 = 0 \\ \quad \vdots \\ x_k - \frac{1}{k} + \lambda_1 + \mu_k = 0 \end{cases}$$

$$\sum_{i=1}^{k} x_i = 1,$$

$$\mu_i x_i = 0, \qquad i = 1, 2, \ldots, k.$$

Multiplying the i^{th} equation in the above system by x_i, we get:

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = 0 \implies \begin{cases} x_1^2 - \frac{x_1}{k} + \lambda_1 x_1 = 0 \\ \vdots \\ x_k^2 - \frac{x_k}{k} + \lambda_1 x_k = 0 \end{cases}$$

Assuming that at the point $\mathbf{x}$ of maximum exactly m components of the vector $\mathbf{x}$ are non-zero, $m < k$, we deduce that all non-zero components are then equal, and the constraint $\sum_{i=1}^{k} x_i = 1$ implies that they are equal to $\frac{1}{m}$. It remains to determine the value of m that maximises the function $f(\mathbf{x}) = \sum\limits_{i=1}^{k} \left(x_i - \frac{1}{k}\right)^2$ under the derived constraints on x_i; namely, the task is

$$f(\mathbf{x}) = m\left(\frac{1}{m} - \frac{1}{k}\right)^2 + (k-m)\frac{1}{k^2} \to \max .$$

Since

$$m\left(\frac{1}{m} - \frac{1}{k}\right)^2 + \frac{k-m}{k^2} = \frac{1}{m} - \frac{1}{k},$$

the maximum is achieved for $m = 1$.

Therefore, we conclude that the maximal value of $f(\mathbf{x})$ under the imposed restrictions is equal to $\frac{k-1}{k}$.

Bibliography

[1] GitHub Repository

[2] Barabási, Albert-László. *Network Science*, Cambridge University Press, Cambridge, 2016.

[3] M. E. J. Newman, Mark. *Networks: An Introduction,* 2$^{\text{nd}}$ *ed.*, Oxford University Press, Oxford, 2018.

[4] Freeman, Linton. "A set of measures of centrality based on betweenness". *Sociometry* **40**(1) (1977): 35–41.

[5] Shi, Jianbo and Malik, Jitendra. "Normalized Cuts and Image Segmentation", *IEEE Transactions on PAMI*, **22** (8) (2000): 888–905.

[6] Ng, Andrew Y., Jordan, Michael I., and Weiss, Yair. "On spectral clustering: analysis and an algorithm", *Advances in Neural Information Processing Systems*, **2** (2002): 1–8.

[7] Sieranoja, Sami and Fränti, Pasi. "Adapting *k*-means for graph clustering", *Knowledge and Information Systems*, **64**(1) (2022): 115–142.

[8] Ester, Martin, Kriegel, Hans-Peter, Sander, Jörg, and Xu, Xiaowei. "A density-based algorithm for discovering clusters in large spatial databases with noise", Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), (1996): 226–231.

[9] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., and Stein, Clifford. *Introduction to Algorithms (*4$^{\text{th}}$ *ed.)*, MIT Press, Oxford, MA, 2022.

[10] Steinerberger, Stefan. "A spectral approach to the shortest path problem." *Linear Algebra and its Applications* **620** (2021): 182–200.

[11] Gallier, Jean. "Spectral theory of unsigned and signed graphs. Applications to graph clustering: a survey." arXiv preprint arXiv:1601.04692 (2016).

[12] Chung, Fan R. K. *Spectral graph theory*, CBMS Regional Conference Series in Mathematics, **92**, American Mathematical Society, Providence, RI, 1997.
Spectral graph theory (revised and improved), Chapter 1: Eigenvalues and the Laplacian of a graph

[13] Marsden, Anne. "Eigenvalues of the Laplacian and their relationship to the connectedness of a graph." University of Chicago, REU (2013).

[14] Kupitz, Yaakov S., Martini, Horst, and Spirova, Margarita. "The Fermat–Torricelli Problem, Part I: A Discrete Gradient-Method Approach." *J. Optim. Theory Appl.* **158** (2013), 305–327.

[15] Klimenta, Mirza, and Ulrik Brandes. "Graph drawing by classical multidimensional scaling: new perspectives." *Graph Drawing: 20th International Symposium, GD 2012*, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers 20. Springer Berlin Heidelberg, 2013.

[16] Brandes, Ulrik, and Christian Pich. "Eigensolver methods for progressive multidimensional scaling of large data." *Graph Drawing: 14th International Symposium, GD 2006*, Karlsruhe, Germany, September 18-20, 2006. Revised Papers 14. Springer Berlin Heidelberg, 2007.

[17] Random Powerlaw Tree, in *NetworkX package documentation*

[18] Tutte Graph, in *NetworkX package documentation*

[19] Chordal Cycle Graph, in *NetworkX package documentation*