FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Balancing Lemmata in Kernelization
Student:	Daria Objeleanscaia
Supervisor:	doc. RNDr. Dušan Knop, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Science 2021
Department:	Department of Theoretical Computer Science
Validity:	until the end of summer semester 2024/2025

Instructions

In 2015 Mnich and Wiese [1] used a so-called balancing lemma in the context of scheduling – namely for Makespan minimization on Parallel Identical Machines. Their use is in parameterized algorithmics where it is assumed that all processing are positive integers and are bounded by the parameter p_max. They showed that one can always transform an optimal solution to a more regular one, where for each job size, all but a few jobs are distributed evenly among the given machines.

The proof of Mnich and Wiese crucially relies on the setting of parallel identical machines, that is, it is not clear how to generalize their original proof even for the setting of related machines (machines with different speed CPUs). The aim of this thesis is to study related problems and discuss possible lines of generalizations while possibly describing many wrong directions. A possible related problem could be the Equitable Connected Partition problem in graphs with bounded vertex cover number [2]. Possible other directions for kernelization can be found via a relation to integer linear programming [3].

Tasks of the thesis:
* Study and describe the balancing technique of Mnich and Wiese
* Describe the formalism of data-reduction techniques
* Try to apply the balancing technique (propose a proof of balancing lemma) for (a special case of) Equitable Connected Partition for bounded vertex cover number
* Discuss the possible research directions.

Electronically approved by doc. Ing. Jan Janoušek, Ph.D. on 25 January 2024 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

References:
[1] Matthias Mnich, Andreas Wiese: Scheduling and fixed-parameter tractability. Mathematical Programming 154(1-2): 533-562 (2015)
[2] Rosa Enciso, Michael R. Fellows, Jiong Guo, Iyad A. Kanj, Frances A. Rosamond, Ondřej Suchý: What Makes Equitable Connected Partition Easy. IWPEC 2009: 122-133
[3] Dušan Knop, Martin Koutecký: Scheduling Kernels via Configuration LP. ESA 2022: 73:1-73:15

Electronically approved by doc. Ing. Jan Janoušek, Ph.D. on 25 January 2024 in Prague.

Bachelor's thesis

BALANCING LEMMATA IN KERNELIZATION

Daria Objeleanscaia

Faculty of Information Technology
Department of Computer Science
Supervisor: doc. RNDr. Dušan Knop, Ph.D.
May 16, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Objeleanscaia Daria. *Balancing Lemmata in Kernelization.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

I am grateful to my supervisor, doc. RNDr. Dušan Knop, Ph.D., for their invaluable guidance and support throughout this research. Without their expertise and encouragement, this work would not have been possible.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 16, 2024

Abstract

This thesis summarizes known results of the Equitable Connected Partition problem and introduces a new kernelization algorithm proving that ECP admits a kernel of size $2^{O(k^2)} \cdot n^{O(1)}$, when parameterized by vertex cover number (k) of the input graph G

Keywords Linear Programming, kernelization, parameterized complexity, balancing lemma, p-equitable graph partition

Abstrakt

Klíčová slova

List of abbreviations

ECP	Equitable Connected Partition
FTP	Fixed Parameter Tractable
VC	Vertex Cover
ILP	Integer Linear Programming
IP	Integer Programming

Chapter 1

Introduction

In graph theory, the Equitable Connected Partition (ECP) problem is a fundamental optimization problem with various real-world applications. ECP seeks to partition the vertices of a graph into equitable and connected subsets. Equitability ensures that each subset has a nearly equal number of vertices, while connectivity ensures that the subsets are connected, facilitating efficient communication or resource allocation.

The main reason behind me starting this research was a problem I encountered at work. I work in a massive company with multiple working locations across Europe and several projects in development. Each month we hire a group of students which need to be allocated to some project. As we want our company to work like workclock we need to allocate our resources effectively and the best way to achieve effectiveness is optimization so I divided into optimization problems research. ECP worked perfectly for my task. First of all, we want all teams to be of approximately same size to ensure that each projects gets equal attention, and that is where equitability comes in handy. In the mean time, a team should be organized and united to achieve progress so every team member should be included, and that is why we want to ensure connectivity of each component (team). Moreover, each project has core employees that are in charge of its development cycle and ensure cooperation between projects as we all work for common purpose. That could be easily associated with vertex cover: if we assign this specific employees to be vertices in vertex cover we ensure that everybody knows each other if not directly, then via 'core vertex cover' employees. New coming employees are not attached to any project yet, and if we call them 'independent workers', their similarity to independent vertex set becomes obvious. This new people are attached to one or more managers (vertex cover), who hired them, but did not make it to a team yet. Last thing in question is how to combine all this constraints of connectivity and equitability when adding new people? That is in fact topic of the research of my thesis!

While ECP offered us valuable insights into our business dilemma rep-

resented as graph partitioning problem, solving it efficiently is challenging, especially for large graphs. Here's where kernelization plays a crucial role. Kernelization aims to simplify problem instances while preserving essential properties, paving the way for more efficient algorithms.

This thesis focuses on kernelizing the ECP problem, specifically parameterized by vertex cover. Vertex cover is a fundamental graph parameter with significant theoretical and practical implications. By parameterizing ECP with vertex cover, we aim to develop efficient algorithms that can handle large graphs more effectively. Further paper will gently introduce all the background research on topic and essential graph knowledge to understand the ECP problem and will present the derivation of its kernel.

1.1 Contribution

This thesis aims to come up with a new kernelization technique for the ECP problem parameterized by vertex cover number, using N-Fold programming methods.

▶ **Theorem 1.1.** *The kernel of* the Equitable Graph Partition *(ECP for short) parametrized by vertex cover of input graph can be solved in time complexity* $2^{O(k^2)} \cdot n^{O(1)}$.

The proof of Theorem 4.4 and the description of the technique is presented in Chapter 4. In Chapter 2, we provide a comprehensive overview of the ECP by introducing necessary notation and presenting the parameters.Finally, we conclude in Chapter 5 by discussing possible directions for future research.

1.2 Paper Organization

For the ease of further understanding Chapter 2 will give a gentle yet concise introduction to most of the terminology used later. It will introduce necessary notions and definitions together with previous research results on the topic. Main part of the thesis — the proof of Theorem 4.4 and the description of the technique is presented in Chapter 4. Finally, we conclude in Chapter 5 by discussing possible directions for future research.

Chapter 2

Preliminaries

Graph partitioning is a group of optimization problems, which aim to break an input graph into pieces satisfying certain constraints, by partitioning vertex set of the graph into disjoint subsets. The problem of graph partitioning or graph clustering deals with the following task: given a graph G = (V, E), where V is the set of graph vertices, E is set of its edges, group the vertices of a graph into clusters. (One might be interested in cases where this graph is weighted, directed, etc., but Hereafter, the term 'graph' refers to a connected, unweighted, un-directed graph, unless stated otherwise.). Formally this problem can be stated as follows

▶ **Definiton 2.1** ([1]). *The* Graph Partitioning *(GP for short) problem can be defined as follows: Given a set of N nodes with a given connectivity, partition them into K sets each with N/K nodes such that the net connectivity (cut-size) is minimal between each set.*

Now that we have discussed the general concept of graph partitioning, I want to follow up with more constrained version of this problem — equitable graph partitioning. In equitable graph partitioning, we aim to divide the graph into clusters such that each cluster has nearly the same number of vertices, such that clusters are connected components of the graph. This optimization problem is a variation of traditional graph partitioning, where we introduce the additional constraint of equitability. Formally, the equitable graph partitioning problem can be defined as follows: it is a natural generalization of the Bisection problem, in which the n vertices of a graph need to be partitioned into d equal-sized parts, for some arbitrary given number d (instead of only two as for Bisection problem). More formally the problem is defined as follows:

▶ **Definiton 2.2.** *Balanced Partitioning*

Let $G = (V, E)$ be a graph, and let us have two positive integers k and d. We say that $V_1, V_2, \ldots, V_r$ is the Balanced Partitioning of G if it is possible to partition the vertices of G into d sets such that each set has a size of at most $\lceil \frac{n}{d} \rceil$ and the size of the cut is at most k. [2]

Given all of the above we can finally state the main problem of the thesis – Equitable Connected Partition (ECP) of the graph, which is the problem of partitioning a graph into a given number of partitions, such that each partition induces a connected subgraph, and the partitions differ in size by at most one. [3] Formally:

▶ **Definiton 2.3.** *Equitable Graph Partitioning Let $G = (V, E)$ be an undirected graph. We say that $V_1, V_2, \ldots, V_r$ is a partitioning of V if and only if $\bigcup_{i=1}^{r} V_i = V$, and for all i, j, $1 \leq i < j \leq r$, $V_i \cap V_j = \emptyset$. A partitioning is equitable if for all i, j, $1 \leq i < j \leq r$, $||V_i| - |V_j|| \leq 1$.[3]*

The importance of the ECP problem comes from the fact of its ability to partition datasets or systems into connected components while maintaining balance and connectivity, making it a fundamental problem with diverse real-world applications, such as: VLSI circuit design [4], parallel computing [5], and image processing [6], among others. As we already know from previous research:

2.1 Graph Notions

For graph-theoretical notation, we follow the monograph by Diestel [7].

▶ **Definiton 2.4.** A simple undirected graph G *is a pair* (V, E)*, where* V *is a non-empty set of vertices and* $E \subseteq (V^2)$ *is a set of edges. We set* $n = |V|$ *and* $m = |E|$*.*

▶ **Definiton 2.5.** *A graph is connected if for every pair of vertices $u, v \in V$, there exists a sequence of distinct vertices $v_1, \ldots, v_\ell$ such that $v_1 = u$, $v_\ell = v$, and $\{v_i, v_{i+1}\} \in E$ for every $i \in [\ell - 1]$.*

▶ **Definiton 2.6.** *Given a vertex $v \in V$, the neighborhood of v is $N(v) = \{u \mid \{u, v\} \in E\}$. The size of the neighborhood of a vertex v is its degree, denoted as $deg(v) = |N(v)|$.*

▶ **Definiton 2.7.** *A vertex cover of a graph G can also more simply be thought of as a set S of vertices of G such that every edge of G has at least one of member of S as an endpoint. The vertex set of a graph is therefore always a vertex cover.*

▶ **Definiton 2.8.** *The smallest possible vertex cover for a given graph G is known as a minimum vertex cover [8]. The size of the vertex cover of a graph G is called the vertex cover number, denoted $\tau(G)$.*

▶ **Definiton 2.9.** *A set of vertices is a vertex cover iff its complement forms an independent vertex set [8]*

From now one all graphs mentioned in this thesis are simple, which means that they are unweighted, undirected, without loops and multiple edges (unless stated otherwise).

2.2 Parametrized complexity

In computer science, parameterized complexity is a branch of computational complexity theory that focuses on classifying computational problems according to their difficulty with respect to multiple parameters of the input or output. The complexity of a problem is then measured as a function of those parameters. This allows the classification of NP-hard problems on a finer scale than in the classical setting, where the complexity of a problem is only measured as a function of the input size. This appears to have been first demonstrated in Gurevich, Stockmeyer & Vishkin [9]. The first systematic work on parameterized complexity was done by Downey & Fellows [10]. For detailed understanding of parameterized complexity I refered to these literature [11, 12, 13], and further on represent a short compilation of it. The NP-hardness of a problem implies that, unless P = NP, there is no polynomial-time algorithm that is able to solve it. This can be solved by analyzing parameterized complexity of the problem. In this case, the running time is analyzed in finer detail: instead of expressing it as a function of the input size, one or more parameters of the instance are defined, and we investigate the effect of these parameters on the running time. The goal is to design algorithms that work efficiently if the parameters of the input instance are small (even if the size of the input is large). [14] Thus, the main objective is to bound the computational challenge of the ”hard” part of the problem by this one or more parameters(k), while ensuring that the remaining part, which depends on the input size (n), can be solved efficiently

The existence of efficient solving algorithms for NP-hard problems is considered unlikely, if input parameters are not fixed; all known solving algorithms for these problems require time that is exponential in the total size of the input. However, some problems can be solved by algorithms that are exponential only in the size of a fixed parameter while polynomial in the size of the input. Such an algorithm is called a fixed-parameter tractable (FPT) algorithm, because the problem can be solved efficiently (i.e., in polynomial time) for constant values of the fixed parameter.

▶ **Definiton 2.10.** *Problems in which some parameter k is fixed are called parameterized problems.[13]*

▶ **Definiton 2.11.** *A parameterized problem that allows for such an FPT algorithm is said to be a fixed-parameter tractable problem and belongs to the class FPT.*

.

For equitable connected partition, parameterized complexity offers a systematic approach to handle its computational complexity. By focusing on the parameter k, which could represent factors such as partition balance or graph structural properties, FPT algorithms efficiently manage the inherent complexities of the problem, providing a more effective solution than traditional algorithms solely based on input size. In parameterized complexity the central notion of efficiency is fixed parameter tractability. FPT contains the fixed parameter tractable problems, which are those that can be solved in time $f(k) \cdot |x|^{O(1)}$ for some computable function f. Typically, this function is thought of as single exponential, such as $2^{O(k)}$, but the definition admits functions that grow even faster. This is essential for a large part of the early history of this class. The crucial part of the definition is to exclude functions of the form $f(n, k)$, such as k^n. The running time of a fixed parameter tractable algorithm is polynomial in the size of the input but can be exponential in terms of parameters.

A surprisingly large number of intractable problems have been shown to exhibit fixed parameter tractable algorithms. A kernelization algorithm is a polynomial time algorithm reducing instances of parameterized problems to equivalent instances whose size can be upper bounded by a function of the parameter. Thus kernelization can be seen as a refinement of the notion of the classical polynomial time tractability from a parameterized perspective.

▶ **Theorem 2.12.** *The parameterized problem is said to admit a kernel if there is a polynomial time algorithm (the degree of polynomial is independent of k), called a kernelization algorithm, that reduces the input instance down to an instance with size bounded by some function $h(k)$ of k only, while preserving the answer. If the function h(k) is polynomial in k, then we say that the problem admits a polynomial kernel.*

In this thesis we would consider the kernelezation of The ECP problem with vertex cover (k) as fixed parameter, which in fact will prove:

▶ **Theorem 2.13.** *Equitable Connected Partition is in FPT with respect to the minimum size of a vertex cover vc(G). [3]*

2.3 Kernelization

Kernelization is a powerful technique used in parameterized complexity theory to preprocess instances of a problem, reducing them to a simpler form while preserving the answer to the problem. The goal of kernelization is to decrease the size of the problem instance while preserving its initial characteristics. In essence, kernelization separates the easy part of the problem, which can be solved efficiently, from the hard part (essential complexity of the problem), which is encapsulated in the parameter k.

▶ **Definiton 2.14.** *A* KERNELIZATION ALGORITHM *for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, for an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, runs in time $O(|I|^c)$, where c is a constant, returns an instance $(I', k') \in \Sigma^* \times \mathbb{N}$, and the following properties hold:*

- *$(I, k) \in L$ if and only if $(I', k') \in L$,*
- *$k' \leq k$,*
- *$|I'| \leq g(k)$, for some function $g\colon \mathbb{N} \to \mathbb{N}$. [15]*

Thus, the result of a the kernelization algorithm is referred to as a KERNEL, with the function g denoting its size. If the function g is polynomial, the kernel is to be considered polynomial as well.

▶ **Example 2.15.** As for our problem: an ETC problem admits a polynomial kernel if there is a polynomial time algorithm that for any instance (G, k) of the problem outputs a new instance (G', k') such that G' has number of vertices bounded by k' and G has a vertex cover at most k if and only if G' has a vertex cover of size at most k'.

The kernelization process typically begins with the identification of problem-specific reduction rules.

▶ **Definiton 2.16** (Reduction rule)**.** *REDUCTION RULE A reduction rule for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is a function $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$, such that, for an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, the function ϕ is computable in polynomial time with respect to both k and the size of I, and returns an equivalent instance (I', k'). [15]*

These rules are then applied iteratively to the input instance in order to simplify it while preserving the solution's existence. The process continues until no further reduction is possible, resulting in a reduced instance (kernel) of initial input which retains the essential complexity of the original problem while significantly decreasing its size. Finally, a fixed-parameter tractable algorithm is applied to the kernel to find the solution.

The significance of polynomial kernels lies in their relationship with fixed-parameter tractability (FPT) algorithms. If a problem has a polynomial kernel, then it is fixed-parameter tractable, meaning there exists an algorithm that can solve the problem efficiently in fixed-parameter time.

The following paragraphs are a condensed compilation of important facts inspired by [14], that are important for further development of our problem: The relationship between kernelization, FPT complexity, and the P = NP problem is complex. If we start with an instance I of an NP-hard problem and can show that in polynomial time we can replace this with an equivalent instance I' with $|I'| < |I|$, then that would imply P =NP. While the existence of a polynomial kernel for a problem implies that it is not NP-hard, it does not necessarily mean that P = NP.

Furthermore, some problems are fixed-parameter tractable but still NP-hard, indicating that fixed-parameter tractability and polynomial kernels do not fully resolve the P = NP question. However, kernelization offers a powerful tool for efficiently solving certain NP-hard problems since it has become possible to derive upper and lower bounds on the sizes of reduced instances (kernels).

2.4 N-Fold integer programming

In this section, we introduce the concept of N-fold integer programming, a powerful technique that forms the backbone of our approach to addressing the Equitable Connected Partition problem's kernel.

We start by introducing the concept of linear integer programming, which serves as a foundation for understanding the more general case of N-fold integer programming. Linear integer programming is a simpler form, where the objective function is the sum of a single linear function. Then, we extend this concept to N-fold integer programming, where the objective function is the sum of N linear functions. Each linear function may have its own set of variables and coefficients. The goal is to find integer values for the variables that optimize the objective function. Therefore, linear integer programming can be seen as a specific case of N-fold integer programming where N is 1.

▶ **Definiton 2.17.** *The integer programming problem is an optimization problem, where* $\mathbb{N}$ *denotes the set of non-negative integers,* A *is an integer matrix and* b*,* c *are integer vectors of suitable dimensions:* $\min\{c^T x \mid Ax = b, x \in \mathbb{N}^q\}$. *[16]*

It is well-known to be generally NP-hard but polynomial time solvable in fixed dimension q, see [17]. For easier understanding of the concept, let us explain the essence of the method on an example

▶ **Example 2.18. Problem**:
A farmer wants to maximize profit by planting wheat and barley on 240 acres of land. Each acre of wheat yields a profit of \$40 and requires \$60 investment, while each acre of barley yields a profit of \$30 and requires \$50 investment. The total budget for planting is \$6000.

Objective: Maximize the profit.

Constraints:

$$\begin{aligned} &\text{Total land: } x + y \leq 240 \text{ acres} \\ &\text{Total budget: } 60x + 50y \leq 6000 \\ &\text{Non-negativity: } x \geq 0, \quad y \geq 0 \end{aligned}$$

Solution:

1. **Objective Function**:
   Maximize $Z = 40x + 30y$

2. **Constraints**:

$$\begin{aligned} x + y &\leq 240 \\ 60x + 50y &\leq 6000 \\ x \geq 0, \quad y &\geq 0 \end{aligned}$$

Standard Form of Linear Programming:

$$\text{minimize } c^T x \text{ subject to} \qquad Ax = b,\ x \in \mathbb{N}^q$$

Where:
$c = [-40, -30]$ is the cost vector, representing the coefficients of the objective function.
$b = [240, 6000]$ is the right-hand side vector, representing the constraints.
$A = \begin{bmatrix} 1 & 1 \\ 60 & 50 \end{bmatrix}$ is the constraint matrix.
$x = [x, y]$ is the vector of decision variables.
$\mathbb{N}$ represents the set of non-negative integers.
$q = 2$ represents the number of decision variables.

In recent years, integer linear programming (ILP) has become a very useful tool in the design and analysis of fixed-parameter tractable algorithms [18]. One of the best known results in this line of research is probably Lenstra's algorithm, roughly showing that ILP with bounded number of variables is solvable in FPT time [17].

Now, when we are set on what ILP is, we can go on with defining what N-Fold IP is. Formally the n-fold integer programming problem is defined as follows (remark here after we base on [16]):

▶ **Definiton 2.19.** *Given a $p \times q$ integer matrix A, positive integer n, and integer vectors $\mathbf{b} = (b_0, b_1, \ldots, b_n)$ and $\mathbf{c} = (c_1, \ldots, c_n)$, where $b_0 \in \mathbb{Z}^q$, $b_k \in \mathbb{Z}^p$, and $c_k \in \mathbb{Z}^q$ for $k = 1, \ldots, n$, the problem is to find a non-negative integer vector $\mathbf{x} = (x_1, \ldots, x_n)$, where $x_k \in \mathbb{N}^q$ for $k = 1, \ldots, n$, that minimizes the objective function $c^T\mathbf{x} = \sum_{k=1}^{n} c_k x_k$. This is subject to the following equations:*

$$\sum_{k=1}^{n} x_k = b_0, \qquad Ax_k = b_k \quad \textit{for} \quad k = 1, \ldots, n$$

The term "n-fold integer programming" refers to the problem being almost separable into n similar programs: $\min\{c_k x : Ax_k = b_k, x_k \in \mathbb{N}^q\}$ in

fixed dimension. However, the constraint $\sum_{k=1}^{n} x_k = b_0$ binds these programs together, resulting in an integer program in large variable dimension nq.

The n-fold matrix of A can be represented as the following $(q + np) \times nq$ matrix, where I_q is the $q \times q$ identity matrix:

$$\begin{bmatrix} A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \\ I_q & I_q & \cdots & I_q \end{bmatrix}$$

Here, I_q is the $q \times q$ identity matrix.

▶ **Theorem 2.20.** *Fix any integer matrix A. Then there is a polynomial time algorithm that, given any n and any integer vectors b and c, solves the corresponding n-fold integer programming problem.*

Building upon previous example of linear programming, I will explain the essence of the N-Fold method when introducing my solution to ECP inChapter 4. In fact, N-Fold is in essence just N linear programming problems combined together and represented as a matrix.

Chapter 3

Related work

My research is most based on finding provided in the referenced literature in my assignment [19, 20, 2], The provided papers explore different aspects of graph partitioning, with a focus on parameterized complexity, which can provide context for my related work on "equitable graph partition parametrized by vertex cover."

In the first paper by Rene van Bevern and colleagues, the focus is on balanced partitioning, specifically Bisection and Vertex Bisection problems, which involve partitioning a graph into equal-sized parts while minimizing the number of edges or vertices that need to be removed to achieve this. The paper highlights that while the Vertex Bisection problem can be particularly challenging, being W[1]-hard when parameterized by both the number of vertices removed and the number of resulting connected components, it becomes fixed-parameter tractable (FPT) when the number of resulting connected components is fixed. This insight is particularly relevant as it suggests a methodological approach because it enforces certain structural constraints (like fixed connected components).

Further, they emphasize that the equitable partitioning can be explored with a similar parameterized approach by focusing on the vertex cover as a parameter.

The extension of these findings to equitable partitions would involve leveraging the small vertex cover size to manage the complexity of partitioning, ensuring each part is not only equal in size but also retains necessary connectivity properties. The approach would likely involve iterative methods to test all possible partitions that can be derived from the vertex cover and the independent set, as outlined in their discussions on algorithmic adaptability for partitioning into more than two parts.

Overall, the research presented in these papers provides foundational algorithms and complexity results that could be adapted and extended to the study of equitable graph partitions parameterized by vertex cover. One good direction for research is to enhance the algorithm provided by me using Balancing

Lemmata by [20].

Chapter 4

Current work

In this section, we propose a new kernelization algorithm for Equitable graph partition problem parameterized by vertex cover number.

For simplicity, in what follows, any graph mentioned is simple, which means that they are unweighted, undirected, without loops and multiple edges. Each graph has its vertex cover V_c together with a positive integer k, which refers to the size of the vertex cover V_c . Similarly, a set of vertices I corresponds to the complement of the vertex cover V_c, i.e., independent set of the given graph G.

Let us recall the problem we are dealing with:

▶ **Definiton 4.1** (Equitable Connected Graph Partitioning Problem)**.** *Let $G = (V, E)$ be a simple graph with vertex cover set $V_c \subseteq V$, $|V_c| = k$, where $k \in \mathbb{N}$. We say that $W_1, W_2, \ldots, W_l$, where $l \in \mathbb{N}$ is the* Equitable Connected Graph Partition *if:*

- $\bigcup_{i=1}^{l} W_i = V$,
- *The subgraph induced by each W_i is connected*
- $|W_i| = |W_j| \pm 1$ *for all i, j.*

In this section, we introduce a new way to solve the Equitable Graph Partition problem, focusing on the size of the vertex cover. Let's start by formally defining the problem: given a simple graph $G = (V, E)$ and a vertex cover V_c of size k, we aim to split the graph into connected parts, with each part having about the same number of vertices and being connected.

Our method involves several steps:

1. Firstly, we introduce a function dividing the vertex cover into distinct subsets. This function generates all possible variations, thereby all potential mappings of vertex cover vertices to partitions.

2. Subsequently, we process each generated mapping. To reinforce the connectivity of the resulting partitions, we allocate k or fewer vertices, from the independent set to each subset of the vertex cover.

3. Next, we write down the constraints of the Equitable Graph Partition problem, converting them further into N-Fold matrix.

4. Finally, we solve this matrix to find the solution to the Equitable Graph Partition problem and demonstrate the equivalence between the Equitable Graph Partition problem and its representation in the form of an N-Fold matrix.

By following these steps, we hope to come up with a good way to solve the Equitable Graph Partition problem efficiently and understand it better using the matrix representation.

We begin dealing with the problem with a simple observation: For any i from 1 to l, $W_i \cap V_c = \emptyset$.

This is because each partition W_i is connected, as required by the problem definition. If W_i did have at least one vertex from the vertex cover, then all its vertices would come from the independent set, meaning they are not connected.

Since we want to consider ECP's version parametrized by vertex cover we start by finding vertex cover of the graph. For that purposes several algorithms can be used: Greedy Algorithm, Approximation Algorithm using Maximal Matching, Exact Algorithm using Brute-Force, Exact Algorithm using Integer Linear Programming (ILP).

Among these algorithms, the greedy algorithm and the approximation algorithm using maximal matching have the best time complexity. The greedy algorithm has a better theoretical complexity, but the approximation algorithm often gives better practical results.

Now that we have found the vertex cover, we define a function

$$f : V \to [l]$$

which maps vertices from the vertex cover set with size k to partitions from W_1 to W_l, where l is the number of partitions. Each partition contains at least one vertex from the vertex cover, due to our previous observation (i.e., $W_i \cap V_c, \forall i$), which can be stated as:
$f : [k] \to [l]$ is l^k.

$$f : [k] \to [l]$$

This function separates Vertex cover into l subsets: $Q_1, \ldots, Q_l$. The number n of such functions $f : [k] \to [l]$ is l^k, as each element in the domain $[k]$ can be mapped to any of the l elements in the co-domain $[l]$.
Moreover, since each partition contains at least one vertex from vertex cover:

$$\text{where } l \leq k, \text{ thus } n \leq l^k \leq k^k$$

Further on, we consider one by one all of the partitions of the vertex cover produced by our function, which is, in fact, set of subsets of vertex cover: $Q_1 \cdots Q_l$. Since vertex cover vertices are already allocated to specific parts we are only left to deal with Independent vertex set IS, where $IS = V \setminus V_c$. We would like to break it into subsets I_{Q_1} to I_{Q_l}, such that all vertices belonging to I_{Q_i}, for $\forall i$, neighbourhood of that vertex is equal to Q_n. $(\forall v \in I_Q : N(v) = Q)$. Now that we have found the vertex cover, we define a function

$$f : V \to [l]$$

which maps vertices from the vertex cover set with size k to partitions from W_1 to W_l, where l is the number of partitions. Each partition contains at least one vertex from the vertex cover, due to our previous observation. (i.e., $W_i \cap V_c, \forall i$), which can be stated as:
$f : [k] \to [l]$ is l^k.

$$f : [k] \to [l]$$

This function tries all possible non-empty subsets of the vertex cover. The number n of functions $f : [k] \to [l]$ is l^k, as each element in the domain $[k]$ can be mapped to any of the l elements in the co-domain $[l]$.
Moreover, since each partition contains at least one vertex from vertex cover:

$$\text{where } l \leq k, \text{ thus } n \leq l^k \leq k^k .$$

▶ Remark 4.2. $I_Q \subseteq I$ is set of Independent vertices that are attached only to vertices of a specific subset Q of the vertex cover.

To ensure connectivity of the k subsets of vertex cover we generated. We do it by preassigning k or less vertices from independent set to make each partition connected. We use k or less vertices because a tree is the least connected type of graph, which need exactly $k - 1 \leq k$ edges to ensure connectivity.

To proceed with our kernelization process of the problem, we need to represent it in an N-fold programming form. Applying the N-fold form to the problem of ECP involves dealing with both local and global constraints to find an optimal division of a graph. Here's how these concepts apply:

Local Constraints

Mutual Exclusivity Each vertex v in the graph G is assigned to one and only one partition W_i.

Partition Coverage Ensures complete and exclusive coverage of all vertices in the graph by the partitions.

Global Constraints

Total Equitability The overall partition must balance the number of vertices across subgraphs as evenly as possible. This is a global constraint because it concerns the relationship between all subgraphs rather than conditions within any single one.

Now that I have outlined the local and global constraints, I proceed to define these constraints in detail:

When addressing the Mutual Exclusivity when allocating the vertices from an independent set into l partitions, I introduce l binary variables for each vertex in the IS. These binary variables serve to denote whether a given vertex is allocated to a particular partition. Specifically, let $X_1^{v_1}, X_2^{v_1}, \ldots, X_l^{v_1}$ represent the binary variables associated with vertex v_1, where l denotes the number of partitions. Consequently, the summation across all such variables belonging to a single vertex is constrained to yield a value of 1, due to the fact that each vertex is exclusively allocated to exactly one partition. Thus:

$$\sum_{i=1}^{k} X_i^v = 1, \quad \text{for } \forall \text{v in the independent set.} \tag{4.1}$$

▶ Remark 4.3. To ensure that partitions stay connected we add the constraint: all binary variables of form $X_k^{v_n}$ are set to zero if vertex v_n is not connected to any of the vertex cover vertices of the $k-th$ partition.

To represent the global constraint that ensures all partitions have approximately the same size, we can express it mathematically using the cardinality of intersections and the sum of binary variables:

$$\left\lfloor \frac{n}{k} \right\rfloor = \sum_{i=1}^{k} |V_c \cap W_i| + \sum_{v \in V_{\text{ind}}} X_i^v \quad \text{for } i \in [k] \tag{4.2}$$

In this equation:

- n is the total number of vertices in the graph.
- k is the number of partitions.
- $\lfloor \frac{n}{k} \rfloor$ represents the floor function of $\frac{n}{k}$, ensuring that each partition has approximately the same size.
- V_c is the vertex cover of the graph.
- W_i is the i-th partition.
- V_{ind} is the independent set of vertices.
- X_i^v represents the binary variable associated with vertex v for partition P_i.

This equation states that the size of each partition (determined by the floor function) should be equal to the sum of the cardinality of the intersection between the vertex cover and the i-th partition, plus the sum of all binary variables associated with independent vertices belonging to the i-th partition, for each i from 1 to l.

Thus, we come to the following N-Fold form:

$$M = \begin{bmatrix} 1 & 1 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 & 1 & \dots & 1 \\ |V_c \cap W_1| & |V_c \cap W_2| & \dots & |V_c \cap W_k| & X_1^v & \dots & X_k^v \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ |V_c \cap W_1| & |V_c \cap W_2| & \dots & |V_c \cap W_k| & X_1^v & \dots & X_k^v \end{bmatrix}$$

Now that I was able to represent ECP in form of an N-fold matrix I can apply The Hemmecke-Shmuel-Romanczuk algorithm, which is an efficient algorithm for solving N-fold integer programming problems, presented in details in this paper. [21] Thus, by applying following algorithm we obtain the main objective of this thesis:

▶ **Theorem 4.4.** *The Kernel for* THE EQUITABLE GRAPH PARTITION *problem, parameterized by the vertex cover of the input graph, can be found in time complexity* $2^{O(k^2)} \cdot n^{O(1)}$*, where* n *is the number of vertices in the graph and* k *is the size of the vertex cover.*

To establish the theorem, I need to demonstrate the equivalence between the EQUITABLE CONNECTED GRAPH PARTITIONING (ECP) problem and its representation in the form of an N-Fold matrix. Specifically, I aim to establish that the existence of a solution for the ECP implies the existence of a corresponding solution for the N-Fold matrix representation. Conversely, I seek to prove that the presence of at least one solution for the N-Fold matrix implies the existence of a solution for the ECP. This bidirectional correspondence underscores the equivalence between the two problem formulations. Formally:

▶ **Theorem 4.5.** *Let* $G = (V, E)$ *be a graph with vertex set* V *and edge set* E*. Let* k *be the number of partitions in the ECP problem, and let* M *be the corresponding N-Fold matrix representation. The* EQUITABLE CONNECTED GRAPH PARTITIONING *problem is equivalent to its representation as an N-Fold matrix if and only if the following statements hold:*

1. *If there exists a solution* S *for the ECP problem, then there exists a corresponding solution* $\mathbf{x}$ *for the N-Fold matrix* M*.*
2. *Conversely, if there exists at least one solution* $\mathbf{x}$ *for the N-Fold matrix* M*, then there exists a solution* S *for the ECP problem.*

Proof.
Statement one To prove the first statement, we need to show that if there exists a solution S for the Equitable Connected Graph Partitioning (ECP) problem, then there exists a corresponding solution $\mathbf{Q}$ for the N-Fold matrix M.

Let's state there exists a solution S for the ECP problem. We'll proceed to show that S corresponds to a valid solution $\mathbf{Q}$ for the N-Fold matrix M.

Since S is a solution for the ECP problem, it satisfies all the constraints of the problem, including the connectivity constraint that each partition induces a connected subgraph.

To begin, it's important to establish that our proposed solution for the Equitable Connected Graph Partitioning (ECP) problem, denoted as S, aligns precisely with a single N-Fold matrix representation. This representation includes all predefined vertex assignments, including the partitioning of the vertex cover into subsets and the assignment of vertices from the independent set to ensure partition connectivity. Further one we will consider exactly this specific N-Fold matrix. Moreover, we have to take into consideration (4.3). For that we go through that all binary variables of form $X_k^{v_n}$ that were set to zero in the N-Fold matrix, and check that vertex $v_n \notin$ partition k. Thus the constrain is satisfied.

We proceed to validate the size constraint(4.2). Given the partition size is already determined by the solution S of our Equitable Connected Graph Partitioning (ECP) problem, we assess the size constraint in the N-Fold matrix representation form. Once we confirm that the obtained number aligns, the constraint is verified.

Now, let's interpret S as a solution for the N-Fold matrix M. For each vertex $v \in V$, we assign v to the partition corresponding to its cluster in S. Specifically, if vertex v belongs to partition i in S, then we set $x_i^* = 1$ and $x_j^* = 0$ for all $j \neq i$. Therefore, the set of conditions (4.1) for v is verified.

This assignment ensures that each vertex is assigned to exactly one partition, satisfying the constraints of the N-Fold matrix M. Additionally, since S satisfies the connectivity constraint of the ECP problem, each partition induced by $\mathbf{Q}$ will be connected.

Therefore, we've shown that if there exists a solution S for the Equitable Connected Graph Partitioning (ECP) problem, then there exists a corresponding solution $\mathbf{Q}$ for the N-Fold matrix M.

Statement two To prove the second item, we need to show that if there exists at least one solution $\mathbf{Q}$ for the N-Fold matrix M, then there exists a solution S for the Equitable Connected Graph Partitioning (ECP) problem.

Let's state there exists at least one solution $\mathbf{Q}$ for the N-Fold matrix M. We'll proceed to show that $\mathbf{Q}$ corresponds to a valid partitioning of the graph $G = (V, E)$.

Since $\mathbf{Q}$ is a solution for the N-Fold matrix M, it satisfies all constraints of the linear programming problem associated with M. Specifically, for each row

of M, the sum of the variables x_i corresponding to the partitions should equal the total number of vertices in V. This ensures that each vertex is assigned to exactly one partition and satisfies (4.1) constraint.

Now, let's interpret $\mathbf{Q}$ as a partitioning of the vertices of G. For each vertex $v \in V$, we assign v to the partition corresponding to the non-zero entry in $\mathbf{Q}$ associated with v. Since $\mathbf{Q}$ satisfies the constraints of M, this assignment ensures that each vertex is assigned to exactly one partition. Moreover, due to remark (4.3) our preliminary vertex assignment guess and setting of some binary variables to zero ensures connectivity and does not allow us to modify graph by adding non-existent edges, thus we do not create non-existent solutions of ECP and satisfy the connectivity constraint.

Therefore, we've shown that if there exists at least one solution $\mathbf{Q}$ for the N-Fold matrix M, then there exists a solution S for the Equitable Connected Graph Partitioning (ECP) problem. To obtain the Equitable Connected Graph Partitioning (ECP) solution from a solution in the N-Fold matrix representation, we need to interpret the values of the binary variables correctly.

Given a solution $\mathbf{Q}$ for the N-Fold matrix M, where $\mathbf{Q} = (Q_1, Q_2, \ldots, Q_k)$, where k is the number of partitions, we can interpret each binary variable X_i as follows:

- If $X_i = 1$, then the corresponding vertex is assigned to partition i.
- If $X_i = 0$, then the corresponding vertex is not assigned to partition i.

By interpreting the values of the binary variables in this way, we can obtain a partitioning of the vertices of the graph $G = (V, E)$ into k partitions.

Now, we are only left to prove that (4.2) holds, which is trivial after previous step of the proof. We sum all of the binary variables representing vertices belonging to same $i - th$ partition in our N-Fold matrix to get the size of the partition and check whether it corresponds to the size of partition we got for ECP in previous step.

Therefore, by reading the values of the binary variables in the N-Fold matrix solution, we can obtain a valid solution for the Equitable Connected Graph Partitioning problem. □

Chapter 5
Conclusion

This thesis provided a kernelization algorithm for the Equitable Connected Graph Partition problem that returns a kernel in $2^{O(k^2)} \cdot n^{O(1)}$ time. The result supports the idea that vertex cover number is a suitable parameter for kernalizing the problem.

5.1 Further Work

This kernelization technique can be further improved by incorporating the Balancing Lemma when distributing k vertices from the independent set to ensure that the vertex cover subsets are connected.

In future research, exploring the integration of the Balancing Lemma into the kernelization process for Equitable Connected Graph Partitioning (ECP) problems could result in more efficient algorithms for solving ECP problems, especially in scenarios where balanced connectivity is crucial, such as network design and optimization.

Bibliography

1. FLOUDAS, Christodoulos A; PARDALOS, Panos M (eds.). *Encyclopedia of optimization*. Boston, MA: Springer US, 2009.
2. BEVERN, René van; FELDMANN, Andreas Emil; SORGE, Manuel; SUCHÝ, Ondrej. On the Parameterized Complexity of Computing Balanced Partitions in Graphs. *Theory of Computing Systems*. 2016, vol. 58, no. 4, pp. 709–745.
3. ENCISO, Rosa; FELLOWS, Michael R.; GUO, Jiong; KANJ, Iyad; ROSAMOND, Frances; SUCHÝ, Ondřej. What Makes Equitable Connected Partition Easy. *Parameterized and Exact Computation,Lecture Notes in Computer Science*. 2009, vol. Not available, pp. 122–133. ISSN 0302-9743, ISSN 1611-3349. Available from DOI: 10.1007/978-3-642-11269-0_10.
4. BHATT, Sandeep N.; LEIGHTON, Frank Thomson. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*. 1984, vol. 28, no. 2, pp. 300–343. Available from DOI: 10.1016/0022-0000(84)90071-0.
5. ARBENZ, Peter; LENTHE, G. Harry van; MENNEL, Uche; MÜLLER, Ralph; SALA, Marzio. Multi-level μ-finite element analysis for human bone structures. In: KÅGSTRÖM, Bo; ELMROTH, Erik; DONGARRA, Jack; WAŚNIEWSKI, Jerzy (eds.). *Proceedings of the 8th International Workshop on Applied Parallel Computing, PARA '06*. Springer, 2007, vol. 4699, pp. 240–250. Lecture Notes in Computer Science. Available from DOI: 10.1007/978-3-540-75755-9_30.
6. LUCERTINI, Mario; PERL, Yehoshua; SIMEONE, Bruno. Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics*. 1993, vol. 42, no. 2, pp. 227–256. Available from DOI: 10.1016/0166-218X(93)90048-S.
7. DIESTEL, Reinhard. *Graph Theory*. 5th. Springer, Berlin, Heidelberg, 2017. Available from DOI: 10.1007/978-3-662-53622-3.

8. PEMMARAJU, Sriram; SKIENA, Steven. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica®*. Cambridge University Press, 2003.
9. GUREVICH, Yuri; STOCKMEYER, Larry; VISHKIN, Uzi. Solving NP-hard problems on graphs that are almost trees and an application to facility location problems. *Journal of the ACM*. 1984, pp. 459–473.
10. DOWNEY, Rod G.; FELLOWS, Michael R. *Parameterized Complexity*. Springer, 1999. ISBN 978-0-387-94883-6.
11. DOWNEY, R. G.; FELLOWS, M. R. *Fundamentals of Parameterized Complexity*. Springer, 2013.
12. FLUM, J.; GROHE, M. *Parameterized Complexity Theory*. Springer, 2006.
13. NIEDERMEIER, Rolf. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. ISBN 978-0-19-856607-6.
14. MARX, D. Parameterized Complexity and Approximation Algorithms. *The Computer Journal*. 2007, vol. 51, pp. 60–78. ISSN 0010-4620, ISSN 1460-2067. Available from DOI: 10.1093/comjnl/bxm048.
15. FOMIN, Fedor V.; LOKSHTANOV, Daniel; SAURABH, Saket; ZEHAVI, Meirav. *Kernelization: Theory of Parameterized Preprocessing: Theory of parameterized preprocessing*. United Kingdom: Cambridge University Press, 2019. ISBN 978-1-107-05776-0. Available from DOI: 10.1017/9781107415157. Publisher Copyright: © Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi 2019.
16. DE LOERA, Jesús A.; HEMMECKE, Raymond; ONN, Shmuel; WEISMANTEL, Robert. N-fold integer programming. *Discrete Applied Mathematics*. 2007, vol. 155, no. 8, pp. 1020–1030. Available from DOI: 10.1016/j.dam.2007.06.009.
17. HENDRIK W. LENSTRA, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*. 1983, vol. 8, no. 4. Available from DOI: 10.1287/moor.8.4.538.
18. GAVENČIAK, Tomáš; KOUTECKÝ, Martin; KNOP, Dušan. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*. 2022, vol. 44, p. 100596. Available from DOI: 10.1016/j.disopt.2020.100596. Optimization and Discrete Geometry.
19. KNOP, Dušan; KOUTECKÝ, Martin. Scheduling Kernels via Configuration LP. In: *ESA*. 2022.
20. MNICH, Matthias; WIESE, Andreas. Scheduling and fixed-parameter tractability. *Mathematical Programming*. 2015, vol. 154, no. 1-2, pp. 533–562.

21. HEMMECKE, Raymond; ONN, Shmuel; ROMANCHUK, Lyubov. N-fold integer programming in cubic time. *Mathematical Programming*. 2009, vol. 120, no. 1, pp. 263–279.