Universidad Nacional de Asunción
Facultad Politécnica

Maestría en Ciencias de la Computación

OPTIMALITY OF SPECTRAL SPARSIFICATION IN DISTRIBUTED SYSTEMS WITH APPLICATION TO DATA CLUSTERING

FABRICIO AUGUSTO MENDOZA GRANADA

Trabajo de Master presentado en conformidad a los
requisitos para obtener el grado de Master en Ciencias de la Computación

San Lorenzo - Paraguay
November - 2020

Universidad Nacional de Asunción
Facultad Politécnica

Maestría en Ciencias de la Computación

OPTIMALITY OF SPECTRAL SPARSIFICATION IN DISTRIBUTED SYSTEMS WITH APPLICATION TO DATA CLUSTERING

FABRICIO AUGUSTO MENDOZA GRANADA

San Lorenzo - Paraguay
November - 2020

Universidad Nacional de Asunción
Facultad Politécnica

Maestría en Ciencias de la Computación

OPTIMALITY OF SPECTRAL SPARSIFICATION IN DISTRIBUTED SYSTEMS WITH APPLICATION TO DATA CLUSTERING

FABRICIO AUGUSTO MENDOZA GRANADA

Advisor:
Prof. Dr. Marcos Daniel Villagra

Trabajo de Maestría presentado en conformidad a los
requisitos para obtener el grado de Master en Ciencias de la Computación

San Lorenzo - Paraguay
November - 2020

To my mother, father
and sister.

ACKNOWLEGEMENTS

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to acknowledge the "Consejo Nacional de Ciencia y Tecnología (Conacyt)" for granting me a scholarship that allow me to finish this graduate course.

Secondly, I like to thank my supervisor, Professor Dr. Marcos Villagra, whose expertise and methodology was invaluable in formulating the research questions. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

Big thanks also goes to the staff of researchers and professors at the "Núcleo de Investigación y Desarrollo Tecnológico (NIDTEC)" from the Polytechnic School whose expertise and patience help to go through this academic program.

I have been also benefited from conversations and discussions with several colleagues from NIDTEC which served me as insightful distractions for cleaning my mind and thinking clearer.

I like to thank to Professor Dr. Inocencio Ortiz for his comments and suggestions and for the time and help he has dedicated to improve the writing of this thesis.

And finally, I would like to thank my parents for their support through this entire learning process.

OPTIMALITY OF SPECTRAL SPARSIFICATION IN DISTRIBUTED SYSTEMS WITH APPLICATION TO DATA CLUSTERING

Autor: FABRICIO AUGUSTO MENDOZA GRANADA

Tutor: Prof. Dr. MARCOS DANIEL VILLAGRA

RESUMEN

La técnica de Dispersión Espectral es utilizada en una variedad de aplicaciones tales como resolución de Sistemas Laplacianos y búsqueda de multicortes en un grafo mediante sus propiedades espectrales. Esencialmente, aproxima el espectro de un grafo por un factor constante reponderando algunas de sus aristas y eliminando otras. En la literatura existen algoritmos determinísticos así como probabilísticos para encontrar dispersores espectrales de grafos. Dado las múltiples aplicaciones en sistemas distribuidos, es de gran interés encontrar dispersores espectrales en tales sistemas. En este trabajo, demostramos que la dispersión espectral funciona bajo la suposición de que los datos de entrada están distribuidos en diferentes sitios. Para lograr esto, introducimos una herramienta matemática que captura la noción de solapamiento de datos entre diferentes sitios en un sistema distribuido. Además, describimos un protocolo que computa un dispersor espectral de un grafo $G = (V, \cup_{i=1}^{s} E_i)$ cuyas aristas, representadas por $\{E_i\}_{i \leq s}$, están alojadas en s diferentes sitios. El resultado del protocolo es un grafo $H = (V, \cup_{i=1}^{s} \hat{E}_i)$ donde H es un dispersor espectral de G y $\hat{E}_i$ es el conjunto de aristas reponderadas en el sitio i.

La idea de datos que se solapan entre diferentes sitios nos ha inspirado a estudiar modelos de comunicación, los cuales funcionan como herramientas teóricas para estudiar algoritmos que trabajan con datos distribuidos. En particular, nos hemos enfocado en el modelo *Number-On-Forehead*, el cual es una poderosa herramienta con aplicaciones teóricas en complejidad de circuitos. Además, desarrollamos un protocolo que detecta si una familia dada de conjuntos de puntos constituye una estructura combinatorial particular llamada *Sistema-*Δ. Un *Sistema-*Δ es una familia de conjuntos con interesección única. Nuestro protocolo requiere a lo más 3 bits de comunicación.

Finalmente, desarrollamos además un protocolo que aproxima el multicorte de un grafo dado en el modelo *Number-On-Forehead* bajo la suposición de que la familia de subconjuntos de aristas constituye un *Sistema-*Δ. Nuestro protocolo tiene un costo de comunicación de $O(\log(\frac{n}{\epsilon^2}\sqrt{1-\delta}))$ donde ϵ es el factor de aproximación del dispersor espectral y δ es un coeficiente que captura la mayor cantidad de datos solapados entre los sitios.

Palabras Clave: 1. dispersión espectral. 2. grafos densos. 3. algoritmos distribuidos. 4.

complejidad de la comunicación. 5. agrupamiento de datos.

OPTIMALITY OF SPECTRAL SPARSIFICATION IN DISTRIBUTED SYSTEMS WITH APPLICATION TO DATA CLUSTERING

Author: FABRICIO AUGUSTO MENDOZA GRANADA

Advisor: Prof. Dr. MARCOS DANIEL VILLAGRA

ABSTRACT

Spectral sparsification is a technique used in a variety of applications such as solving Laplacian systems and finding multicuts in graphs. Essentially, spectral sparsification approximates the spectra of a given graph up to a constant factor by reweighting some edges and deleting others. There are deterministic as well as probabilistic algorithms in the literature for finding spectral sparsifiers of graphs. Due to the several applications of graphs in distributed systems, it is of great interest to find spectral sparsifiers in such systems. In this work, we demonstrated that spectral sparsification works under the assumption that the entire data set is distributed among several sites. In order to do that, we introduced a mathematical tool that captures the notion of overlapping data among different sites in a distributed system. As a result, we constructed a protocol that computes an spectral sparsifier of a given graph $G = (V, \cup_{i=1}^{s} E_i)$ whose edges, represented by $\{E_i\}_{i \leq s}$, are allocated in s different sites. The result of the protocol is a graph $H = (V, \cup_{i=1}^{s} \hat{E}_i)$ where H is an spectral sparsifier of G and $\hat{E}_i$ is the set of reweighted edges of site i.

The idea of data that overlaps among different sites has inspired us to study models of communication, which serves as a theoretical tool to study algorithms that works with distributed data. In particular, we focused in the *Number-On-Forehead* model, which is a model with theoretical applications in circuit complexity. Furthermore, we developed a protocol that detects if a given family of sets of points constitutes a particular combinatorial structure called Δ-System. A Δ-System is a family of sets with unique intersection. Our protocol requires at most 3 bits of communication.

Finally, we developed a protocol that approximates a multicut of a given graph in the *Number-On-Forehead* model under the assumption that the family of set of edges is a Δ-System. Our protocol has a communication cost of $O(\log(\frac{n}{\epsilon^2}\sqrt{1-\delta}))$ where ϵ is the spectral sparsifier approximation factor and δ is a coefficient that captures the greatest amount of overlapping data among the sites.

Keywords: 1. spectral sparsification. 2. dense graphs. 3. distributed algorithms. 4. communication complexity. 5. data clustering.

TABLE OF CONTENTS

Page

1 INTRODUCTION **1**
1.1 Introduction . . . 1
1.2 Justification and Motivation . . . 2
1.3 Objectives . . . 2
1.3.1 General Objective . . . 2
1.3.2 Specific Objectives . . . 2
1.4 Book Organization . . . 2

2 SPECTRAL GRAPH THEORY AND SPECTRAL SPARSIFICATION **4**
2.1 Graphs . . . 4
2.2 Spectral Graph Theory . . . 6
2.3 Spectral Sparsification . . . 11

3 OVERLAPPING CARDINALITY PARTITION **12**
3.1 Overlapping Cardinality Partition . . . 12
3.2 Input Data Graph and Laplacian Decomposition . . . 15
3.3 Constructing Spectral Sparsifiers from Induced Subgraphs . . . 19

4 SUNFLOWERS AND NUMBER-ON-FOREHEAD COMMUNICATION **22**
4.1 Communication Protocols . . . 22
4.2 Multiparty Communication Models and Number-On-Forehead . . . 24
4.3 Δ-Systems in NOF model . . . 26
4.4 Communication Complexity Applications . . . 30

5 APPLICATIONS ON THE MULTICUT GRAPH PROBLEM **33**
5.1 Clustering . . . 33
5.2 Graph Clustering and Spectral Clustering . . . 35
5.3 Data Clustering with Sunflowers . . . 39

6 CONCLUSIONS AND FUTURE WORK **42**

BIBLIOGRAPHY **45**

Appendices **49**

A Sets, Functions and Asymptotic Notation **50**
A.1 Sets . . . 50
A.2 Functions . . . 52
A.3 Asymptotic Notation . . . 53

B Linear Algebra **54**
B.1 Spectral Theory 54
B.2 Spectral Theory of Symmetric Matrices 55

LIST OF FIGURES

Page

2.1 A Graph and its Laplacian matrix 7
2.2 Spectral Sparsification Illustration 10

3.1 Input Data from Multiples Sources 13
3.2 Example of Overlapping Cardinality Partition 15
3.3 Enumerating Process 16
3.4 Input Data Graphs 17
3.5 Union of Spectral Sparsifiers 21

4.1 A Communication Protocol Tree Example 24
4.2 Protocol Tree for the Equality Problem on the NOF Model 26
4.3 Protocol Tree for Solving the Δ-System Problem 29

5.1 Clustering Classification Tasks 34
5.2 Sunflower in the NOF Model 39

LIST OF SYMBOLS

Common symbols and notations

$\{x \mid Q(x)\}$	set description
$x \in X$	x belongs to X
$X \subseteq Y$	X is a subset of Y
$X \subset Y$	X is a proper subset of Y
$\lvert X \rvert$	set cardinality
$\{X_i\}_{i \leq m}$	family of m sets
$\cup$	union
$\cap$	intersection
$-$	difference
$\bigcup_{i=0}^{n} X_i$	finite union of sets
$\bigcap_{i=0}^{n} X_i$	finite intersection of sets
Δ	symmetric difference
$\emptyset$	empty set
$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}_{\geq 0}$	set of positive real number and zero
$[n]$	set $\{1, \ldots, n\}$
$\min_{x \in S}$	minimum element of a set S
$\max_{x \in S}$	maximum element of a set S
$M_1 \preccurlyeq M_2$	$0 \leq x^T M_2 x - x^T M_1 x$, where M_1, M_2 are $n \times n$ matrices
$\neg$	negation
$\Rightarrow$	implication
$\Longleftrightarrow$	if and only if
M^T	transpose of a matrix M
G	graph
L_G	Laplacian matrix of graph G
V	set of vertices
E	set of edges
ι_G	isoperimetric number of G
ϕ_G	conductance of G
$\partial(S)$	boundary of $S \subseteq V$
χ_S	characteristic vector of a subset $S \subseteq V$
$\#(x)$	occurrence number of an element x in a family of sets
v	vertex
e	edge
$\binom{n}{k}$	binomial coefficient or "n choose k"
$\sum_{i=0}^{n} a_i$	summation
(a, b)	ordered pair
$O(f(n))$	Big-Oh

CHAPTER 1

INTRODUCTION

1.1 Introduction

Graph theory plays a fundamental role in the design of efficient algorithms. It is a transversal area in computer science. Graphs are widely used to model discrete relationships between objects into fixed systems. Moreover, many properties of graphs can be further analyzed by means of their algebraic objects associated to them such as matrices. Its applications cover a great variety of subjects in different areas, and machine learning is one of them. Due to the highly increase of data storage, *dense* graphs are usually encounter in many applications. This may yield to slowly processing of data and derive in non-efficient applications. For this reason, approximation of graphs is of great interest in the graph theory community. This approximation depends upon the properties of study. In this work we will be interested in approximating graphs by their spectra. Such approximations have great impact in the development of efficient algorithms for machine learning such as *clustering* [1] and *principal component analysis* [2, 3, 4] communities as we shall see.

The design of algorithms has not been always related to one computational model, and as technology evolves new computational models appear as well. This thesis studies a computational model that does not take place into a single physical site. Moreover, every site may represent a computer with the power to execute any kind of algorithm. Typical resources used to measure the efficiency of algorithms such as *time* and *memory* are no longer useful to analyze. This may bring in new challenges when designing efficient algorithms. For example, the input may not be available for all sites at the same time, so they will need to communicate among them. This yields the algorithm designer propose solutions with optimal communication, so that the problem could be solved without overloading the communication channel.

The results of this thesis were previously presented in [2, 5, 6, 7].

1.2 Justification and Motivation

The fast increase of availability of data from multiple sources and the imperious necessity of processing them using an optimal amount of resources motivates this work. This thesis focuses on the study of the well known spectral sparsification technique for graphs [8]. It pretends to contribute some theoretical results about spectral sparsification in distributed systems where not all data is available for every site. Furthermore, data may be repeated multiple times so the spectral properties of the resultant graph should be consistent with the original one. Moreover, it is also presented some theoretical results on communication complexity, and a protocol for computing the well known machine learning technique called spectral clustering [9].

1.3 Objectives

1.3.1 General Objective

Give a theoretical guarantee that spectral sparsification and its application called clustering works when applied to distributed systems where data may appear multiple times due to the several sources of information.

1.3.2 Specific Objectives

- Find a reliable representation for the Laplacian matrix of the input data graph in a distributed system. As every site will work with a different Laplacian matrix of its input data graph it is important to find a good representation of the entire input data graph in terms of those Laplacians.
- Find a good approximation of the representation of the entire input data graph with respect to the original input data graph which it is assumed has no repeated data.
- Show that the clustering technique combined with spectral sparsification works in the Number-On-Forehead communication model under the assumption of the existence of a sunflower structure in the underlying input data graph.

1.4 Book Organization

This thesis is organized as follows.

Chapter 2

We will give a brief introduction of spectral graph theory. We will focus only on the relevant definitions and results for our work. Definitions such as Laplacian matrix and spectral approximation of graphs will be given.

Chapter 3

In this chapter we will present our main result. First, we will introduce the notion of repeated data and a set-theoretic idea to represent them. Then, we will use this structure to construct a representation of the Laplacians that come from different sites. Finally, we will show how any spectral sparsification algorithm approximates the representation of Laplacians to the original one up to a constant factor.

Chapter 4

In this chapter we will give a brief introduction to the subject of communication complexity and the Number-On-Forehead (NOF) model. We will also introduce the combinatorial structure called Δ-System or *sunflower*. Then, we will present an optimal algorithm for detecting sunflowers in the NOF model with constant amount of communicated bits. Finally, we will conclude with the applicability of communication complexity in other fields.

Chapter 5

In this chapter we will focus on introducing the well known machine learning technique called *spectral clustering*. Then, we will propose a protocol on the NOF model for performing *spectral clustering* under the assumption that the entire input data graph is a sunflower.

Chapter 6

In this chapter we will give general conclusions about the work and propose some future works based on the results obtained in this thesis.

In [2] we started the study of *spectral sparsification* and in Chapter 2 we introduced the notation given there. In [5] we introduced the basis of *communication complexity* and proposed to study *spectral clustering* in a distributed model under the assumption of the existence of overlapping data. This is reflected in Chapters 4 and 5. Furthermore, in [6] we proposed a protocol for computing *spectral clustering* with overlapping data. This result is presented in Chapter 5. Finally, in [7] we gave our main results about *spectral sparsification* with distributed data. These results are presented in Chapter 3.

CHAPTER 2

SPECTRAL GRAPH THEORY AND SPECTRAL SPARSIFICATION

In this section we will introduce some standard notations and definitions in *spectral graph theory* and *spectral sparsification* as well as the algorithms used to find spectral sparsifiers of graphs. First, in Section 2.1 we will introduce some important notations and definitions from *graph theory*. In Section 2.2 we will define the *Laplacian* matrix of a given graph, some applications in different areas and its most relevant properties for this work. In Section 2.3 we will define the *spectral sparsifier* of a given graph and mention the most recent algorithms in this line of work.

2.1 Graphs

This section presents only the kind of graphs we work with, that is, *undirected graphs*.

An *undirected* graph G is an ordered pair (V, E), where V is a finite set and E consist of *unordered* pair of vertices $\{u, v\}$ with $u, v \in V$ and $u \neq v$. By convention, we use the notation (u, v) for an edge, rather than the set notation $\{u, v\}$, and we consider (u, v) and (v, u) the same edge. In simple undirected graphs, self-loops are forbidden, and so every edge consists of two distinct vertices. We often use the notation $e = (u, v)$ or $e = uv$ for denoting edges in undirected graphs. If $e = (u, v)$ is an edge of an undirected graph $G = (V, E)$ we say that e is *incident on* vertices u and v. We also say that vertex u is *adjacent* to vertex v or the vertices u and v are *neighbors*.

A *path* of *length* k from vertex u to vertex u' in a graph $G = (V, E)$ is a sequence $\langle v_0, v_1, \ldots, v_k \rangle$ of vertices such that $u = v_0$ and $u' = v_k$, and $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \ldots, k$. The length of the path is the number of edges on it. If there is a path p from u to u', we say that u' is *reachable* from u via p. In an undirected graph, a path $\langle v_0, v_1, \ldots, v_k \rangle$ forms a *cycle*

if $k \geq 3$ and $v_0 = v_k$; the cycle is simple if $v_1, \ldots, v_k$ are distinct. A graph with no cycles is *acyclic*.

An undirected graph is *connected* if every vertex is reachable from all other vertices. The *connected components* of a graph are the equivalence classes of vertices under the "is reachable from" relation. An undirected graph is connected if it has only one connected component. The edges of the connected component are those that are incident on only the vertices of the connected component.

We say that $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given that $V' \subseteq V$, the subgraph of G *induced* by V' is the graph $G' = (V', E')$ where $E' = \{(u, v) : u, v \in V'\}$.

Several kinds of graphs has special names. A *complete graph* is an undirected graph in which every pair of vertices is adjacent. A *bipartite graph* is an undirected graph $G = (V, E)$ in which V can be partitioned into two sets V_1 and V_2 such that $(u, v) \in E$ implies either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$. An acyclic, undirected graph is a *forest*, and a connected, acyclic, undirected graph is a *tree*. There two variants of graph we will talk about. A *multigraph* is like an undirected graph, but it can have both multiple edges between vertices and self-loops. A *hypergraph* is like an undirected graph, but each *hyperedge*, rather than connecting two vertices, connects an arbitrary subsets of vertices.

The *degree* of a vertex $v \in V$ is the number of neighbors it has. We will represent the degree of a vertex as a function $d : V \to \mathbb{N}$. The degree of a vertex $v \in V$ is $d(v) = \sum_{uv \in E} 1$, this is, we add 1 to $d(v)$ if u is adjacent to v. Sometimes we will use the notation d_v instead of $d(v)$. For a subset $V' \subseteq V$ we define $d(V') = \sum_{v \in V'} d(v)$.

A *weighted*, undirected graph $G = (V, E, w)$ is an undirected graph whose edges has *weights* assigned to them. The weights are defined by a function $w : E \to \mathbb{R}$. Notice that an *unweighted*, undirected graph can be seen as a weighted graph $G = (V, E, w)$ whose weight function is $w : E \to \{0, 1\}$ assigning 0 to edges $e \notin E$ and 1 to edges $e \in E$. We can also generalized the definition of degree to *weighted degree*. Let $v \in V$ a vertex of G, the weighted degree of v is $d(v) = \sum_{uv \in E} w(u, v)$. Also, for a subset $E' \subseteq E$ the weight of E' is $w(E') = \sum_{uv \in E'} w(u, v)$.

Representing graphs is important for computational systems which operates with them. There are many basic ways to represent graphs [10]: *adjacency matrix* representation, *adjacency list* representation, *edge list* representation, *incidence matrix* representation, etc. In this work we are focus on the adjacency matrix representation A_G which is defined as follows

$$A_G(u, v) = a_{uv} = \begin{cases} w(u, v) & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}, \tag{2.1}$$

where the subscript G that indicates the underlying graph G will be omitted when is clear from the context. Notice that A_G will be a binary matrix[1] if G is an unweighted graph; otherwise the

[1] A binary matrix is a matrix whose entries are elements of $\{0, 1\}$.

matrix is called *weighted adjacency matrix*. We can also define the *degree matrix* and *weighted degree matrix* D_G as well

$$D_G(u,v) = d_{uv} = \begin{cases} d(u) & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}. \tag{2.2}$$

Lastly, we will introduce a notion that it will be used throughout the entire work. A *cut* in a graph $G = (V, E)$ is a subset of edges $E' \subset E$ such that the subgraph $G = (V, E - E')$ has exactly two *connected components*. Each connected component has its own subset of vertices and they constitute a partition of V. The size of a cut E' is given by $w(E')$ and denoted by $cut_G(V', \overline{V'})$ or simply $cut_G(V')$ with $V' \subseteq V$.

2.2 Spectral Graph Theory

Spectral graph theory studies the properties of a graph from the linear algebra perspective. It studies matrices associated to a given graph and their eigenvalues and eigenvectors. The Laplacian matrix of a graph G is the main subject of study in this chapter. It is usually represented by L_G, when the graph G is clear from context the subscript is dropped. Laplacian matrices are symmetric, have zero row-sums, and have non-positive off-diagonal entries. Some applications of the Laplacian matrix found in [11] are *regression on graphs*, *spectral graph theory*, *solving maximum flow by interior point algorithms*, *resistor networks* and *partial differential equations*. Laplacian matrices could be studied over weighted or unweighted, undirected and simple graphs. Some results on the study of unweighted Laplacian matrices given in [12] are lower bounds on the diameter of a graph, number of spanning trees and number of connected components. In this work we will focus on weighted graphs rather than unweighted. Now, we will give the definition of Laplacian matrices.

Definition 2.1. Given a undirected weighted graph $G = (V, E, w)$ with $n = |V|$, $m = |E| \leq \binom{n}{2}$ and $w : E \to \mathbb{R}_{\geq 0}$. Let $A_G \in \mathbb{R}^{n \times n}$ be the *weighted adjacency matrix* and $D_G \in \mathbb{R}^{n \times n}$ the *weighted degree matrix* of G defined as in (2.1) and (2.2) respectively. The *Laplacian matrix* of a given graph G is defined as

$$L_G = D_G - A_G. \tag{2.3}$$

An example of a Laplacian matrix is shown in Figure 2.1.

Another way to look at the Laplacian is through its quadratic form B.6. Given a vector $x \in \mathbb{R}^n$, the Laplacian quadratic form of G is

$$x^T L_G x = \sum_{(u,v) \in E} w_{uv}(x_u - x_v)^2. \tag{2.4}$$

Operational researchers and computer scientists are often interested in cutting, partitioning

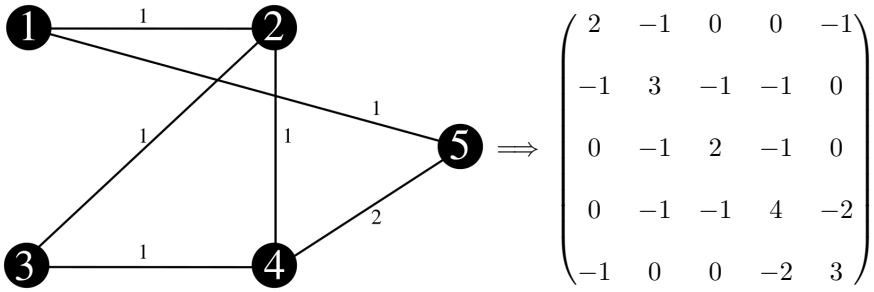


Figure 2.1: The graph at the left side has 5 vertices, every pair of vertices are connected by an edge of weight 1 except the edge (4,5) with weight 2. The rows and columns of the Laplacian matrix on the right side are indexed by the vertices of the graph. The diagonal contains the weighted degree of every vertex, for example the element at the 4-th row and column is 4. The non-diagonal elements denote the weight between every pair of vertices, for example the element at the row four and column five is -2. If two pair of vertices does not share an edge then, its corresponding element in the Laplacian matrix is equal to zero.

and clustering graphs. For those purposes the Laplacian quadratic form offers a nice characterization. Let $S \subseteq V$ be a subset of vertices, the of edges crossing S to $V - S$ is known as the boundary of S and denoted by $\partial(S)$—note that the boundary of S is equivalent to a *cut* that partitions V into S and $V - S$. Let $\chi_S \in \{0,1\}^{|V|}$ be a vector with its entries indexed by the vertices in V such that

$$\chi_S(v) = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

We call χ_S the characteristic vector of S. Then $\chi_S^T L_G \chi_S$ is equal to the sum of the weighted edges belong to $\partial(S)$. Most common applications are often interested in minimizing the sum of weighted edges in the boundary of S divided by the size of S. Thus, when we deal with unweighted graphs we use the *isoperimetric number* defined as

$$\iota(S) = \frac{|\partial(S)|}{\min(|S|, |V - S|)}. \tag{2.6}$$

On the other hand, if we work with weighted graphs then we must use the *conductance* of S defined as

$$\phi(S) = \frac{w(\partial(S))}{\min(d(S), d(V - S))}, \tag{2.7}$$

where w and d are the generalizations of weight and degree functions. The isoperimetric number and the conductance of G are defined as the minima of these quantities over all subsets of vertices

$$\iota_G = \min_{S \subset V} \iota(S) \text{ and } \phi_G = \min_{S \subset V} \phi(S). \tag{2.8}$$

When the set S that minimizes ι_G or ϕ_G is found then it is common to cluster the graph G into induced subgraphs over S and $V - S$. Later we will introduce a generalization of ϕ_G for finding a partition of size k over the vertex set of G and we will introduce an algorithm called *spectral clustering* for computing such partition. Now, we will introduce the normalized version of the Laplacian.

Definition 2.2. The *normalized symmetric Laplacian* is defined as

$$\begin{aligned} L_G^{sym} &= D_G^{-\frac{1}{2}} L_G D_G^{-\frac{1}{2}} \\ &= I - D_G^{-\frac{1}{2}} A D_G^{-\frac{1}{2}}, \end{aligned}$$

and the *random walk normalized Laplacian* as

$$L_G^{rw} = D_G^{-1} L_G$$

Notice that L_G^{sym} multiplies every i, j-th element of L_G by $\frac{1}{\sqrt{d_i}\sqrt{d_j}}$. On the other side, L_G^{rw} multiplies every i-th row of L_G by $\frac{1}{d_i}$.

In the following proposition we will summarize the properties of L_G and L_G^{sym}. For a proof we refer the reader to [9].

Proposition 2.1. The Laplacian and normalized Laplacian have the following properties:

1. L_G and L_G^{sym} are both symmetric,
2. L_G and L_G^{sym} are both *positive semidefinite*,
3. The algebraic multiplicity of the smallest eigenvalue of L_G is equal to the number of connected components of G
4. The second smallest eigenvalue approximates a sparse cut in G.

The three matrices are used for spectral partitioning of graph algorithms. We will study the applications of L_G and L_G^{sym} later. Information of how matrix L_G^{rw} is used for graph segmentation can be found in [13].

We will now introduce the theory and algorithms for approximating an arbitrary graph by a sparse one. First let us recall that sparse graphs are graphs with a number of edges less than $\binom{n}{2}$. In fact, the literature consider that sparse graphs are families of graphs with number of edges bounded by some constant or some logarithmic function on its number of vertices [14, 15]. There are other definitions of graph sparsity for general graph structures. In [16] and [17] the authors defined a family of graphs being sparse if every subset of $n' \leq n$ vertices spans subgraphs of number of edges linear on n'. In this thesis, we are interested in sparse graphs whose number of edges are bounded by a linear function over the number of vertices.

Spectral sparsification was first introduced in [18] to solve symmetric diagonal-dominant (SDD) linear systems. Then, in [8] a new notion of graph sparsification was introduced and it was based on spectral similarity of the Laplacian. They proved that every graph has a spectral sparsifier of quasi-linear size and presented an algorithm to construct it in time $O(m \log^c m)$, where m is the number of edges in the original graph c is a positive constant. Furthermore, in [19] an elementary deterministic algorithm was given for constructing spectral sparsifiers of size $O(n/\epsilon^2)$ in $O(mn^3/\epsilon^2)$ time. These algorithms use a greedy strategy to iteratively update a matrix A by a rank-one matrix $tv^T v$ that depends on the Laplacian L and a constant t. Also in [20] an algorithm that improves the spectral sparsifier size given in [8] was presented. The size of the *spectral sparsifier* was $O(n \log n/\epsilon^2)$. This latter algorithm was based on random sampling of the edges with a probability distribution based on their effective resistance. Such probabilities are proportionally inverse to the edge weights. In [14] was summarized the results of some algorithms presented in [8, 19, 20] and the notions of spectral sparsifiers which we introduce now.

The approximation between graphs depend on a similarity notion. There are a vast number of notions of similarity among graphs, we will explore some of them and then introduce the one that we will work with. First, when we measure similarity between graphs we will assume that they have the same number of vertices n. All these similarity notions were previously mentioned in [14].

The *cut similarity* establishes that two graphs are similar if the size of their cuts are approximately the same. We say that two graphs $G = (V, E, w)$ and $\tilde{G} = (V, E, \tilde{w})$ are σ-cut similar if

$$cut_{\tilde{G}}(S)/\sigma \leq cut_G(S) \leq \sigma cut_{\tilde{G}}(S) \tag{2.9}$$

for all $S \subset V$. Moreover, every graph is *cut*-similar to a graph with average degree $O(\log n)$ and it could be computed in polylogarithmic time. The notion of cut similarity was introduced in attempts to develop faster algorithms for the minimum cut and maximum flow problems.

The *distance similarity* relies on the assignation of *lengths* to the edges. This induce a *shortest path* between every pair of vertices, this is, $dist_G(u, v)$ for all $u, v \in V$. We say that two different graphs are σ-distance similar if every pair or vertices has approximately the same shortest path, this is

$$dist_G(u, v)/\sigma \leq dist_{\tilde{G}}(u, v) \leq \sigma dist_G(u, v) \tag{2.10}$$

for all $u, v \in V$.

The *spectral similarity* of graphs is defined using their quadratic form. Recall from Equation B.6 we have that $Q_G(x) = x^T L_G x$ for all $x \in \mathbb{R}^n$. Then, two graphs G and $\tilde{G}$ are σ-spectrally

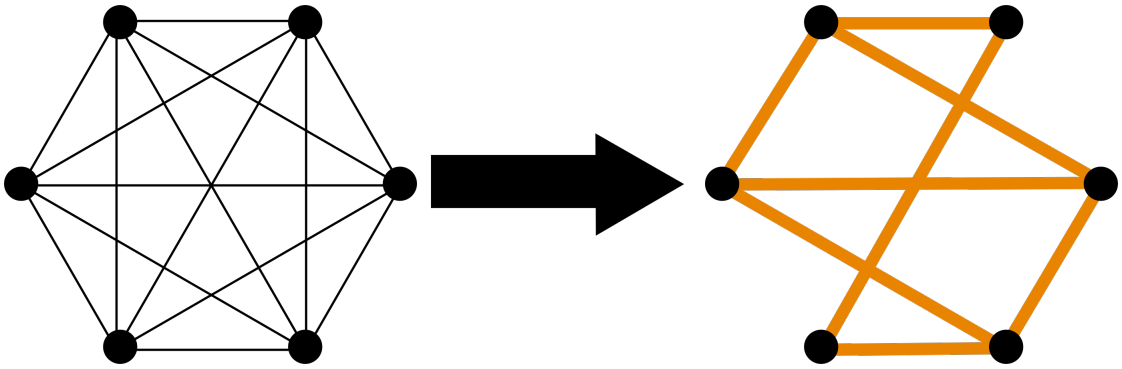


Figure 2.2: An illustration of how spectral sparsification works

similar if

$$Q_{\tilde{G}}(x)/\sigma \leq Q_G(x) \leq \sigma Q_{\tilde{G}}(x) \tag{2.11}$$

holds for all x. Furthermore, if χ_S is the characteristic vector of $S \subset V$ then $Q_G(\chi_S)$ gives a quantity related to a cut in G. This implies that cut similarity is a special case of spectral similarity [14]. Now, we will introduce a similarity notion related to this one, the *matrix similarity*.

First, for two symmetric matrices A and B in $\mathbb{R}^{n\times n}$, we write $A \preccurlyeq B$ to indicate $0 \leq x^T Bx - x^T Ax$ for all $x \in \mathbb{R}^n$, which in turns implies that $B - A$ is positive semidefinite. Next, we say that A and B are σ-spectrally similar if

$$B/\sigma \preccurlyeq A \preccurlyeq \sigma B. \tag{2.12}$$

This relation is named spectral similarity because it implies that A and B have similar eigenvalues. Recall from the *minimax principle* B.4 that $\lambda_i(A) = \min_{S:dim(S)=i} \max_{x\in S;x\neq 0} \frac{x^T Ax}{x^T x}$. Thus, if $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A and $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n$ are the eigenvalues of B then, for all i $\lambda_i/\sigma \leq \tilde{\lambda}_i \leq \sigma\lambda_i$. Using this notation we can write that

$$L_{\tilde{G}}/\sigma \preccurlyeq L_G \preccurlyeq \sigma L_{\tilde{G}}. \tag{2.13}$$

That is, two graphs are σ-spectrally similar if their Laplacian matrices are σ-spectrally similar. In general, [14] established that a good spectral sparsifier $\tilde{G}$ of G should have the following three properties: I) $\tilde{G}$ is σ-spectrally similar to G, II) Edges of $\tilde{G}$ consist on reweighted edges of G and III) $\tilde{G}$ has at most $d|V|$ edges where d is a logarithmic function in n. An illustration of a spectral sparsifier can be observed in Figure 2.2.

2.3 Spectral Sparsification

As we observed, spectral sparsifier algorithms rely on two major characteristics to be considered as optimal algorithms. This characteristics are the size of the spectral sparsifier and the time required to compute it. To the best of our knowledge the best spectral sparsification algorithm is presented in [21]. This algorithm relies on two important characteristic from the former algorithms, a random sampling procedure based on effective resistance and rank-one update based on barrier functions. The algorithm outputs a spectral sparsifier of $O(\frac{qn}{\epsilon^2})$ edges in $\tilde{O}(\frac{qmn^{5/q}}{\epsilon^{4+4/q}})$ time. In this work we are more interested in the size of the spectral sparsifiers rather than the time to compute it as we will see later. We should mention as well that spectral sparsifiers have been studied in other computational models as *streaming models* [22]. In the streaming model the goal is to compute the spectral sparsifier of a graph with low memory requirements. That is, the memory use to store partial results of the algorithm is low. The input data is received in a sort of array that should be processed. Results of spectral sparsifiers in the streaming model are given in [22].

Finally, from now on we will use a notation of spectral sparsification different from the one presented at Equation 2.13. For graphs G and H we will say that H is a spectral sparsifier of G if

$$(1-\epsilon)L_G \preccurlyeq L_H \preccurlyeq (1+\epsilon)L_G, \tag{2.14}$$

where $0 < \epsilon \leq 1$ is the approximation factor. Notice that Equation 2.13 could be inferred from Equation 2.14.

In the following chapter we will present a general approach to compute L_H from Equation 2.14 when the graph G is expressed as the union of different subgraphs G_i. An application of this known result will yield a distributed algorithm for computing H when each G_i are in different physical places.

CHAPTER 3

OVERLAPPING CARDINALITY PARTITION

In this chapter we will introduce a mathematical object that could be used to represent overlapping information in computational systems. We use this structure to show that the spectral sparsification technique introduced in the last chapter works well under distributed systems with overlapping information. The main contribution of this chapter, presented in Theorem 3.1, is an estimation of the approximation factor and an explicit calculation of the edge weights in the union of the spectral sparsifiers of graphs G_i allocated in a distributed system. The chapter is divided in three sections. Section 3.1 will introduce a mathematical object to express how many times an item of a set is repeated in a family of subsets. In Section 3.2 we will use the idea of repeated data to show that the Laplacian of a given graph can be expressed as a linear combination of Laplacians that depends on those repeated data. In Section 3.3 we will show that the union of spectral sparsifiers of given graphs is a spectral sparsifier of the union of such graphs.

3.1 Overlapping Cardinality Partition

Duplicated data is often found in database systems because of redundancy policies, registers that have been saved more than once or overlapping source of information. As we work with distributed systems, this could lead to data that is saved in more than one database system. Furthermore, as we work with input data graphs, repeated data could lead to repeated edges or repeated vertices. In this section we will characterize duplicated data in distributed systems as a family of sets over $\mathbb{N}$ as is observed in Figure 3.1. Furthermore, every set in the family will represent data from a unique source of information, and, different sets in the family will represent data from a different source of information. Then, we will construct a more suitable

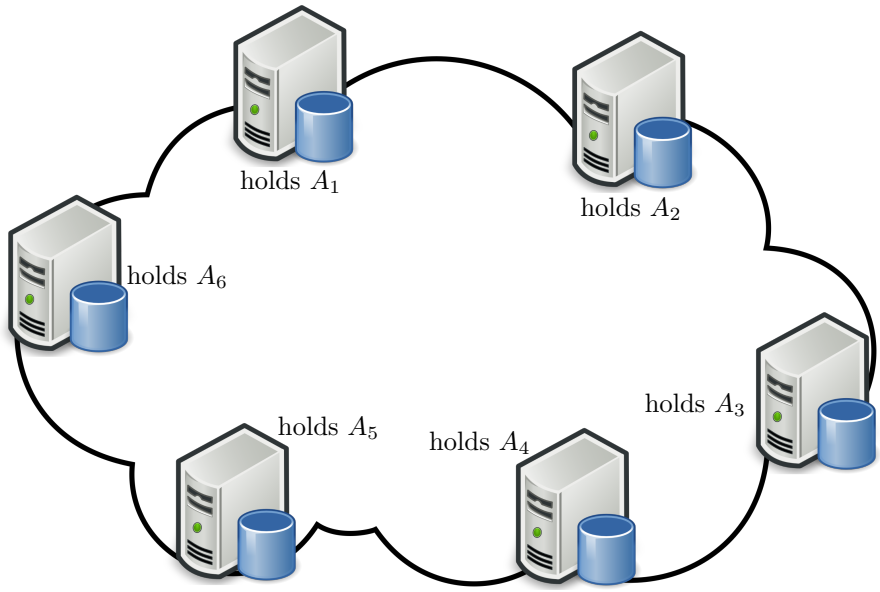


Figure 3.1: This distributed system has its data distributed along six databases or sites. Every site holds the dataset A_i which consist of a set of points in $\mathbb{N}$. These points could represent vertices or edges of the input graph. The entire family of sets is represented as $\{A_1, A_2, A_3, A_4, A_5, A_6\}$.

representation of data in terms of the Laplacian of a graph introduced in the last chapter.

The family of sets observed in Figure 3.1 will be our main subject of study. We will have typically that $A_i \subseteq [m]$[1] and $\bigcup_{i=1}^{t} A_i = [m]$ where t denotes the number of sites. Given a point $a \in [m]$, we want to know how many subsets of $\{A_i\}_{i \leq t}$ contains a so we define the *occurrence number*.

Definition 3.1 (Occurrence Number)**.** Let $\mathcal{A} = \{A_1, \ldots, A_t\}$ be a family of subsets of $[m]$. For any $a \in [m]$, the occurrence number of a in $\mathcal{A}$, denoted $\#(a)$, is the maximum number of sets from $\mathcal{A}$ in which a appears.

The *occurrence number* captures how many times a given element appears in different subsets. Example 3.1 shows how Definition 3.1 works.

Example 3.1. Let $m = 11$ and $\mathcal{A} = \{\{1, 4, 5\}, \{1, 2, 3, 5, 6, 7, 8\}, \{3, 7, 8, 9\}, \{4, 5, 6, 10\}, \{7, 8, 9, 11\}\}$. Here we have that $\#(1) = 2$, $\#(2) = 1$, $\#(3) = 2$, and so on. □

Now, we may want to count how many elements appears an equal number of times in different subsets. This could be carried out by the *overlapping cardinality* of a subset which we define as follows.

Definition 3.2 (Overlapping Cardinality)**.** Let $\mathcal{A} = \{A_1, \ldots, A_t\}$ be a family of subsets of $[m]$ for some fixed m and $A = \bigcup_{i=1}^{t} A_i$. The *overlapping cardinality* of a subset $A' \subseteq A$ in $\mathcal{A}$ is a positive integer c such that for each $a \in A'$ its occurrence number $\#(a) = c$; otherwise the overlapping cardinality of A' in $\mathcal{A}$ is 0.

[1] See Section A.1

The overlapping cardinality identifies the maximum number of times the elements of a subset appears in a family of subsets. Also notice that if all subsets of $\mathcal{A}$ have overlapping cardinality equal to 1 then, $\mathcal{A}$ is a partition of $[m]$. In the following example it is shown how to compute the overlapping cardinality of a subset.

Example 3.2. Let $m = 11$ and $\mathcal{A}$ be as in Example 3.1. Here we have that $A = \bigcup_{i=1}^{t} A_i = [m]$. Now consider the sets $\{5, 8\}$ and $\{1, 2, 3\}$.

- The overlapping cardinality of $\{5, 8\}$ in $\mathcal{A}$ is 3, because $\#(5) = \#(8) = 3$.
- The overlapping cardinality of $\{1, 2, 3\}$ in $\mathcal{A}$ is 0 because the occurrence number of one of the elements of the set is different from the others, namely, $\#(1) = \#(3) = 2$, and $\#(2) = 1$. □

In the following definition we use the idea of overlapping cardinality to construct a partition on the set $\mathcal{A}$ of subsets of $[m]$.

Definition 3.3 (Overlapping Cardinality Partition)**.** Given a family $\mathcal{A}$ as in Definition 3.2, an *overlapping cardinality partition* over A on $\mathcal{A}$ is a partition $\{A'_1, \ldots, A'_k\}$ of A where each A'_i has overlapping cardinality c_i on $\mathcal{A}$. We call the sequence $(c_1, c_2, \ldots, c_k)$, with $1 \leq c_1 < c_2 < \cdots < c_k$, the *overlapping cardinalities* over the family $\mathcal{A}$.

From Definition 3.3 it is easy to observe that subsets with different overlapping cardinalities are different. Now, observe the following fact

Fact 3.1. Let X and Y be two subsets of A with the same overlapping cardinality and $\{A'_1, \ldots, A'_k\}$ an overlapping cardinality partition over A. Then, $X \bigcup Y$ is a subset of some A'_i.

From Fact 3.1 is easy to notice that an overlapping cardinality partition is not unique. Now, observe an example of overlapping cardinality partition.

Example 3.3. Take $\mathcal{A}$ from Examples 3.1 and 3.2. An overlapping cardinality partition is

$$\{\{2, 10, 11\}, \{1, 3, 4, 6, 7, 9\}, \{5, 8\}\}.$$

Here, $\{2, 10, 11\}$ has overlapping cardinality equal to 1 because $\#(2) = \#(10) = \#(11) = 1$. The subset $\{1, 3, 4, 6, 7, 9\}$ has overlapping cardinality equal to 2 because $\#(1) = \#(3) = \#(4) = \#(6) = \#(7) = \#(9) = 2$. Finally, in Example 3.2 we saw that the subset $\{5, 8\}$ has overlapping cardinality 3. □

A pictorial representation of Example 3.3 is shown in Figure 3.2.

In the next section we will use the overlapping cardinality partition to decompose a Laplacian matrix over an input data allocated along different sites as a linear combination of Laplacians.

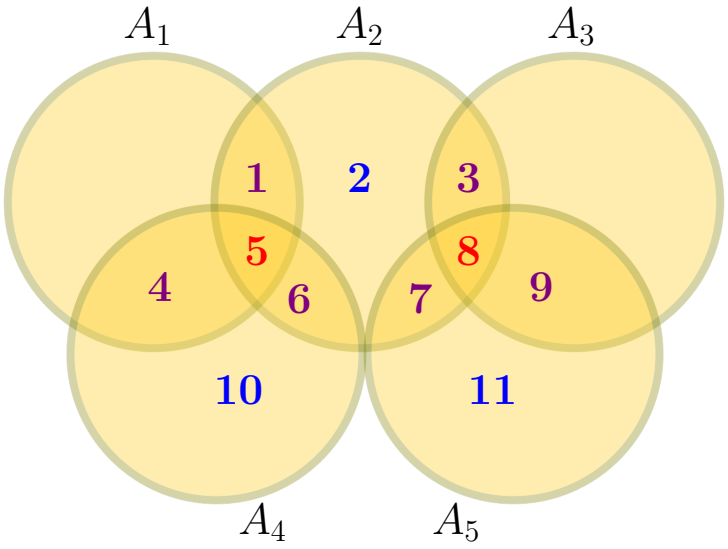


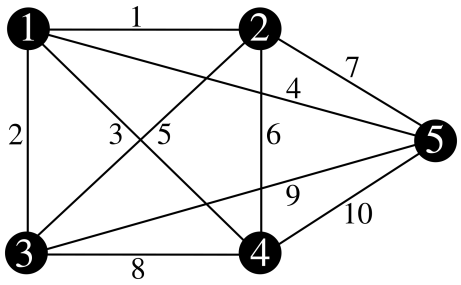
Figure 3.2: From Example 3.3, the elements of the partition $\{\{2, 10, 11\}, \{1, 3, 4, 6, 7, 9\}, \{5, 8\}\}$ belongs to 1, 2 and 3 sets of the family and are colored in blue, violet and red respectively.

3.2 Input Data Graph and Laplacian Decomposition

We begin with an idea of how a subset of natural numbers could be represented as a graph. Let us take a family $\mathcal{A}$ of subsets of $[m]$ as in Example 3.1. For every $A_i \in \mathcal{A}$, the elements $e \in A_i$ will represent the edges of our graph. As every edge is represented as a pair of vertices, usually (i, j) then, the element e in the i-th row and j-th column of any $n \times n$ matrix could represent the edge (i, j). We will use an *enumerating process*. Given n and e, an $O(n^2)$ algorithm can find the corresponding edge (i, j) in G. Just take e, initialize a counter in 1 and go along through the columns and rows of the upper triangular part of the given adjacency matrix increasing the counter in one unit until its equal to e. The i-th row and j-th column at which the counter is equal to e is the desired edge (i, j). Also, as the graph is undirected, the (j, i) element should be equal to e as well in the given matrix. Suppose the sites only know the value of m and that G is a dense graph. Then, they can compute the number of vertices by finding the maximum n such that $\binom{n-1}{2} < m$. The enumerating process is observed in Figure 3.3. Let us consider $E \subset \mathbb{N}$ and its corresponding graph $G = (V, E)$ constructed by an enumerating process. We will call E the *enumerating edges* of G. We should remark that this enumerating process is just to keep track of the edges that every site is aware of and to make notation more comfortable. That is, instead of representing an edge and its weight in a given graph as a triplet i, j, w_{ij} we will represent it as w_{ij}^e.

Now, we will continue with a lemma that shows that the Laplacian of an input graph can be rewritten as a linear combination of Laplacians. These Laplacians correspond to induced subgraphs constructed from an overlapping cardinality partition of the family of edge-set.

Let $G = (V, E, w)$ be an undirected and weighted graph with a weight function $w : E \to \mathbb{R}_{\geq 0}$, let $\mathcal{E} = \{E_1, \ldots, E_t\}$ be a collection of subsets of E such that $\bigcup_{i=1}^{t} E_i = E$ where $E_i \neq \emptyset$ and $G_i = (V, E_i, w_i)$ is an induced subgraph of G where $w_i : E_i \to \mathbb{R}_{\geq 0}$ and $w_i(e) = w(e)$ for all



$$\Longrightarrow \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 5 & 6 & 7 \\ 2 & 5 & 0 & 8 & 9 \\ 3 & 6 & 8 & 0 & 10 \\ 4 & 7 & 9 & 10 & 0 \end{pmatrix}$$

Figure 3.3: In this example we have a complete graph with 5 vertices. We will use the enumerating process to represent a set of points in $\mathbb{N}$ as an undirected graph with no loops which is exactly what the Laplacians usually represent.

$e \in E_i$ and 0 otherwise.

Lemma 3.1. If $1 \leq c_1 < c_2 < \cdots < c_k$ are the overlapping cardinalities over the family $\mathcal{E}$ with an overlapping cardinality partition $\{E'_{c_j}\}_{j\leq k}$, then $\sum_{i=1}^{t} L_{G_i} = \sum_{j=1}^{k} c_j L_{G'_{c_j}}$ where $L_{G'_{c_j}}$ is the Laplacian of $G'_{c_j} = (V, E'_{c_j}, w'_{c_j})$.

Proof. First notice that, for all $e = xy \in E'_{c_j}$ there exists a subfamily of $\mathcal{E}$ with cardinality equal to c_j such that e belongs to every member of it and of its associated subgraph.

Take any $xy \in E'_{c_j}$ for some $j \in \{1, \ldots, k\}$. There exists c_j induced subgraphs $G_{i_1}, \ldots, G_{i_{c_j}}$ of G that have xy as an edge, and all other induced subgraphs $G_{k_1}, \ldots, G_{k_\ell}$ do not have xy as and edge, where $c_j + \ell = t$. This means that

$$\sum_{i=1}^{t} L_{G_i}(x, y) = c_j \cdot L_{G'_{c_j}}(x, y) = -c_j \cdot w(x, y). \tag{3.1}$$

Now, let $d_G(x)$ denote the degree of x in G. We know that $d_G(x) = \sum_y w(x, y)$ where $xy \in E$. Since $\{E'_{c_j}\}_{j\leq k}$ is a partition of E, we can rewrite the degree of x as

$$d_G(x) = \sum_{xy_{c_1} \in E'_{c_1}} w(x, y_{c_1}) + \cdots + \sum_{xy_{c_k} \in E'_{c_k}} w(x, y_{c_k}). \tag{3.2}$$

Then, the degree of x in the graph G'_{c_j} is

$$L_{G'_{c_j}}(x, x) = \sum_{xy_{c_j} \in E'_{c_j}} w(x, y_{c_j}) = d_{G'_{c_j}}(x). \tag{3.3}$$

If we take an edge $xy_{c_j} \in E'_{c_j}$, where x is fixed, we know that xy_{c_j} appears only in the induced subgraphs $G_{i_1}, \ldots, G_{i_{c_j}}$, and hence, we obtain

$$\sum_{i=1}^{t} \left(\sum_{xy_{c_j} \in E'_{c_j}} w_i(x, y_{c_j}) \right) = c_j \cdot d_{G'_{c_j}}(x). \tag{3.4}$$

If we take another edge $uv \in E'_{c_m}$, with $m \neq j$, note that uv does not belong to any of

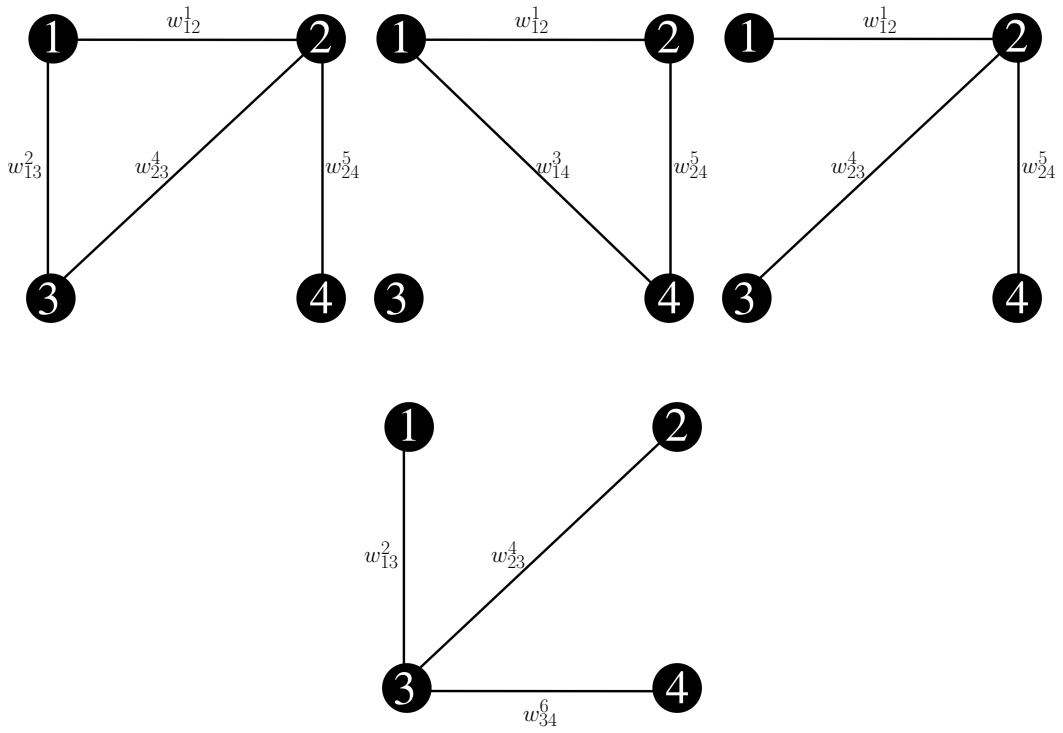


Figure 3.4: Subgraphs of K_4 as stated in Example 3.4. The union of them is K_4. Lemma 3.1 claims that the sum of the Laplacians of these subgraphs could be rewritten as the sum of Laplacians of subgraphs induced by the overlapping cardinality partition associated to their family of edges.

the graphs $G_{i_1}, \ldots, G_{i_{c_j}}$ and each Laplacian matrix $L_{G_{i_1}}, \ldots, L_{G_{i_{c_j}}}$ has 0 in its (u, v)-entry. Therefore, adding uv to Eq.(3.1) we have that

$$\sum_{i=1}^{t} (L_{G_i}(x, y) + L_{G_i}(u, v)) = c_j \cdot L_{G'_{c_j}}(x, y) + c_m \cdot L_{G'_{c_m}}(u, v).$$

Extending this argument to all equivalent classes in $\{E'_{c_j}\}_{j \leq k}$, for each non-diagonal entry (x, y), with $xy \in E$, it holds

$$\sum_{i=1}^{t} L_{G_i}(x, y) = \sum_{j=1}^{k} c_j \cdot L_{G'_{c_j}}(x, y). \tag{3.5}$$

A similar argument can be made for the diagonal entries with Eq.(3.4), thus obtaining

$$\sum_{i=1}^{t} \left(\sum_{xy_{c_1} \in E'_{c_1}} w_i(x, y_{c_1}) + \cdots + \sum_{xy_{c_k} \in E'_{c_k}} w_i(x, y_{c_k}) \right) = \sum_{i=1}^{t} L_{G_i}(x, x) = \sum_{j=1}^{k} c_j \cdot L_{G'_{c_j}}(x, x). \tag{3.6}$$

Equations (3.5) and (3.6) imply the lemma. □

In the following example we show how Lemma 3.1 works

Example 3.4. Let G be a the complete, undirected and weighted graph K_4 with its edges distributed along four different sites. The family $\mathcal{E} = \{\{1,2,4,5\},\{1,3,5\},\{1,4,5\},\{2,4,6\}\}$ represents the enumerating edges of G. Furthermore, $\mathcal{E}' = \{\{1,4,5\},\{3,6\},\{2\}\}$ is the overlapping cardinality partition of $\mathcal{E}$ and $3, 1$ and 2 are the overlapping cardinalites of the sets in $\mathcal{E}'$ respectively. We can induce from $\mathcal{E}$ every subgraph of G as is observed in Figure 3.4.

Then, when computing the sum of their Laplacians $L_{G_1} + L_{G_2} + L_{G_3} + L_{G_4}$ we get

$$\begin{pmatrix} d_1^1 & -w_{12}^1 & -w_{13}^2 & 0 \\ -w_{12}^1 & d_2^1 & -w_{23}^4 & -w_{24}^5 \\ -w_{13}^2 & -w_{23}^4 & d_3^1 & 0 \\ 0 & -w_{24}^5 & 0 & d_4^1 \end{pmatrix} + \begin{pmatrix} d_1^2 & -w_{12}^1 & 0 & -w_{14}^3 \\ -w_{12}^1 & d_2^2 & 0 & -w_{24}^5 \\ 0 & 0 & d_3^2 & 0 \\ -w_{14}^3 & -w_{24}^5 & 0 & d_4^2 \end{pmatrix} + \begin{pmatrix} d_1^3 & -w_{12}^1 & 0 & 0 \\ -w_{12}^1 & d_2^3 & -w_{23}^4 & -w_{24}^5 \\ 0 & -w_{23}^4 & d_3^3 & 0 \\ 0 & -w_{24}^5 & 0 & d_4^3 \end{pmatrix} + \begin{pmatrix} d_1^4 & 0 & -w_{13}^2 & 0 \\ 0 & d_2^4 & -w_{23}^4 & 0 \\ -w_{13}^2 & -w_{23}^4 & d_3^4 & -w_{34}^6 \\ 0 & 0 & -w_{34}^6 & d_4^4 \end{pmatrix} \tag{3.7}$$

where the superscript for every diagonal element indicates the site. Notice that the degree of every vertex is not the same in all sites but their sums are over the same set of possible neighbors. As an example observe d_1^1 and d_1^2. The first is the sum of w_{12}^1 and w_{13}^2 and the second is the sum of w_{12}^1 and w_{14}^3. Now, the Sum 3.7 is equivalent to

$$\begin{pmatrix} d_1 & -3w_{12}^1 & -2w_{13}^2 & -w_{14}^3 \\ -3w_{12}^1 & d_2 & -3w_{23}^4 & -3w_{24}^5 \\ -2w_{13}^2 & -3w_{23}^4 & d_3 & -w_{34}^6 \\ -w_{14}^3 & -3w_{24}^5 & -w_{34}^6 & d_4 \end{pmatrix}, \tag{3.8}$$

where

$$\begin{aligned} d_1 &= d_1^1 + d_1^2 + d_1^3 + d_1^4 \\ &= 3w_{12}^1 + 2w_{13}^2 + w_{14}^3, \\ d_2 &= d_2^1 + d_2^2 + d_2^3 + d_2^4 \\ &= 3w_{12}^1 + 3w_{23}^4 + 3w_{24}^5, \\ d_3 &= d_3^1 + d_3^2 + d_3^3 + d_3^4 \\ &= 2w_{13}^2 + 3w_{23}^4 + w_{34}^6, \\ d_4 &= d_4^1 + d_4^2 + d_4^3 + d_4^4 \\ &= w_{14}^3 + 3w_{24}^5 + w_{34}^6. \end{aligned} \tag{3.9}$$

Then, it is easy to see that the matrix in (3.8) could be expressed as the following sum

$$\begin{pmatrix} d_1^{1'} & -3w_{12}^1 & 0 & 0 \\ -3w_{12}^1 & d_2^{1'} & -3w_{23}^4 & -3w_{24}^5 \\ 0 & -3w_{23}^4 & d_3^{1'} & 0 \\ 0 & -3w_{24}^5 & 0 & d_4^{1'} \end{pmatrix} + \begin{pmatrix} d_1^{2'} & 0 & -2w_{13}^2 & 0 \\ 0 & d_2^{2'} & 0 & 0 \\ -2w_{13}^2 & 0 & d_3^{2'} & 0 \\ 0 & 0 & 0 & d_4^{2'} \end{pmatrix} + \begin{pmatrix} d_1^{3'} & 0 & 0 & -w_{14}^3 \\ 0 & d_2^{3'} & 0 & 0 \\ 0 & 0 & d_3^{3'} & -w_{34}^6 \\ -w_{14}^3 & 0 & -w_{34}^6 & d_4^{3'} \end{pmatrix}. \tag{3.10}$$

Notice that every matrix correspond to a Laplacian of a subgraph induced from $E_i' \in \mathcal{E}'$ multiplied by a natural number. This number is the overlapping cardinality of E_i'. Finally, we get

$$3L_{G_1'} + 2L_{G_2'} + L_{G_3'} \tag{3.11}$$

as Lemma 3.1 claimed.

In the following section we will use Lemma 3.1 to show how the spectral sparsification works when the edges of a graph G are distributed as in Example 3.4.

3.3 Constructing Spectral Sparsifiers from Induced Subgraphs

Here we present our main contribution, we will use Lemma 3.1 to show that the spectral sparsifier of $\sum_j^k c_j L_{G'_{c_j}}$ is a spectral sparsifier of the Laplacian L_G of an input graph G. We assume that the edges of G are distributed among different sites and every site is aware of a subgraph of G. Furthermore, every site i can compute a spectral sparsification of G_i. As a final step every site will send its sparsified graph G_i' through some communication channel. Then, the union of the spectral sparsifiers will be denoted computationally by the sum of their Laplacians. Our contribution is stated in the following theorem.

Theorem 3.1. Let $(1 = c_1 < c_2 < \cdots < c_k)$ be the overlapping cardinalities over the family $\mathcal{E}$ with $\{E'_{c_j}\}_{j\leq k}$ its associated overlapping cardinality partition and $L_{G_1}, \ldots, L_{G_t}$ be the Laplacians of $G_1, \ldots, G_t$. If $H_i = (V, D_i, h_i)$ is an ϵ-spectral sparsifier of G_i, then $H = (V, \bigcup_i^t D_i, h)$ is an ϵ'-spectral sparsifier of G where $h(e) = \frac{\sum_i^t h_i(e)}{c_1 c_k}$ and $\epsilon' \geq 1 - \frac{1-\epsilon}{c_k}$.

Proof. Let L_{H_i} be the Laplacian of H_i. By hypothesis we have that for every $i \in [t]$ and $x \in \mathbb{R}^V$

$$(1-\epsilon)x^T L_{G_i} x \leq x^T L_{H_i} x \leq (1+\epsilon)x^T L_{G_i} x.$$

Then we may take the summation over all $i \in [t]$ to get

$$(1-\epsilon)\sum_{i}^{t} x^T L_{G_i} x \leq \sum_{i}^{t} x^T L_{H_i} x \leq (1+\epsilon)\sum_{i}^{t} x^T L_{G_i} x. \tag{3.12}$$

Now, lets consider the left hand side of the Equation (3.12). Using Lemma 3.1 we get

$$\begin{aligned}(1-\epsilon)\sum_{i=1}^{t} x^T L_{G_i} x &= (1-\epsilon)\sum_{i=1}^{k} c_i \cdot x^T L_{G'_{c_i}} x \\ &\geq (1-\epsilon)c_1 \sum_{i=1}^{k} x^T L_{G'_{c_i}} x \\ &= (1-\epsilon)c_1 x^T L_G x,\end{aligned}$$

where the last equality follows from the fact that $\{E'_{c_j}\}_{j\leq k}$ is a partition of E. Similarly for the right hand side of Equation (3.12) we have that

$$(1+\epsilon)\sum_{i}^{t} x^T L_{G_i} x \leq (1+\epsilon)c_k x^T L_G x. \tag{3.13}$$

Therefore, by multiplying equations (3.13) and (3.13) by $\frac{1}{c_1 c_k}$ we obtain

$$(1-\epsilon)\frac{x^T L_G x}{c_k} \leq x^T L_H x \leq (1+\epsilon)\frac{x^T L_G x}{c_1},$$

where $x^T L_H x = (\sum_{i=1}^{t} x^T L_{H_i} x)/(c_1 c_k)$.

To finish the proof, note that we want $1-\epsilon' \leq (1-\epsilon)/c_k$ and $(1+\epsilon)/c_1 \leq 1+\epsilon'$ with $\epsilon \leq \epsilon' < 1$. In order to solve this, we choose an $\epsilon' \geq 1 - \frac{1-\epsilon}{c_k}$. First notice that $1-\epsilon' \leq 1 - 1 + \frac{1-\epsilon}{c_k} = \frac{1-\epsilon}{c_k}$. Then we have that $\frac{1+\epsilon}{c_1} \leq \frac{1+\epsilon'}{c_1} = 1+\epsilon'$. □

An example of what Theorem 3.1 states is shown in Figure 3.5.

Before concluding this chapter we should remark that these results could be extended to subgraphs G_i with different weights. Notice that the function h from Theorem 3.1 is a summarizing function of the other subgraphs weights functions. In the next chapter we will introduce one way in which algorithms in distributed systems are analyzed.

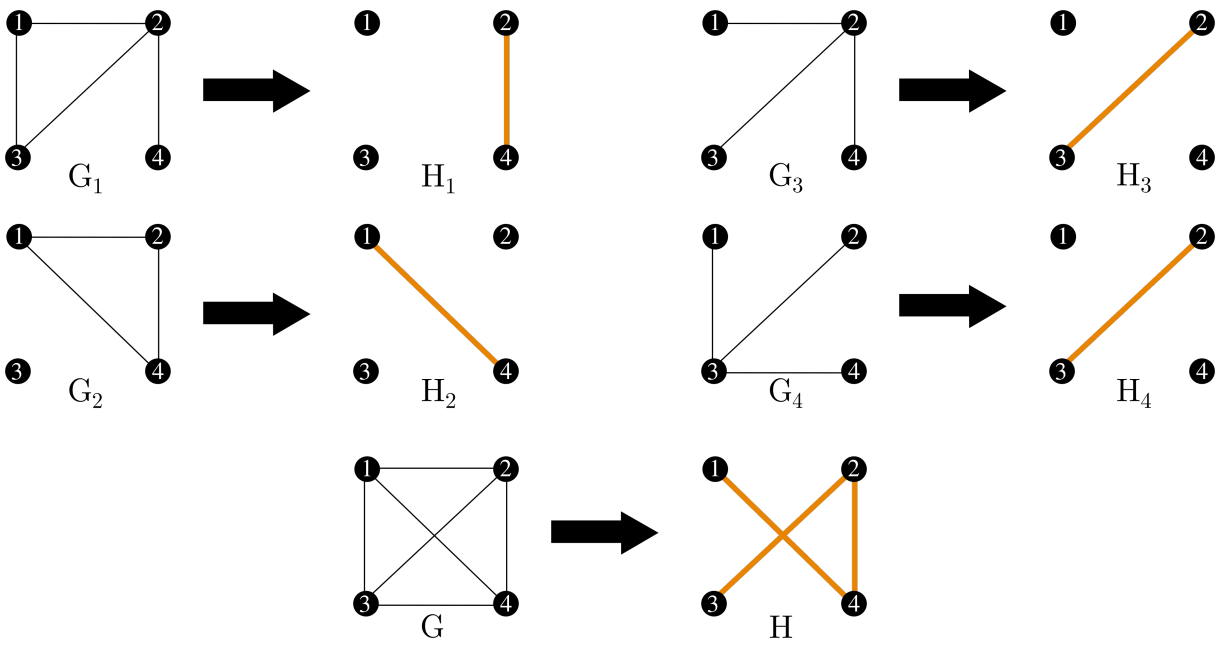


Figure 3.5: At the top we can observe the subgraphs of K_4 from Example 3.4 with their respective spectral approximations. At the bottom is shown what Theorem 3.1 claims, that is, the union of spectral sparsifiers is a spectral sparsifier of the union of subgraphs.

CHAPTER 4

SUNFLOWERS AND NUMBER-ON-FOREHEAD COMMUNICATION

In this chapter we will introduce the rich subject of communication complexity and its theoretical applications. The main result of this chapter is a multiparty protocol that detects whether or not a given family of subsets is a well known mathematical object called Δ-System. We will start in Section 4.1 describing the *communication protocols* and how they are used to solve problems in distributed systems. The formal definition of protocols and the cost of them will be given as well. We will focus on the most basic and simple scenario where there are just two players. In Section 4.2 we will introduce the *multiparty communication* problems and their two most relevant models, the *Number-In-Hand* and *Number-On-Forehead* model. We will primarily focus on the latter. Then, in the Section 4.3 we will introduce the well known mathematical structure called $\Delta-$System. Furthermore, we will develop a protocol to verify whether such structure exist or not in the Number-On-Forehead model. Finally, in Section 4.4 we will describe how communication complexity helps to develop better algorithms in different computational models.

4.1 Communication Protocols

Suppose that there are two o more computers, systems or humans that want to jointly solve a problem which they cannot do it by themselves. These problems may arise in many contexts such as a computer with multiple processors that need to communicate among them through a bus or many computers distributed along different physical sites that need to process some function over the entire data they hold, or maybe two humans that hold each of them a binary

string number and wish to know if the sum of such numbers is odd or even. More generally, if any problem has its underlying data distributed among different *pieces* then, there will be some kind of communication among the parts in order to solve such problem. The component, computers or systems that holds these pieces of distributed data are usually called *players*, *sites* or *parties*.

As in the study of algorithms, the complexity associated to a communication problem measures the amount of resources used to solve problems. In this case, the resources are the bits exchanged. A problem may have different solutions, the measurement of bits exchanged of a given solution is called the *cost* of the solution. We should emphasize that the complexity of a given problem is the cost of the most efficient solution.

The problem is usually represented as a function f over the inputs of every party. A solution to the problem is usually called a *protocol*. A protocol specifies a sequence of interactions among the parties, just as an algorithm specifies a sequence of instructions. The maximum number of bits exchanged in the protocol over the worst-case input is the *cost* of the protocol. The complexity of a problem is the *deterministic communication complexity* of the function f, that is the minimum cost over all protocols which compute f.

Protocols usually have a pictoric and more intuitive representation called *protocol trees*. A protocol tree is a tree where the internal nodes and the root represent parties and the edges represent the communicated bits from one party to another. The leaves of the tree represent the outcome of the protocol. An example is showed in Figure 4.1. Now, we will introduce the definition of protocol trees which can be found in [23].

Definition 4.1. A protocol $\mathcal{P}$ over a domain $X \times Y$ with range Z is a binary tree where each internal node v is labeled either by a function $a_v : X \to \{0,1\}$ or by a function $b_v : Y \to \{0,1\}$, and each leaf is labeled with an element $z \in Z$. The value of the protocol $\mathcal{P}$ on the input (x, y) is the label of the leaf reached by starting from the root, and walking on the tree. At each internal node v labeled by a_v walking on the left if $a_v(x) = 0$ and right if $a_v(x) = 1$, and at each internal node labeled by b_v walking left if $b_v(y) = 0$ and right if $b_v(y) = 1$. The cost of the protocol $\mathcal{P}$ on input (x, y) is the length of the path taken on input (x, y). The cost of the protocol $\mathcal{P}$ is the height of the tree.

We will start with the most basic model, which arises from two parties, let us analyze the following example.

Example 4.1. Suppose that Alice and Bob hold $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ respectively and they wish to compute the function $f : (\{0,1\}^n)^2 \to \{0,1\}$ defined as follows

$$f(x,y) = \begin{cases} 1 & \text{if } x + y \text{ is an odd number} \\ 0 & \text{otherwise} \end{cases} . \tag{4.1}$$

In the following we will describe the protocol presented at Figure 4.1 according to Definition 4.1. Alice and Bob both evaluate the functions $a_1 : X \to \{0,1\}$ and $b_2, b_3 : Y \to \{0,1\}$ at the

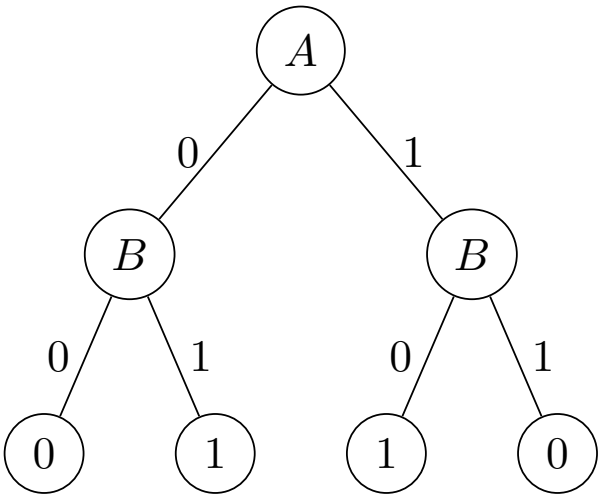


Figure 4.1: A protocol tree to evaluate the Function 4.1 from Example 4.1. The protocol goes as follow, one of the parties computes its less significant bit and send it to the other party. Then, the other party evaluates if the numbers sum up to an odd number or not. Finally, the second party may send $f(x, y)$ to the other party and the total communication cost is 2 bits. A more detailed explanation is given in Example 4.1.

first and second level respectively. The function a_1 evaluates to 0 if $x_{n-1} = 0$ as well as b_2 and b_3 evaluate to 0 if $y_{n-1} = 0$ (the less significant bits). On the other hand, if the values of x_{n-1} and y_{n-1} are 1 then, a_1, b_2 and b_3 evaluate to 1. Then, the paths in the tree are formed by the evaluation of the function on the inputs x and y respectively. The values getting by the functions labeled every edge of the tree. If x and y sum up to an even number then, their values are represented by the most right and most left paths of the protocol tree with a leaf labeled with 0. Otherwise, the two internal paths lead to leaves labeled with 1. The protocol tree is observed as well in Figure 4.1.

Notice that, the problem is usually represented as a function that depends on the distributed data and, the most important resource we want to measure is the amount of bits exchanged. Given that, we will assume that the parties have unlimited computational power.

Now, we will introduce the multiparty communication model which is of great interest in our work.

4.2 Multiparty Communication Models and Number-On-Forehead Model

A multiparty communication model involves more than two parties. There are three natural models according to the way of communication among the parties [24]. The first one is the *blackboard model*, where any message sent by a player is written on a blackboard visible to all the parties. Imagine a board game with k players siting around a table where all the game movements should be done on the table where every player can see it. The second model is

called the *message-passing model*, where a player p_i sending a message specifies another player p_j that will receive this message. The third model is called the *coordinator model*, where there is an additional $(k+1)-$th player called the *coordinator*, who receives no input. Players can only communicate with the coordinator, and not with each other directly. The message-passing model and the coordinator model are both related to each other [24] and to another computational model studied in distributed computing called the *congested clique model* [1]. From now on, we will focus on the blackboard model of communication. There are two models that are reference for this subject. The *Number-In-Hand* (NIH) model and the *Number-On-Forehead* (NOF) model. The first is a generalization of the two party case where every site only has access to its own input. On the contrary in the NOF model every site has access to all inputs but not to its own. Both models have theoretical applications on proving lower bounds on other computational models and we will also summarize them in Section 4.4. Now we will introduce the NOF model.

Let $P_1, P_2, \ldots, P_s$ be a set of sites where a site P_j has an input $x_j \in \{0,1\}^r$, with r a positive integer. In a multiparty communication protocol, with $s \geq 3$, the sites want to jointly compute a function $f : \{0,1\}^r \times \cdots \times \{0,1\}^r \to Z$ for some finite codomain Z. In the Number-On-Forehead model of communication, or NOF model, each site only has access to the other sites' input but not to its own, that is, a site P_j has access to $(x_1, ..., x_{j-1}, x_{j+1}, ..., x_s)$. In the board game's context the NOF model corresponds to k players holding each one a card on its forehead and only visualizing the cards of the other parties, and where all game movements depend upon the card that every player can see. In order to compute f the sites must communicate, and they do so by writing bits on a blackboard which can be accessed by all sites in the model. The protocol tree $\mathcal{P}$ in the NOF model is a binary tree as well. The only difference is that the internal functions of every node are represented as $f_i : X_1 \times \cdots \times X_{i-1} \times X_{i+1} \times \cdots \times X_s \to Z$ for $i \in [s]$.

Now, let us analyze the following example.

Example 4.2. There are $s \geq 3$ sites who wish to compute the function $f : (\{0,1\}^n)^s \to \{0,1\}$ defined as follows

$$f(x_1, x_2, \ldots, x_s) = \begin{cases} 1 & x_i = x_j \text{ for all } i, j \in [s] \\ 0 & \text{otherwise} \end{cases}.$$

This function can be computed with at most 2 bits of communication. First, a given site P_i verifies whether the values $(x_1, ..., x_{i-1}, x_{i+1}, ..., x_s)$ are equal or not. If not, then P_i sends 0 to the blackboard and the protocol ends with 1 bit of communication. Otherwise, P_i sends 1 to the blackboard and the protocol continues. Next, any other site P_j verifies if $(x_1, ..., x_{j-1}, x_{j+1}, ..., x_s)$ are equal. If they are not equal, then x_i is the only input different from the rest and P_j sends 0 to the blackboard. Otherwise, the entire set of inputs are equal and P_j sends 1 to the blackboard and the protocol ends with 2 bits of communication. The protocol tree for this example is observed in Figure 4.2.

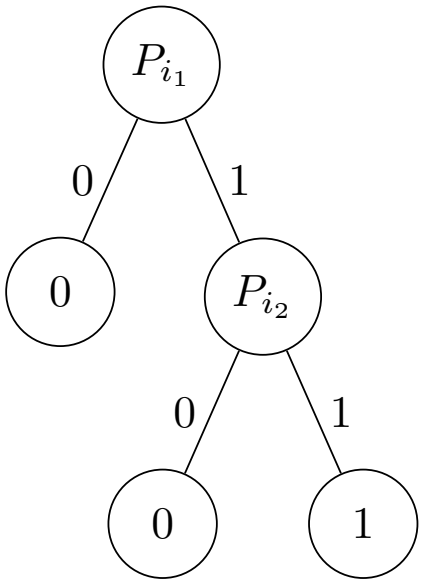


Figure 4.2: Given any two sites P_{i_1} and P_{i_2} with internal functions f_{i_1} and f_{i_2} respectively. The functions f_{i_1} and f_{i_2} output 1 if their inputs are equal. P_{i_1} sends 0 if the values he holds are not equal, this is observed at the most left leaf. Otherwise, he sends 1 and P_{i_2} takes its turn. If P_{i_2} notice that its values are not equal then, he reports 0 which is observed at its most left child. Otherwise he reports 1. The protocol concludes with communication cost at most 2 bits.

The example above shows that a protocol in the NOF model can take advantage of the overlapping information among the sites. Now, we will introduce the main function we want to compute.

4.3 Δ-Systems in NOF model

A *sunflower* or Δ-*System* is a family of sets $\mathcal{A} = \{A_1, ..., A_t\}$ where $A_i \subseteq [n]$ and $(A_i \cap A_j) = \bigcap_{k=1}^{t} A_k = K$ for all $i \neq j$ and some natural n. We call K the *kernel* of $\mathcal{A}$ and A_i a *petal* of the sunflower. The family $\mathcal{A}$ is a *weak* Δ-*System* if $|A_i \cap A_j| = \lambda$ for all $i \neq j$ for some constant λ [25]. It is known that if $\mathcal{A}$ is a weak Δ-System and $|\mathcal{A}| \geq \ell^2 - \ell + 2$, where $\ell = \max_{i=1}^{t}\{|A_i|\}$, then $\mathcal{A}$ is a Δ-System [26].

In the NOF model every site P_i holds a subset A_i but it is not aware of it. Given that P_i is not aware of A_i but the rest, we will denote $\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_s\}$ as $\mathcal{A}_{-i}$. We will say that P_i *observes* a Δ-System if and only if $\mathcal{A}_{-i}$ is a Δ-System. Now, as A_i is a subset of $\{1, \ldots, n\}$ we may use an n-binary string x_i to represent the subset A_i with $x_{ij} = 1$ if $j \in A_i$ and $x_{ij} = 0$ otherwise. Then, we can define the following function $f : (\{0,1\}^n)^s \to \{0,1\}$ as follows

$$f(x_1, \ldots, x_s) = \begin{cases} 1 & \text{if the sets } A_i \text{ uniquely intersect at some subset } K \\ 0 & \text{otherwise} \end{cases}.$$

Now, we will develop a protocol to compute f in the NOF model. Notice that the existence of

the sunflower may reduce the communication cost of any protocol due to the common knowledge that every site will be aware of. We express that in the following fact

Fact 4.1. If $s = |\mathcal{A}| \geq 3$ and $\mathcal{A}$ is a Δ-System with kernel K, then any $\mathcal{A}_{-i}$ is a Δ-System with kernel K.

Now, we will introduce some technical lemmas and propositions that will help us to construct a protocol for verifying whether or not a given family of subsets is a Δ-System. The following lemma states a sufficient condition for the existence of a Δ-System in the input data in the NOF model with the requirement, however, that we need at least four or more sites.

Lemma 4.1. Let $s = |\mathcal{A}| \geq 4$. If, for all $i \in [s]$, we have that $\mathcal{A}_{-i}$ is a Δ-System, then $\mathcal{A}$ is a Δ-System.

Proof. Suppose that $\mathcal{A}$ is not a Δ-System, then we want to prove that for some $1 \leq i \leq s$, $\mathcal{A}_{-i}$ is not a Δ-System.

With no loss of generality, suppose that there exists exactly two sets A_i and A_j that certify that $\mathcal{A}$ is not a Δ-System; that is, there exists A_i and A_j such that $A_i \cap A_j = K'$, and, for any $a \neq i$ and $b \neq j$, it holds that $A_a \cap A_j = A_b \cap A_i = K$, with $K \neq K'$. Now take any $\mathcal{A}_{-c}$, with c different from i and j. Then $\mathcal{A}_{-c}$ cannot be a Δ-System because A_i and A_j belong to $\mathcal{A}_{-c}$ and there is at least another set in $\mathcal{A}_{-c}$ because $|\mathcal{A}| \geq 4$. □

With Lemma 4.1 we can construct a protocol that certifies whether $\mathcal{A}$ is a Δ-System or not with at most s bits of communication. Before doing that let us analyze the following fact

Fact 4.2. Let $\mathcal{A}$ be a family of sets in the NOF model and let $i \in [s]$. If $\mathcal{A}_{-i}$ is not a Δ-System then $\mathcal{A}$ is not a Δ-System.

Proof. Suppose that $\mathcal{A}_{-i}$ is not a Δ-System. Then there must be two sets, A_j and A_k such that $A_j \cap A_k = K'$ and for any $a, b \neq i$ and $a \neq j$ and $b \neq k$, it holds that $A_a \cap A_j = A_b \cap A_k = K$ with $K \neq K'$. In order to get $\bigcap_{l \leq s} A_l$ we may intersect A_i with K. But, notice that no matter what elements A_i has, the sets A_j and A_k will always avoid $\mathcal{A}$ being a Δ-System. □

With Fact 4.2 the sites can know when $\mathcal{A}$ is not a Δ-System. Now, the following proposition describes a protocol to detect a Δ-System in the NOF model.

Proposition 4.1. There exists a protocol that verifies if $\mathcal{A}$, with $|\mathcal{A}| \geq 4$, is a Δ-System or not with at most s bits of communication exchanged.

Proof. Every site P_i will send a bit indicating whether or not $\mathcal{A}_{-i}$ is a Δ-System, 1 if it is and 0 otherwise. If s 1's are sent to the blackboard then $\mathcal{A}$ is a Δ-System by Lemma 4.1. If a 0 is sent then by Fact 4.2 $\mathcal{A}$ is not a Δ-System. □

With Proposition 4.1, a multiparty communication protocol with a number of sites $s \geq 4$ can check for the existence of a sunflower structure in its input data. Furthermore, if the input

data is allocated among sites as a sunflower, then, by Fact 4.1, any site immediately knows the kernel of the sunflower.

Now, we will improve the protocol described in Proposition 4.1 to detect a Δ-System with a constant amount of communication. Let us analyze the following Lemma.

Lemma 4.2. Let $s = |\mathcal{A}| \geq 4$. If $\mathcal{A}$ is not a Δ-System then no more than 2 sites observe a Δ-System.

Proof. We will follow the proof of Lemma 4.1. Suppose that $\mathcal{A}$ is not a Δ-System. Then, there must be at least two sites A_i and A_j such that they intersect at K' with $K' \neq K = \bigcap_{l=1}^{s} A_l$. We will analyze three cases: (i) Exactly two sites intersect at K', (ii) More than two, but not $s-1$, sites intersect at K' and (iii) Exactly $s-1$ sites intersect at K'. In the first case, if A_i and A_j are the only subsets that intersect at K' then, $\mathcal{A}_{-i}$ and $\mathcal{A}_{-j}$ are the only subfamilies that are a Δ-System. For the second case, suppose that there exists some A_l that intersects to A_i and A_j at K' as well, then, neither $\mathcal{A}_{-i}$ nor $\mathcal{A}_{-j}$ nor $\mathcal{A}_{-l}$ is a Δ-System because every family has two subsets that intersects at $K' \neq K$. For the third case, $s-1$ subsets intersect at K' and only one subfamily of $\mathcal{A}$ is a Δ-System. □

Notice that the contrapositive of Lemma 4.2 says that we only need to know if any three sites observe a Δ-System to certify that the family $\mathcal{A}$ is indeed a Δ-System.

With the Lemma 4.2 and Fact 4.2 we can define a protocol with a constant amount of communication that verifies whether a family of sets is a Δ-System or not in the NOF model. The former idea is presented in the following lemma

Lemma 4.3. Let $\mathcal{A}$ be a family of subsets $A_i \subseteq [n]$ and $\{\mathcal{A}_{-i}\}$ the family of subsets that represent the NOF model. Then, there exists a protocol that verifies whether or not $\mathcal{A}$ is a Δ-System with at most 3 bits of communication.

Proof. The protocol goes as follow. Let i_1, i_2 and i_3 be three arbitrary sites. Every site $l \in \{i_1, i_2, i_3\}$ must send a bit to the blackboard as follows

$$P_l \text{ sends } \begin{cases} 1 & \text{if } \mathcal{A}_{-l} \text{ is a } \Delta\text{-System} \\ 0 & \text{otherwise} \end{cases}.$$

Now, if P_{i_1} sends 0 then by Fact 4.2 $\mathcal{A}$ is not a Δ-System and the protocol ends with $f(x_1, \ldots, x_s) = 0$ and 1 bit of communication. Otherwise, the protocol continues and P_{i_2} must sends a bit indicating whether $\mathcal{A}_{-i_2}$ is a Δ-System or not. As in the first case, if P_{i_2} sends 0 then, by Fact 4.2 the protocol ends with $f(x_1, \ldots, x_s) = 0$ and 2 bits of communication. On the other side, if P_{i_2} sends 1 the protocol continues. At the third step, P_{i_3} must check whether $\mathcal{A}_{-i_3}$ is a Δ-System or not. If P_{i_3} sends 0 then, by Fact 4.2 we have $f(x_1, \ldots, x_s) = 0$. Otherwise, by Lemma 4.2 we have $f(x_1, \ldots, x_s) = 1$. In both cases the protocol ends with 3 bits of communication. □

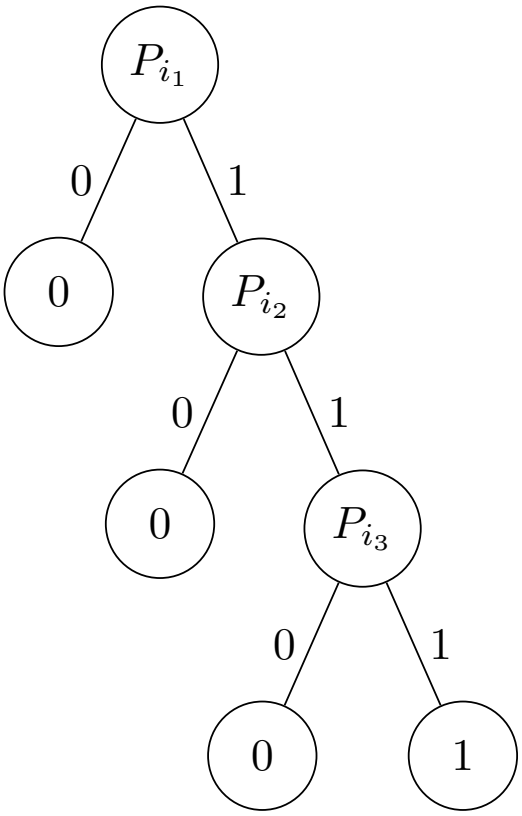


Figure 4.3: The left child of P_{i_1} indicates that P_{i_1} does not observe a Δ-System. At its right child he sends 1 and P_{i_2} takes its turn. If P_{i_2} does not observe a Δ-System then, he concludes the communication sending a 0 to the blackboard. Otherwise, he sends 1 and P_{i_3} takes its turn. Again, if P_{i_3} does not observe a Δ-System he sends 0 to the blackboard. Otherwise he sends 1 and the protocol concludes with at most 3 bits of communication.

We conclude this section with a protocol tree that emerges from Lemma 4.3 which is observed in Figure 4.3. In the next chapter we will use the protocol derived from Lemma 4.3 to construct an efficient protocol in the NOF model for performing a well known machine learning technique under the assumption that the input data has a sunflower structure. Finally, in the next section we will introduce some applications of communication complexity to construct efficient algorithms in different computational models. This will allow us to give a general perspective of how communication complexity could be applied in future works.

4.4 Communication Complexity Applications

Communication complexity is a rich theoretical subject which is widely used to prove optimality of algorithms in different computational models in terms of resources such as *time* and *space*. Communication complexity is mostly applied to prove *lower bounds* on problems. A lower bound usually characterizes the minimum amount of a resource we need to complete a task. Proving lower bounds is considered harder than proving *upper bounds* on problems. This is because one may need to prove that no algorithm performs better with less resources than the possible lower bound. On the other hand, proving an upper bound only requires to construct an algorithm for such problem. Now, we will begin describing some applications of communication complexity.

The most obvious application is found in distributed computing. Distributed computing also studies algorithms and protocols in distributed systems. The differences with communication complexity relies on the fact that the network topology is not known for all the parties and the resource to measure complexity of the algorithms are the *rounds* of communication. More formally, a distributed system is defined as a graph with n vertices where every vertex represents a party. The parties may hold part of the input but the most studied algorithms try to learn something about the structure of the network and there is not inputs besides the names of the vertices. Communication complexity provides lower bounds on the number of rounds needed to execute an algorithm over the network. Usually, those algorithms are called *distributed algorithms* on the contrary of the so called *protocols* in communication complexity. In order to prove lower bounds in a distributed setting the n vertices of the network are mostly partitioned into few parts. Each part represents a standard party in the communication complexity problem and, the messages between the parts are viewed as communication between the parties. Examples of distributed algorithms are given as follows. Given a network, the *diameter* is the largest distance between two vertices in the underlying graph. Intuitively, the diameter corresponds to the maximum time (number of rounds) it takes to pass a message between two points in the network. The proof of the lower bound computing the diameter is by reduction to the randomized two-party communication complexity of disjointness [23, 27]. The *girth*, which is a fundamental parameter of graphs, is the length of the shortest cycle in the graph. The proof of its lower bound is via reduction to the disjointness problem in the NOF model with 3 parties [27].

Communication complexity is also used to prove lower bounds on one of the most natural models for computing boolean functions, that is, boolean circuits. Circuits are directed acyclic graphs whose vertices, often called gates, are associated with boolean operators or input variables. They have two major complexity measures, the *size*, that is the number of vertices on the underlying graph and the *depth*, that is the length of the longest path between two vertices in the graph. It is well known that any function computed by an algorithm on time $T(n)$ steps can also be computed by circuits of size approximately $T(n)$ [28]. This implies that to prove a lower bound on any algorithm it suffices to show that no small circuit can carry out the computation. This can be done by the connection of circuit complexity and communication complexity due to the so called *Karchmer-Wigderson* game [27] which is a two party protocol where Alice and Bob are trying to compute a relation rather than a function [23].

Other applications that rely on reduction from two party communication complexity are *proof systems* and *data structures complexity* [27]. A proof system consists of a set of rules that allow one to logically derive theorems from axioms. Proof systems provide a formal framework for proving theorems and for studying the complexity of proofs [27]. Data structures provide efficient access to data, many fundamental algorithms rely on them. Lower bounds on the performance of data structures are often obtained by appealing to communication complexity. The complexity of a data structure is usually focused on the space of the data structure and the time used to perform the operations.

Finally, the last application of communication complexity we will review is related to the amount of memory required to solve a problem. Specifically, we will focus on algorithms with bounded memory. The standard way of studying such algorithms are *branching programs* or *binary decision diagrams* which are equivalent [29]. A branching program is a *layered directed graph* [27] with length ℓ and width w. Branching programs are related to communication complexity via circuit complexity. Programs with width 5 and length $2^{O(D)}$ can simulate boolean formulas of depth D [27].

A special kind of branching program is a *streaming algorithm*. The input of a streaming algorithm is called *data stream* and is often read once in order: $x_1, \ldots, x_n$. Streaming algorithms are motivated by applications where massive amounts of data need to be processed quickly. Communication complexity is used to show space lower bounds in streaming algorithms for solving a variety of problems. One of the most remarkable problem is the estimation of *frequency moments* which represent demographic information about the data [30]. Its relevance relies on query optimization methods for relational and object-relational database systems as well as many parallel database applications [30]. The k-th frequency moment, for $k \geq 0$, of a sequence of elements $(a_1, \ldots, a_m)$ with $a_i \in [n]$ for $n \in \mathbb{N}$ is defined as

$$F_k = \sum_{i=1}^{k} m_i^k, \tag{4.2}$$

where $m_i = |\{j : a_j = i\}|$ is the number of occurrences of i in the sequence. The streaming

algorithm that approximates the value of Equation 4.2 is due to [30]. Further, a logarithmic lower bound for an approximation of Equation 4.2 was shown in [30] by a reduction to the well studied communication problem called disjointness in the two party model and NIH model. The general approach for the reduction in the two party case is to split the stream data into two parts. Alice simulates the execution of the algorithm on the first part of the stream. Then, she sends to Bob her results allowing Bob to continue the simulation on the second part of the stream [27]. Graph problems such as connectivity, spanners, sparsifiers and matchings were also well studied in the streaming model as well [31, 32, 33].

Branching programs are more powerful than streaming algorithms because it may read the variables multiple times in an arbitrary order. Communication complexity is used to prove lower bounds on branching programs too. To prove a lower bound first it is necessary to prove that any branching program can be efficiently simulated, at least in some sense, by a communication protocol in the NOF model as we shall see. A key claim in [27] states that any branching program of length $\gamma n \log^2 n$, with $\gamma \geq 0$, and width w is equivalent to a k party communication protocol in the NOF model with $O(\gamma \log(w) \log^2(n))$ bits of communication.

CHAPTER 5

APPLICATIONS ON THE MULTICUT GRAPH PROBLEM

In this chapter we will show an application of Theorem 3.1 to distributed data clustering in the Number-On-Forehead model of communication for the case when the input data is allocated as a sunflower among sites. We will start introducing the well known unsupervised machine learning task called clustering in Section 5.1. In Section 5.2 we will introduce the graph clustering technique and the spectral clustering algorithm used to solve it. In Section 5.3 we will developed a protocol for computing clustering in the NOF model.

5.1 Clustering

Data analysis provides a basis for the understanding of all kind of objects and phenomena. One of the most important tasks of data analysis activities is to classify or group data into a set of categories or clusters. Data objects that are classified in the same group should display similar properties based on some criteria [34]. Classification systems play a fundamental role in accomplishing such tasks. Basically, classification systems are either supervised or unsupervised, depending on whether they assign new data objects to one in a finite number of discrete supervised classes or unsupervised categories [34]. The supervised classes are known before the classification procedure begins whereas the unsupervised categories are not. Supervised classification often make use of a mathematical function $y = f(x, w)$ to map the set of input variables $x \in \mathbb{R}^d$, where d is the dimensionality of the space of inputs, to a set of classes $y \in \{1, \ldots, C\}$, where C is the total number of class types and w is a vector of adjustable parameters. The values of w are determined by an inductive algorithm executed on a finite data set of input-output examples $\{(x, y)_i\}_{i \leq N}$ where N is the number of available data [34].

In unsupervised classification, also known as *clustering* or *exploratory data* analysis, no

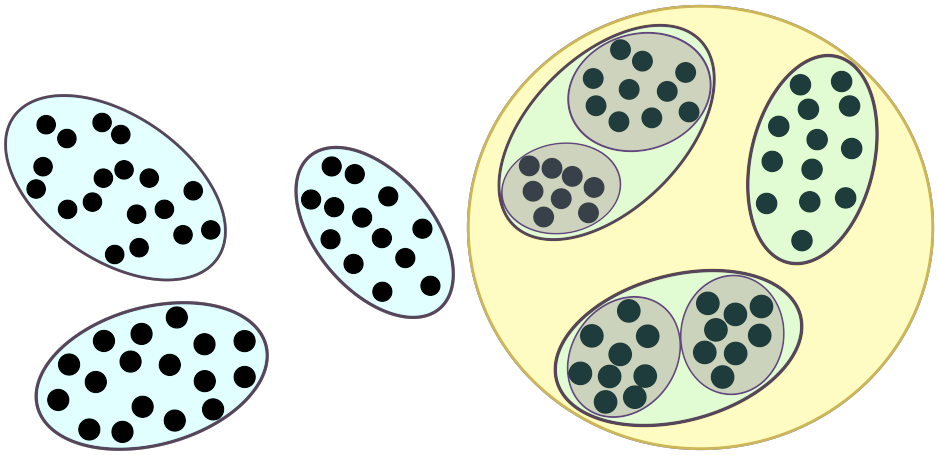


Figure 5.1: On the left side of the image we observe the hard partitional clustering which does not allow points to belong to more than one cluster. On the right side it is observed the hierarchical clustering which construct a tree-like family of connections among the studied objects.

labeled data are available. The goal of clustering is to separate a finite, unlabeled data into a finite and discrete set of "natural", hidden data structures [34]. This label assignment depends upon the closeness among the objects in the dataset. The objects the clustering algorithms deals with are usually represented as a set of points $X = \{x_1, \ldots, x_n\}$ where $x_i \in \mathbb{R}^d$ for some natural d and $i \in [n]$. The features of every object are the components $x_{ij} \in \mathbb{R}$ for $j \in [d]$. The categories that the algorithms are seeking for are subsets $C_l \subseteq X$ for $l \in [k]$ where k is the number of categories. These categories should have a meaningful representation in the application or context where the clustering task is being used on. A rough but widely agreed framework is to classify clustering techniques as hierarchical and partitional clustering, based on the properties of clusters generated [35]. The former may assign more than one category to each object whereas the latter assigns to each object a single category. Mathematically, the two subtasks are defined as follows

Definition 5.1 (Hard Partitional Clustering)**.** Algorithms in this category attempt to find a family of subsets of X, $\{C_1, \ldots, C_k\}$ such that

- $C_i \neq \emptyset$ for all $i \in [k]$,
- $\bigcup_{i=1}^{k} C_i = X$ and
- $C_i \cap C_j = \emptyset$ for all $i \neq j \in [k]$.

Definition 5.2 (Hierarchical Clustering)**.** Algorithms in this category attempt to find a tree-like nested structure of X, $H = \{H_1, \ldots, H_q\}$ with $q \leq n$ such that $C_i \in H_m$ and $C_j \in H_l$ with $m > l$ implies $C_i \subset C_j$ or $C_i \cap C_j = \emptyset$ for all $i, j \neq i, m, l = 1, \ldots, q$.

For hard partitional clustering, each data object is exclusively associated with a single cluster. It may also be possible that an object is allowed to belong to all K clusters with a degree of membership, such approach is studied in *fuzzy clustering* [34]. Examples of the two

types of clustering are observed in Figure 5.1. We will focus on *hard partitional clustering* in this work. Now, *cluster analysis* is a machine learning task that goes from preprocessing data to results interpretation. The typical clustering analysis usually consists on four general steps in a machine learning application which are described as follows.

1. *Feature selection or extraction*, consists on choosing the more distinguishing features from a set of candidates, while feature extraction utilizes some transformations to generate useful and novel features from the original ones.

2. *Clustering algorithm design or selection*, this step is usually combined with the selection of a proximity measure and the cluster criterion function. The proximity measure directly affects the formation of the clusters and the clustering criterion function makes the partition of clusters an optimization problem.

3. *Cluster validation*, this step provides the users a degree of confidence on the clusters obtained from the execution of the algorithms, usually there are three types of validation criteria: external indices, internal indices and relative indices.

4. *Results interpretation*, this step attempts to provide users with meaningful insights from the original data.

The results of this chapter are related to the steps 1 and 2 of the clustering analysis procedure observed above. In particular we will study an algorithm that transforms the original features into a more suitable one. In the following section we will study a specific clustering technique called *graph clustering*. Such clustering technique involves several approaches and we will focus specifically on *spectral clustering*. We will also define the proximity measure and clustering criteria function in order to get the optimization model for our problem.

5.2 Graph Clustering and Spectral Clustering

As we saw, a general but widely agreed framework is to classify clustering techniques based on the structure of the clustering generated. We could also classify them by the theories and techniques used to construct the clustering algorithms. This framework includes *square-error based*, *graph theory*, *combinatorial search techniques*, *fuzzy set theory*, *neural networks* and *kernel techniques* [35]. This work is focused on graph theory techniques. However, the well known square-error based algorithm called *K-means* is used as well and, for a general review of it we recommend the survey [35]. Before getting into the clustering graph theoretical approach we will introduce the notion of proximity. Most clustering algorithms rely on the proximity measure between the objects, this measure could be splitted as well into two categories [34] defined as follows.

Definition 5.3 (Distance or Dissimilarity Function)**.** Let $D : X \times X \to \mathbb{R}$ be a function on the set X that satisfies the following conditions

- Symmetry, $D(x_i, x_j) = D(x_j, x_i)$,
- Positivity, $D(x_i, x_j) \geq 0$ for all $x_i, x_j \in X$.

If conditions

- Triangle Inequality, $D(x_i, x_j) \leq D(x_i, x_k) + D(x_k, x_j)$ and
- Reflexivity, $D(x_i, x_j) = 0$ if and only if $x_i = x_j$ for all x_i, x_j, x_k

hold then, the distance function D is also called a metric.

Definition 5.4 (Similarity Function)**.** Let $S : X \times X \to \mathbb{R}$ be a function on X that satisfies the following conditions

- Symmetry, $S(x_i, x_j) = S(x_j, x_i)$,
- Positivity, $S(x_i, x_j) \geq 0$ for all $x_i, x_j \in X$.

If the conditions

- $S(x_i, x_j)S(x_j, x_k) \leq [S(x_i, x_j) + S(x_j, x_k)]S(x_i, x_k)$ and
- $S(x_i, x_j) = 1$ if and only if $x_i = x_j$ for all x_i, x_j, x_k

are also satisfied then, the function is called a similarity function.

As we mentioned before, the goal of clustering is to divide a set of points into clusters such that points in the same cluster are similar among them and points from different clusters are not. In the graph theoretical context, each data point is interpreted as a vertex of a graph G and each edge between a pair of vertices represents the proximity between this points. There are a lot of definitions of graph clustering in the literature, for a reference we recommend [36]. Despite all the definitions, there is a desirable property that a graph clustering algorithm should hold. Every cluster as a set of vertices should be connected, that is, for every pair of vertices there must be at least one, preferably several paths between them. If v cannot be reached from u then, they must not be in the same cluster. Furthermore, the paths should be internal to the cluster, that is, if a subset of vertices C forms a cluster in G then, the induced subgraph by C should be connected in itself. This property is related to a very well known concept in clustering called *minimum cut* [10]. In the following we address two examples of graph clustering from [36].

Example 5.1. Suppose that we have a bipartite graph $G = (C \cup P, E)$ where C represents a set of costumers and P a set of products. Every edge (x, y) represents that the costumer x bought product y. Two possible clustering targets could be to partition the set of costumers by the type of products they purchase or to partition the set of products purchased by the same people.

Example 5.2. Suppose that we have a set of points $X \subset \mathbb{R}^d$ and a similarity function $f : X \times X \to \mathbb{R}$ according to Definition 5.4. We want to find a partition of X such that every pair of points in the same subset has high similarity between them and every pair of points from distinct subsets has low similarity between them. Next, every point in X could be represented as a vertex of a graph G and given two points x and y in X, they are connected in G by an edge with weight $f(x, y)$.

These are just two examples of how clustering problems may arise in a graph theoretical context. However, we will focus on dealing with problems that come up as Example 5.2 in distributed systems. The technique we will use to solve such problem is called *spectral clustering* and a general guide on it can be found in [9]. A solution to the problem in Example 5.2 concerns finding a multicut in G which is equivalent to minimize the function

$$cut(V_1, \ldots, V_k) = \sum_{i=1}^{k} cut(V_i, \overline{V_i}), \tag{5.1}$$

with respect to all families of subsets $V_1, \ldots, V_k \subset V$. However, minimizing Function 5.1 could lead to subsets with only one vertex which is not a reasonable output. Clustering algorithms should partition the set of points into a reasonable family of larger subsets. This problem can be overcome by making the cut larger when the size or volume[1] of their sets are not proportional. The functions *RadioCut* and *Ncut* [9] are used to accomplish be that

$$RadioCut(V_1, \ldots, V_k) = \sum_{i=1}^{k} \frac{cut(V_i, \overline{V_i})}{|V_i|}, \tag{5.2}$$

$$NCut(V_1, \ldots, V_k) = \sum_{i=1}^{k} \frac{cut(V_i, \overline{V_i})}{d(V_i)}. \tag{5.3}$$

The only difference between Equation 5.2 and 5.3 is that the former takes into account the importance of the vertices in terms of their degree. Notice that both functions are different clustering criteria functions and they will lead us to different optimization problems. Now, the function *RadioCut* can be relaxed in the following way

$$\min_{H \in \mathbb{R}^{n \times k}} Tr(H^T L H) \text{ subject to } H^T H = I, \tag{5.4}$$

and the function *Ncut* in the following way

$$\min_{U \in \mathbb{R}^{n \times k}} Tr(U^T L_{sym} U) \text{ subject to } U^T U = I. \tag{5.5}$$

Both equations have the standard form of the trace minimization problem [37] and the Rayleigh-Ritz theorem tells us the solution. Problem 5.4 is solved by a matrix H with the first k eigenvectors of L as columns and Problem 5.5 by a matrix U with the first k eigenvectors of

[1] In this context we define the volume as the generalization of the degree function of a vertex.

L_{sym} as columns [9].

Whereas spectral clustering algorithms based on Problems 5.5 and 5.4 was extensively applied in different machine learning applications, to the best of our knowledge not much theoretical work on the algorithms was done in terms of approximation factors with respect the optimal partition. This scenario changes when we look at Function 2.7 that tells us the proportion between the outer and inner connectivity into a subset of vertices. The notion of *conductance* can be generalized to the well known *k-way expansion* constant which is defined as

$$\rho(k) = \min_{V_1,\ldots,V_k} \max_{1\leq i\leq k} \phi(V_i). \tag{5.6}$$

Despite widespread use of various graph partitioning schemes over the past decades, the quantitative relationship between the k-way expansion constant and the eigenvalues of the Laplacian matrix of a graph were unknown until the authors of [38] proved the following inequality

$$\frac{\lambda_k}{2} \leq \rho(k) \leq O(k^2)\sqrt{\lambda_k}, \tag{5.7}$$

known as the *discrete cheeger inequality*. Informally, Inequality 5.7 shows that G has a k-way partition with low conductance if and only if λ_k is small. This is expressed as a suitable lower bound on the *gap* $\Upsilon = \frac{\lambda_{k+1}}{\rho(k)}$. Well-clustered graphs which satisfy a gap assumption on Υ have been studied in [39]. The gap assumption on Υ is closely related to the gap between λ_k and λ_{k+1} observed in [9]. Structural results that show close connections between the eigenvectors and the indicator vectors of the clusters have been also given in [39]. In fact, if $\{S_i\}_{i=1}^k$ is a k-way partition of G achieving $\rho(k)$, $\{\overline{g}_i\}_{i=1}^k$ are the normalized indicator vectors of $\{S_i\}_{i=1}^k$, $\{\overline{f}_i\}_{i=1}^k$ are the eigenvectors corresponding to the k smallest eigenvalues of L_{sym} and $\Upsilon = \Omega(k^2)$ then the vectors $\overline{g}_i$ and $\overline{f}_i$ are close up to factor of $O(k/\Upsilon)$.

Finally, we will describe the spectral clustering algorithm. First, let us define the spectral embedding $F : V \to \mathbb{R}^k$ by

$$F(u) = \frac{1}{\mu(u)} \cdot (f_1(u), \ldots, f_k(u))^T, \tag{5.8}$$

where $\mu(u) = \sqrt{d_u}$ is typically used as a normalization factor for $u \in V$ [39]. The *spectral clustering* consists on applying the *k-means* algorithm over the embedded points $F(u)$ for every $u \in V$. Now, let $\{A_i\}_{i=1}^k$ be a partition returned by the *spectral clustering* algorithm. It was shown in [39] that every A_i has low conductance and there exists a large overlap between every A_i and S_i. This provides theoretical justification for the widespread use of *spectral clustering* algorithms.

Finally, we should mention that other parameters as *Ncut* can be used as well to bound an approximation factor of the clustering results as well as the k-way expansion constant does [39]. This is because the normalized cut is the sum of conductance of all sets in a given partition. In the following section we will construct a protocol in the NOF model that will exploit the

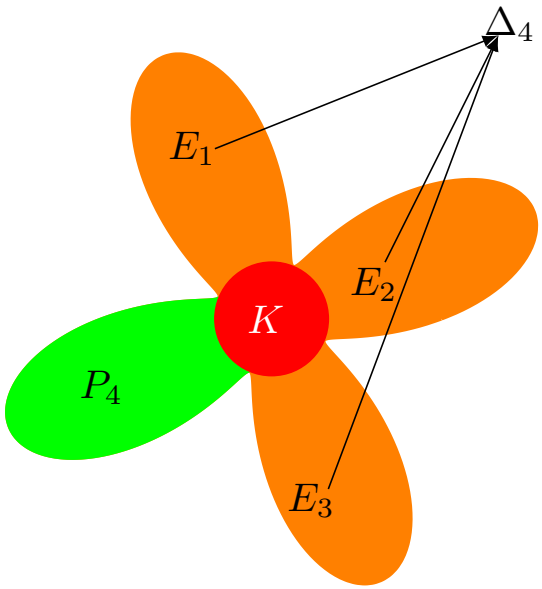


Figure 5.2: In the above picture it is observed a Δ-System of 4 sets representing a NOF model of communication of 4 sites with an underlying graph as an input. Each site holds a subset of edges E_i for $i \in [4]$. The family of edges $\mathcal{E} = \{E_1, E_2, E_3, E_4\}$ constitute a sunflower with unique intersection K. The petal in green represent the site P_4 which is aware only of $\mathcal{E}_{-4}$ and Δ_4 is the symmetric difference between the sets in $\mathcal{E}_{-4}$.

existence of a Δ-System to execute the spectral clustering algorithm over the underlying input graph.

5.3 Data Clustering with Sunflowers

Whereas clustering analysis and spectral clustering are widely studied and implemented in centralized settings, not much work was done in the distributed setting. Spectral clustering was first studied in distributed systems in [1], where a protocol for computing clustering in the *message passing model* and the *broadcast model* was proposed. Both models can be classified as a NIH model of communication. In particular, Chen et al [1]. showed that spectral clustering can be executed in the message passing model and the broadcast model with $\tilde{O}(ns)$ and $\tilde{O}(n+s)$ bits of communication respectively.

In this section, we will present a NOF communication protocol that execute spectral clustering over the underlying input data graph. We will assume that the set of edges is distributed among s sites and they have a unique intersection as we can see in Figure 5.2. We saw in Chapter 4 that we need constant communication to verify whether or not a family of sets has unique intersection. Furthermore, if a Δ-System exists every site immediately knows the kernel. In our application the kernel represents a set of edges that every site is aware of. As every site is aware of a part of the input they only need to know the missing part. This missing part is in the forehead of every site. We will develop a protocol to compute spectral clustering in every site by making every site aware of a spectral sparsifier of the entire input data graph.

First, we start by defining an overlapping coefficient of the edges of G which can be seen as a measure of how well spread out are the edges among sites. We will use the same notation introduced in Chapter 3. The edges of G will be distributed among s sites in the NOF model. The family $\mathcal{E} = \{E_i\}_{i \leq s}$ will represent the distributed edges of G and, the family $\mathcal{E}_{-i}$ will represent the edges that site i can "observe". Furthermore, F_j will be equal to the union of all sets in $\mathcal{E}_{-j}$ and Δ_j for $j \in [s]$ will represent the symmetric difference among the elements in $\mathcal{E}_{-j}$.

Definition 5.5. The overlapping coefficient on site P_j is defined as $\delta(j) = \frac{|\bigcap_{i \neq j} E_i|}{|F_j|}$ and the greatest overlapping coefficient is defined as $\delta = \max_{j \in [s]} \delta(j)$.

The following proposition presents a simple protocol that makes every site aware of the entire input graph.

Proposition 5.1. Let P_j be a site and let $\mathcal{E}$ be a weak Δ-System with each $|E_k| = \ell$ for $k = 1, 2, \ldots, s$, with a kernel of size λ. Suppose that $s \geq \ell^2 - \ell + 3$. If site P_j sends all the edges in Δ_j, then every other site will know the entire graph G. The number of edges this communication protocol sends is at most $|F_j|(1 - \delta) + \ell$.

Proof. We will prove this proposition by showing how each site constructs the graph G. First, a given site P_j computes Δ_j and writes it on the blackboard. Since $s \geq \ell^2 - \ell + 3$, by the result of Deza [26], we know that $\mathcal{E}$ is a sunflower with kernel K and by Fact 4.1 this kernel is the same in all sites. At this point all sites $i \neq j$ know Δ_j, therefore, they can construct G by their own using the kernel K of $\mathcal{E}$. In one more round, one of the sites $i \neq j$ writes E_j so that site P_j can also construct G.

In order to compute the communication cost of the protocol, first notice that $\delta = \lambda/(|\bigcup_{i \neq j} E_i|) = \lambda/(|\Delta_j| + \lambda)$,where we used the fact that the union of all edges in every site equals the union of the symmetric difference and the kernel K. Then we have that $\delta|\Delta_j| = \lambda - \delta\lambda$, which implies $|\Delta_j| = \frac{\lambda - \delta\lambda}{\delta} = |\bigcup_{i \neq j} E_i||(1 - \delta)$, where the last equality follows from the fact that $|\bigcup_{i \neq j} E_i| = \lambda/\delta$. Finally, after E_j was sent to the blackboard the communication cost is $|\bigcup_{i \neq j} E_i||(1 - \delta) + \ell$. □

The protocol described in Proposition 5.1 can be used to make every site aware of the complete input data graph. Once every site knows the entire input they can apply spectral clustering on it. Now, we will show how spectral sparsification can be used to reduce the amount of communicated bits.

Theorem 5.1. Let $\mathcal{E}$ be a weak Δ-system with each $|E_k| = \ell$ for $k = 1, 2, \ldots, s$, and suppose that $s \geq \ell^2 - \ell + 3$. There exists a communication protocol such that after two rounds of communication every site knows an ϵ-spectral sparsifier of the entire graph G with communication cost $O\left(\log\left(\frac{n}{\epsilon^2}\sqrt{1 - \delta}\right)\right)$.

Proof. From [26] we know that $\mathcal{E}$ is a sunflower with a kernel K of size λ and, by Fact 4.1, K is equal in all sites. First, a site P_j computes a spectral sparsifier $H_j = (V, \hat{\Delta}_j)$ of the induced

subgraph $G_j = (V, \Delta_j)$ using the spectral sparsification algorithm of [21]. This way we have that $|\hat{\Delta}_j| = O(n/\epsilon^2)$ where $0 < \epsilon \leq 1/120$. Then site P_j writes $\hat{\Delta}_j$ on the blackboard. Any other site $i \neq j$ constructs an ϵ-spectral sparsifier $H_i' = (V, \hat{E}_j)$ of $G_i' = (V, E_j)$. By Theorem 3.1, the graph $H = (V, \hat{\Delta}_j \cup \hat{E}_j)$ is a ϵ'-spectral sparsifier of G. In a second round, a given site P_i writes $\hat{E}_j$ on the blackboard. Finally, site P_j receives $\hat{E}_j$ and by Theorem 3.1 it can also construct an ϵ'-spectral sparsifier for G. Finally, the communication complexity is upper-bounded by $O\left(\log\left(\frac{n}{\epsilon^2}(1-\delta)\right) + \log\left(\frac{n}{\epsilon^2}\right)\right) = O\left(\log\left(\frac{n}{\epsilon^2}\sqrt{1-\delta}\right)\right)$. □

Finally, from Theorem 5.1 we can observe that every site is aware of an ϵ'-spectral sparsifier H of G and every site can compute spectral clustering.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis we studied three different areas of theoretical computer science with a variety of applications in real world problems. Two of the three areas were focused on graphs related problems and efficient algorithms to deal with them. The third area was more focused on algorithms that perform with distributed data. In the following we present general conclusions about the results we obtained.

General Conclusions

The first and most relevant result of this thesis was the distributed spectral sparsification approximation factor given in Chapter 3. This approximation factor guarantees that any spectral sparsifier algorithm will work under any circumstances where data is distributed and overlapped. This is of great importance due to the wide usage of spectral approximation of graphs in different areas of computer science. We also believe that the structure introduced to achieve such results could have great relevance on other areas where there is a lack of representation of overlapping data. This is because, to the best of our knowledge, no great amount of studies about how repeated data affects algorithms in different computational models was done.

The second result of this thesis was a constant communication protocol for detecting Δ-Systems in the NOF model. As we saw in Chapter 4, communication complexity has a variety of applications in different computational models. The importance of this results relies on optimal protocols that could lead to optimal algorithms in other computational models. For instance, various algorithms try to find intersections among data when solving computational problems. Furthermore, the *sunflower* problem is related to the well known *disjointness* problem which is widely applied to show lower bounds on different problems. We believe that a detailed analysis in the relation between the sunflower and disjointness problem could be further studied.

The last result was an application of the two previous results. We take advantage of the constant communication protocol developed in Chapter 4 to construct a protocol for performing spectral clustering in the NOF model. Due to the constant cost protocol, it was easy to assume that the family of edges of the underlying input graph was a sunflower. Given the common knowledge of the kernel and the possibility of applying spectral sparsification in every site, we showed that spectral clustering could be executed efficiently in the NOF model. Although the NOF model has more theoretical applications, clustering, also known as *the partition* problem, could be used to show optimality in other models.

We conclude this thesis with possible future lines of works which emerged from our studies.

Future Work

Generalization of Results on Distributed Spectral Sparsification

The concept of "union of graph" expressed as sum of Laplacians introduced in Chapter 3 can be further studied. For instance, the family of subsets $\{E_1, \ldots, E_t\}$ could be interpreted as a multigraph and the function $h = \sum_{i=1}^{t} h_i(e)/c_1c_k$ could be generalized to a summarized function for deriving a graph from the multigraph. In that case, better summarizing functions could be studied so that the approximation factor improves. There is also a possibility of studying different approximation factors for every site $\epsilon_1, \ldots, \epsilon_t$ given that every site computes its spectral sparsifier on its own.

Detection of Overlapping Cardinality Partitions in Distributed Systems

The sunflower structure studied in Chapter 4 under the NOF model is a special case of an *overlapping cardinality partition* where there exists only two sets in the partition with cardinalities $c_1 = 1$ and $c_2 = k$. Further studies on a generalization of the constant protocol for detecting Δ-System could be done in order to construct protocols for getting any kind of overlapping cardinality partitions. There is also a possibility of shifting the computational model from NOF to NIH or the congested clique model where common information about the parties can also reduce complexity measures.

Analysis of Optimality Factors of Spectral Clustering with Spectral Sparsification in Distributed Systems

In Chapter 5 we gave a brief introduction to the subject of graph partitioning based on spectral embedding which has a variety of applications as clustering for machine learning. Despite all the recent achievements on showing that such technique is theoretically optimal [39], to the best of our knowledge, not much work has been done in terms of studying the optimality of

spectral clustering applied using distributed data and spectral sparsification. Furthermore, as we saw in Chapter 5 there is no connection between the optimization functions *normalized cut* and *k-way expansion constant*. Such a connection between these two functions could further improve the optimality approximation factors in terms of the eigenvalues of the Laplacian.

Spectral Clustering in the NOF Model Applications

Finally, an ultimate line of work is about how the clustering problem in the NOF model could be applied in other computational models. We propose two sublines of work here. First, it is well known that the partitioning problem is *NP-hard* [9] and, that it is used to show that other problems are *NP-hard* via reduction [10]. Could the clustering problem be applied to show lower bounds as well in the NOF model? Secondly, could we extend the notion of NOF model to sites that are aware of less than $k-1$ inputs and applied this in other computational models? Other questions emerged in the context of *Ordered Binary Decision Diagrams* which are ultimate related to the NOF model of computation as we saw in Chapter 4. Such models are used in different computational models which means that our result could be used to show the minimum requirement of resources for applying clustering in other computational models. In order to do that we first need to show lower bounds on our protocol.

Bibliography

[1] Jiecao Chen, He Sun, David Woodruff, and Qin Zhang. "Communication-Optimal Distributed Clustering". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 3727–3735. URL: http://papers.nips.cc/paper/6562-communication-optimal-distributed-clustering.pdf.

[2] Fabricio A Mendoza Granada, Sergio Mercado, and Marcos Villagra. "Deterministic Graph Spectral Sparsification". In: *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics* 6.2 (2018). URL: https://proceedings.sbmac.org.br/sbmac/article/viewFile/2311/2327.

[3] Sergio Mercado and Marcos Villagra. "Bounds on the Spectral Sparsification of Symmetric and Off-Diagonal Nonnegative Real Matrices". In: *arXiv:2009.11133* (2020). URL: https://arxiv.org/abs/2009.11133.

[4] Sergio Mercado and Marcos Villagra. "A Study of the Optimality of PCA under Spectral Sparsification". In: *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics* 7.1 (2020). URL: https://proceedings.sbmac.emnuvens.com.br/sbmac/article/view/3017.

[5] Fabricio A Mendoza Granada and Marcos Villagra. "Distributed spectral clustering on the coordinator model". In: *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics* 6.2 (2018). URL: https://proceedings.sbmac.emnuvens.com.br/sbmac/article/view/2307.

[6] Fabricio Mendoza-Granada and Marcos Villagra. "Number-On-Forehead Communication Complexity of Data Clustering with Sunflowers". In: *Anais do IV Encontro de Teoria da Computação*. SBC. 2019. URL: https://doi.org/10.5753/etc.2019.6394.

[7] Fabricio Mendoza-Granada and Marcos Villagra. "A Distributed Algorithm for Spectral Sparsification of Graphs with Applications to Data Clustering". In: *Proceedings of the 18th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*. Vol. 5. AIRO Springer Series, 2020. URL: https://arxiv.org/abs/2003.10612.

[8] Daniel A. Spielman and Shang-Hua Teng. "Spectral Sparsification of Graphs". In: *SIAM Journal on Computing* 40.4 (2011), pp. 981–1025. DOI: 10.1137/08074489X.

[9] Ulrike von Luxburg. "A tutorial on spectral clustering". In: *Statistics and Computing* 17.4 (Dec. 2007), pp. 395–416. ISSN: 1573-1375. DOI: 10.1007/s11222-007-9033-z.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. Computer science. MIT Press, 2009. ISBN: 9780262533058. URL: https://books.google.com.py/books?id=aefUBQAAQBAJ.

[11] Daniel A Spielman. "Algorithms, graph theory, and linear equations in Laplacian matrices". In: *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*. World Scientific. 2010, pp. 2698–2722. URL: https://doi.org/10.1142/9789814324359_0164.

[12] A.E. Brouwer and W.H. Haemers. *Spectra of Graphs*. Universitext. Springer New York, 2011. ISBN: 9781461419396. URL: https://books.google.com.py/books?id=F98THwYgrXYC.

[13] Marina Meilă and Jianbo Shi. "Learning Segmentation by Random Walks". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. Denver, CO: MIT Press, 2000, pp. 837–843. URL: https://dl.acm.org/doi/abs/10.5555/3008751.3008873.

[14] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. "Spectral Sparsification of Graphs: Theory and Algorithms". In: *Commun. ACM* 56.8 (Aug. 2013), pp. 87–94. ISSN: 0001-0782. DOI: 10.1145/2492007.2492029.

[15] J. Nešetřil and P.O. de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2012. ISBN: 9783642278754. URL: https://books.google.com.py/books?id=n8UuUePJ%5C_iIC.

[16] Audrey Lee and Ileana Streinu. "Pebble game algorithms and sparse graphs". In: *Discrete Mathematics* 308.8 (2008). Third European Conference on Combinatorics, pp. 1425–1437. ISSN: 0012-365X. DOI: https://doi.org/10.1016/j.disc.2007.07.104.

[17] Ileana Streinu and Louis Theran. "Sparse hypergraphs and pebble game algorithms". In: *European Journal of Combinatorics* 30.8 (2009). Combinatorial Geometries and Applications: Oriented Matroids and Matroids, pp. 1944–1964. ISSN: 0195-6698. DOI: https://doi.org/10.1016/j.ejc.2008.12.018.

[18] Daniel A. Spielman and Shang-Hua Teng. "Nearly-Linear Time Algorithms for Graph Partitioning, Graph Sparsification, and Solving Linear Systems". In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '04. Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 81–90. ISBN: 1581138520. DOI: 10.1145/1007352.1007372.

[19] Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. "Twice-Ramanujan Sparsifiers". In: *SIAM Review* 56.2 (2014), pp. 315–334. DOI: 10.1137/130949117.

[20] Daniel A. Spielman and Nikhil Srivastava. "Graph Sparsification by Effective Resistances". In: *SIAM Journal on Computing* 40.6 (2011), pp. 1913–1926. DOI: 10.1137/080734029.

[21] Yin Tat Lee and He Sun. "Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time". In: *SIAM Journal on Computing* 47.6 (2018), pp. 2315–2336. DOI: 10.1137/16M1061850.

[22] M. Kapralov, Y. T. Lee, C. N. Musco, C. P. Musco, and A. Sidford. "Single Pass Spectral Sparsification in Dynamic Streams". In: *SIAM Journal on Computing* 46.1 (2017), pp. 456–477. DOI: 10.1137/141002281.

[23] Eyal Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997. ISBN: 9780521560672. URL: https://books.google.com.py/books?id=yiV6pwAACAAJ.

[24] Jeff M. Phillips, Elad Verbin, and Qin Zhang. "Lower Bounds for Number-in-Hand Multiparty Communication Complexity, Made Easy". In: *SIAM Journal on Computing* 45.1 (2016), pp. 174–196. DOI: 10.1137/15M1007525.

[25] Alexandr V. Kostochka. "Extremal Problems on Δ-Systems". In: *Numbers, Information and Complexity*. Ed. by Ingo Althöfer, Ning Cai, Gunter Dueck, Levon Khachatrian, Mark S. Pinsker, Andras Sárközy, Ingo Wegener, and Zhen Zhang. Boston, MA: Springer US, 2000, pp. 143–150. ISBN: 978-1-4757-6048-4. DOI: 10.1007/978-1-4757-6048-4_14.

[26] Michel Deza. "Solution d'un problème de Erdös-Lovász". In: *Journal of Combinatorial Theory, Series B* 16.2 (1974), pp. 166–167. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(74)90059-8.

[27] A. Rao and A. Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020. ISBN: 9781108497985. URL: https://books.google.com.py/books?id=n23IDwAAQBAJ.

[28] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN: 9781139477369. URL: https://books.google.com.py/books?id=nGvI7cOuOOQC.

[29] I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000. ISBN: 9780898719789. URL: https://books.google.com.py/books?id=xqqJj42ZoXcC.

[30] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *Journal of Computer and System Sciences* 58.1 (1999), pp. 137–147. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1545.

[31] Andrew McGregor. "Graph Stream Algorithms: A Survey". In: *SIGMOD Rec.* 43.1 (May 2014), pp. 9–20. ISSN: 0163-5808. DOI: 10.1145/2627692.2627694.

[32] Erik D. Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. "On Streaming and Communication Complexity of the Set Cover Problem". In: *Distributed Computing*. Ed. by Fabian Kuhn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 484–498. ISBN: 978-3-662-45174-8. DOI: https://doi.org/10.1007/978-3-662-45174-8_33.

[33] Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. "Dynamic streaming spectral sparsification in nearly linear time and space". In: *arXiv:1903.12150* (2019). URL: https://arxiv.org/abs/1903.12150.

[34] R. Xu and D. Wunsch. *Clustering*. IEEE Press Series on Computational Intelligence. Wiley, 2008. ISBN: 9780470382783. URL: https://books.google.com.py/books?id=kYC3YCyl%5C_tkC.

[35] Rui Xu and D. Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on Neural Networks* 16.3 (2005), pp. 645–678. DOI: 10.1109/TNN.2005.845141.

[36] Satu Elisa Schaeffer. "Graph clustering". In: *Computer Science Review* 1.1 (2007), pp. 27–64. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2007.05.001.

[37] Helmut Lütkepohl. *Handbook of Matrices*. Wiley, 1996. ISBN: 9780471970156. URL: https://books.google.com.py/books?id=wQPvAAAAMAAJ.

[38] James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. "Multiway Spectral Partitioning and Higher-Order Cheeger Inequalities". In: *J. ACM* 61.6 (Dec. 2014). ISSN: 0004-5411. DOI: 10.1145/2665063.

[39] Richard Peng, He Sun, and Luca Zanetti. "Partitioning Well-Clustered Graphs: Spectral Clustering Works!" In: *SIAM Journal on Computing* 46.2 (2017), pp. 710–743. DOI: 10.1137/15M1047209.

[40] A. Umesh Vazirani, A. Christos H. Papadimitriou, and A. Sanjoy Dasgupta. *Algorithms*. McGraw-Hill Education, 2006. ISBN: 9780073523408. URL: https://books.google.com.py/books?id=3sCxQgAACAAJ.

[41] P.D. Lax. *Linear Algebra and Its Applications*. Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley, 2013. ISBN: 9781118626924. URL: https://books.google.com.py/books?id=GNDigtFSMTgC.

Appendices

APPENDIX A

Sets, Functions and Asymptotic Notation

Most chapters of this thesis deals with elements of discrete mathematics. This section reviews some necessary notations to follow this book such as sets, functions and asymptotic notation.

A.1 Sets

A set is a collection of distinguishable objects, called its *members* or *elements*. If an object x is a member of a set S, we write $x \in S$. If x is not a member of S, we write $x \notin S$. We can describe a set by listing its elements. For example a set S containing the numbers 1, 2 and 3 could be expressed as follow $\{1, 2, 3\}$. Given a natural $n \in \mathbb{N}$ the set $\{1, 2, \ldots, n\}$ will be represented as $[n]$. Usually, it is not allowed for a set to contain the same element more than once, and its elements has not prior order. A set with repeated elements is called a *multiset*. Two sets A and B are equal if they contain the same elements. We will denote the *empty* set as $\emptyset$.

If all elements of a set A belongs to a set B then, we say that A is a subset of B and denote as $A \subseteq B$. A set A is a *proper subset* of B, written as $A \subset B$, if $A \subseteq B$ and $A \neq B$. For any set A we have $A \subseteq A$. For two sets A and B we have, $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$. For any three sets A, B and C, if $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$. Finally, for any set A, we have $\emptyset \subseteq A$.

We sometimes define a set in terms of another set. Let A a set and B a subset of A. We can define B by stating a property that distinguishes the elements of B. For example, we can define the set over the naturals by $\{2x : x \in \mathbb{N}\}$. Given two sets A and B, we can also define new sets by applying *set operations*:

- The *intersection* of sets A and B is the set

$$A \cap B = \{x : x \in A \text{ and } x \in B\} \tag{A.1}$$

- The *union* of sets A and B is the set

$$A \cup B = \{x : x \in A \text{ or } x \in B\} \tag{A.2}$$

- The *difference* between the set A and B is the set

$$A - B = \{x : x \in A \text{ and } x \notin B\} \tag{A.3}$$

- The *symmetric difference* of sets A and B is the set

$$A \Delta B = (A - B) \cup (B - A) \tag{A.4}$$

The generalization of intersection and union for n subsets is defined as

$$\bigcap_{i=1}^{n} A_i = \{x | x \in A_i \text{ for all } i = 1, \ldots, n\} \text{ and} \tag{A.5}$$

$$\bigcup_{i=1}^{n} A_i = \{x | x \in A_i \text{ for some } i = 1, \ldots, n\}. \tag{A.6}$$

Also, we will use a generalization of the symmetric difference defined as the union of all elements that belong to just one subset

$$\Delta_{i=1}^{n} A_i = \bigcup_{i=1}^{n} \{x : x \in A_i \text{ and } x \notin A_j \text{ for all } j \neq i\}. \tag{A.7}$$

The elements of a set can also be sets, such sets are called *family or collection of sets*. Given a family of sets $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, we will usually use a shorthand notation defined as $\{A_i\}_{i \leq n}$ to denote such family. Often, all the sets under consideration are subsets of some larger set U called *universe*. For example, if we are considering various sets made up only of naturals, the set $\mathbb{N}$ is an appropriate universe. Given an universe U, we define the *complement* of a set A as $\overline{A} = U - A$. Two sets A and B are *disjoint* if they do not have elements in common, that is $A \cap B = \emptyset$. A collection $\{A_i\}_{i \leq n}$ of nonempty sets forms a partition of a set A if

- the sets are *pairwise disjoint*, that is, $A_i \cap A_j = \emptyset$ for all $i, j \in [n]$,
- their union is A, that is $\bigcup_{i=1}^{n} A_i = A$.

The number of elements in a set A is the *cardinality* (or *size*) of the set and its denoted as $|A|$. The cardinality of the empty set is $|\emptyset| = 0$. If the cardinality of a set is a natural number

we say the set is *finite*; otherwise, it is *infinite*. A finite set of n elements is sometimes called an *n-set*. A subset of k elements of a set is sometimes called a *k-subset*.

We denote the set of all subsets of a given set A, including the empty set and A itself, by 2^A; we call 2^A the *power set* of A. The power set of a set A has cardinality $2^{|A|}$.

The *Cartesian product* of two sets A and B, denoted $A \times B$, is defined as

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}, \tag{A.8}$$

where (a, b) is called an *ordered pair*. When A and B are both finite sets, the cardinality of their Cartesian product is $|A \times B| = |A| \cdot |B|$. The Cartesian product of n sets $A_1, \ldots, A_n$ is the set of *n-tuples*

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) : a_i \in A_i \text{ for } i = 1, \ldots, n\}, \tag{A.9}$$

whose cardinality is $|A_1 \times \cdots \times A_n| = |A_1| \cdots |A_n|$. We denote an n-fold Cartesian product over a single set A by the set A^n whose cardinality is $|A^n| = |A|^n$ if A is finite.

A.2 Functions

A *binary relation* R on two sets A and B is a subset of the Cartesian product $A \times B$. If $(a, b) \in R$, we sometimes write aRb. When we say that R is a binary relation on a set A, we mean that R is a subset of $A \times A$. For example, the "less than" relation on the natural numbers is the set $\{(a, b) : a, b \in \mathbb{N} \text{ and } a < b\}$. An *equivalence relation* is a binary relation R on a set A holding three properties: (i) *Reflexivity*, aRa for all $a \in A$. (ii) *Symmetry*, aRb implies bRa for all $a, b \in A$. (iii) *Transitivity*, aRb and bRc implies aRc for all $a, b, c \in A$. A well known fact about relations is that an *equivalence relation* on a set A is equivalent to a partition of A [10].

Given two sets A and B, a *function* f is a binary relation on $A \times B$ such that for all $a \in A$, there exists precisely one $b \in B$ such that $(a, b) \in f$. The set A is called *domain* of f, and the set B is called *codomain* of f. We sometimes write $f : A \to B$; and if $(a, b) \in f$, we write $b = f(a)$, since b is uniquely defined by the choice of a. Given a function $f : A \to B$, if $b = f(a)$ we say that a is the *argument* of f and that b is the *value* of f at a.

When the domain of a function f is a Cartesian product, we often omit the extra parentheses surrounding the argument of f. For example, if we had a function $f : A_1 \times \cdots \times A_n \to B$, we would simple write $b = f(a_1, \ldots, a_n)$. We also call each element a_i an *argument* of f, though technically the (single) argument of f is the n-tuple $(a_1, \ldots, a_n)$.

If $f : A \to B$ is a function and $b = f(a)$, then we sometimes say that b is the *image* of a under f. The image of a set $A' \subseteq A$ under f is defined as $f(A') = \{b \in B : b = f(a) \text{ for some } a \in A'\}$. The *range* of f is the image of its domain, that is, $f(A)$.

A function f is a *surjection* if its range is its codomain. A function $f : A \to B$ is an *injection*

if distinct arguments to f produce distinct values, that is, $a \neq a'$ implies $f(a) \neq f(a')$. Finally, a function $f : A \to B$ is *bijection* if it is injective and surjective.

A.3 Asymptotic Notation

The order of growth of resources used by an algorithm gives a simple characterization of the algorithm's efficiency and also allows to compare the relative performance of alternative algorithms. When we look at input sizes large enough to make only the order of growth of resources used relevant, we are studying the *asymptotic* efficiency of algorithms. The notations used to describe the asymptotic behaviour of algorithms are defined in terms of functions whose domain is the set of natural numbers.

Definition A.1. [40] Let $f(n)$ and $g(n)$ be two functions from positive integers to positive reals that describe the amount of resources used by an algorithm given the input size n. We say that $f = O(g)$ (which means that "f grows no faster than g") if *there is a constant* $c > 0$ such that $f(n) \leq c \cdot g(n)$

For a given function $g(n)$, we pronounce $O(g(n))$ as "big-oh of g of n" or sometimes just "oh of g of n". The constant c allows to disregard what happens for small values of n. For example, suppose we are choosing between two algorithms for a particular computational task. One is characterized by $f_1(n) = n^2$, while the other by $f_2(n) = 2n + 20$. Which is better? This depends on the value of n. For $n \leq 5$, f_1 is smaller; thereafter, f_2 is the clear winner. The superiority of f_2 is captured by the big-oh notation. Now, suppose that another algorithm comes along and uses $f_3(n) = n + 1$ resources. In this case, f_3 is better than f_2 but only by a constant factor. We usually treat functions as equivalent if they differ only by multiplicative factors.

Bih-Oh notation provides an *asymptotic upper bound* on a function. We can also define an analogous notation for lower bounds as follow

$$f = \Omega(g) \text{ means } g = O(f). \tag{A.10}$$

For a function $g(n)$ we pronounce $\Omega(g)$ as "big-omega of g of n" or sometimes just "omega of g of n". Ω-notation provides an *asymptotic lower bound*.

APPENDIX B

Linear Algebra

This chapter introduces some basic definitions and notations of Linear algebra.

B.1 Spectral Theory

Spectral theory analyzes matrices by decomposing them into their basic constituents. The main object of study is the following equation

$$Ax = \lambda x, \tag{B.1}$$

where $\lambda \in \mathbb{C}$, $A \in \mathbb{C}^{n \times n}$ and $x \in \mathbb{C}^n$. The scalar λ is the *eigenvalue* of A associated to the *eigenvector* x. Notice that by Equation B.1 x is a vector whose direction is not affected by the multiplication of A. It only gets rescaled or reoriented according to the value and sign of λ. This and other relevant properties for this work will be reviewed in this section. First, lets rewrite the Equation B.1 as

$$(A - \lambda I)x = 0. \tag{B.2}$$

Then, as $x \neq 0$ we deduce that x belongs to the *nullspace* of $(A - \lambda I)$, which implies that

$$\det(A - \lambda I) = 0 \tag{B.3}$$

The last assertion implies that there exists an algebraic equation of degree n for λ [41]. The left-hand side of B.3 is called the *characteristic polynomial* of the matrix A and is denoted as p_A.

According to the fundamental theorem of algebra, a polynomial of degree n with com-

plex coefficients has n complex roots; some of the roots may be multiple[1]. The roots of the characteristic polynomial are the eigenvalues of A.

Theorem B.1. Eigenvectors of a matrix A corresponding to distinct eigenvalues are linearly independent.

From Theorem B.1 we deduce the following theorem

Theorem B.2. If the characteristic polynomial of an $n \times n$ matrix A has n distinct roots, then A has n linearly independent eigenvectors.

In this case the eigenvectors forms a basis; therefore every vector y in $\mathbb{C}^n$ can be express as a linear combination of the eigenvectors. In general, when the characteristic polynomial of A has multiple roots, we cannot expect A to have n linearly independent eigenvectors. To face this situation one turns out to *generalized eigenvectors* [41].

B.2 Spectral Theory of Symmetric Matrices

The *scalar product* of two vectors x and y is defined by

$$x^T y = \sum_{j=1}^{n} x_j y_j. \tag{B.4}$$

Definition B.1. Two vectors x and y are called *orthogonal (perpendicular)*, denoted as $x \perp y$, if

$$x^T y = 0.$$

Furthermore, an $n \times n$ matrix M is called *orthogonal* if $M^T M = I$. A *symmetric* matrix $A \in \mathbb{R}^{n \times n}$ obeys the following property

$$A^T = A. \tag{B.5}$$

A symmetric matrix $A \in \mathbb{R}^{n \times n}$ has real eigenvalues and a set of eigenvectors that form an *orthonormal* basis of $\mathbb{R}^n$ [41]. The set of eigenvalues of A are called the *spectrum* of A. Furthermore, every symmetric matrix induce a *quadratic form*

$$\begin{aligned} Q(x) &= x^T A x \\ &= \sum_{i,j=1}^{n} a_{ij} x_i x_j. \end{aligned} \tag{B.6}$$

[1] An element a of a field F is a root of multiplicity k of a polynomial $p(x)$ if there is a polynomial $s(x)$ such that $s(a) \neq 0$ and $p(x) = (x - a)^k s(x)$. If $k = 1$, then a is called a *simple root*. If $k \geq 2$, then a is called a *multiple root*

Quadratic forms are used to show an important result about symmetric matrices which we state as follow

Theorem B.3. Given any symmetric matrix $A \in \mathbb{R}^{n\times n}$, there is an orthogonal matrix M such that

$$M^T AM = D, \tag{B.7}$$

where D is a diagonal matrix whose entries are the eigenvalues of A, M satisfies $M^T M = I$ and the columns of M are the eigenvectors of A.

We will now give a variational characterization of eigenvalues that are very useful to locate them. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix with a set eigenvectors which forms a orthonormal basis of $\mathbb{R}^n$. The *Rayleigh quotient* of A is defined by

$$R(x) = \frac{x^T Ax}{x^T x}, \tag{B.8}$$

for all $x \in \mathbb{R}^n$. Then, the following theorem holds

Theorem B.4. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Denote the eigenvalues of A, arranged in increasing order by $\lambda_1, \lambda_2, \ldots, \lambda_n$. Then

$$\lambda_j = \min_{\dim S = j} \max_{x\in S; x\neq 0} R(x), \tag{B.9}$$

where S linear subspace of $\mathbb{R}^n$.

Theorem B.4 is due to E. Fischer and it is also called *minimax principle*. The matrix A is called *positive* if $x^T Ax > 0$ and *positive semi-definite* if $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$.