**LEBANESE AMERICAN UNIVERSITY**

# Correlation Clustering with Overlaps: A Multi-parameterized Approach

By

Amin Fakhereldine

A thesis

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Computer Science

School of Arts and Sciences

May 2020

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

# THESIS APPROVAL FORM

Student Name: Amin Fakhereldine I.D. #: 201706566

Thesis Title: Correlation Clustering with Overlaps: A Multi-parameterized Approach

Program: Master of Science in Computer Science

Department: Computer Science and Mathematics

School: School of Arts and Sciences

The undersigned certify that they have examined the final electronic copy of this thesis and approved it in Partial Fulfillment of the requirements for the degree of:

Master of Science in the major of Computer Science

Thesis Advisor's Name: Dr.Faisal Abu-Khzam

Signature: [redacted] Date: 27 / 5 / 2020 (Day / Month / Year)

Committee Member's Name: Dr.Nashat Mansour

Signature: [redacted] Date: 27 / 5 / 2020 (Day / Month / Year)

Committee Member's Name: Dr.Ramzi Haraty

Signature: [redacted] Date: 27 / 5 / 2020 (Day / Month / Year)

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

# THESIS COPYRIGHT RELEASE FORM

## LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

By signing and submitting this license, you (the author(s) or copyright owner) grants the Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic formats and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: Amin Fakhereldine

Signature: [redacted]

Date: 30 / 5 / 2020
Day Month Year

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

# PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that:

1. I have read and understood LAU's Plagiarism Policy.
2. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
3. This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Amin Fakhereldine

Signature: [redacted]

Date: 30 / 5 / 2020
Day Month Year

# ACKNOWLEDGMENT

In the long path of completing this thesis, many people have supported me in the times of confusion and helped me to make this achievement possible. I would like to thank my advisor, Dr. Faisal Abu-Khzam, who reviewed my numerous drafts and helped me with my different ideas. Also thanks to my committee members, Dr. Nashat Mansour, and Dr. Ramzi Haraty, who supported me and were always ready to help me in the process of completing my thesis. And finally, thanks to my family, Maria, and my friends who were always there to support me and offer love and care.

Correlation Clustering with Overlaps:

A Multi-parameterized Approach

Amin Fakhereldine

## ABSTRACT

The Cluster Editing problem asks for transforming a given graph into a disjoint union of cliques by applying a minimal number of edge-editing operations. The allowed operations include addition of non-existing edges and deletion of existing ones. We study a multi-parameterized version of the problem that limits the global number of allowed edge editing operations in the graph and the local amounts of the edge edits performed per vertex. Moreover, we allow the new vertex splitting operation, which allows the resulting clusters to overlap. In other words, data elements (or vertices) will be allowed to be members in more than one cluster instead of limiting them to only one single cluster, as in classical clustering methods. We present a heuristic algorithm and a semi-exact algorithm for the Multi-Parameterized Cluster Editing with Vertex Splitting problem. In our experimental analysis, we study the efficiency of our algorithms as well as the effectiveness of allowing vertex splitting. In particular, we show that allowing vertex splitting yields higher clustering accuracy and higher intra-cluster similarity.

Keywords: Correlation Clustering, Cluster Editing, Fixed-parameter Tractability, Clustering with Overlaps, Vertex Splitting.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

**Figure** **Page**

# Chapter One

# Introduction

In the *Cluster Editing* problem, we are given a simple undirected graph $G$ and a non-negative integer $k$ and asked whether $G$ can be transformed into a disjoint union of cliques, henceforth a *cluster graph*, by applying no more than $k$ edge editing operations. An operation can add an edge to the graph or remove one from it. The problem can model unsupervised correlation clustering [2] in which an undirected graph represents data. Data elements are represented by vertices while edges connect vertices corresponding to similar elements. We aim at connecting similar vertices such that the resulting connected components of the graph will be cliques. The problem was given notable attention due to its applications in various fields such as data mining [14, 33, 23, 15], machine learning [27, 1, 32, 9], computational biology [31], and many others.

The *Cluster Editing* problem forces each vertex to belong to only one cluster. However, in real life applications, a network member can play roles in more than one cluster such as in gene regulatory networks [3]. A first attempt at modeling the clustering with overlaps problem using a parameterized cluster editing problem was made in [22]. More recently, Abu-Khzam et al. [6] introduced the problem of *Cluster Editing with Vertex Splitting* as a new variant in which a vertex is allowed to *split* into two or more vertices and join more than one cluster. An example of elements that play roles in more than one cluster are *protein hubs* that exist in protein networks as a few number

of nodes each having edges with a huge number of nodes. Another example are the social media networks, such as *Facebook* and *LinkedIn*, in which a user usually acts as an active member in more than one group.

Krivánek and Morávek [28] proved *Cluster Editing* to be NP-hard. Shamir et al. [34] proved that the problem is NP-complete, and that it remains NP-complete for $p \geq 2$ where $p$ is the exact number of clusters in the desired solution.

The problem is considered fixed-parameter tractable due to a simple branching algorithm that runs in $\mathcal{O}(3^k)$ [20]. Gramm et al. [25] introduced the first paramterized algorithm for the problem of *Cluster Editing*. The algorithm allows up to $k$ edge editing operations (additions and deletions) and runs in $\mathcal{O}(2.27^k)$. The running time has been a subject to several improvements later. Gramm et al. [24] presented a parameterized algorithm that runs in $\mathcal{O}(1.92^k)$, then an improvement to $\mathcal{O}(1.82^k)$ was done by Böcker et al. [12]. Currently, the most efficient fixed parameterized algorithm is the one in [10] which uses a transformation to an integer-weighted variant that runs in $\mathcal{O}(1.618^k)$. Moreover, a kernel was proved by Gramm et al. [25] of size $2k^2 + k$ for the *Cluster Editing* problem. The problem kernel was subject to several improvements as well. It was improved to $24k$ [21], then to $6k$ [20], and $4k$ [26]. The smallest current kernel for the problem is of size $2k$ [17].

Abu-Khzam [2] introduced a multi-parameterized version that limits the number of edge additions and deletions that can be performed by a single vertex, and sets a lower bound on the sizes of the clusters. It was proven that *Cluster Editing* is solvable by a polynomial time algorithm if the minimum cluster size is greater than twice the total number of operations a single vertex can be part of. Otherwise, a kernel can be obtained with a number of edges linear in $k$. In addition, it was proven that the problem remains NP-hard even when the edge deletion budget is 1 and the edge addition budget per vertex is at most 2. Later, Abu-Khzam et al. [6] introduced the problem of *Cluster Editing with Vertex Splitting* as a new variant. In this variant, a vertex is allowed to split into more than one copy, bringing the possibility of overlapping clusters. The problem was proven to be FPT by the sum of the edge editing operations and the

vertex splittings, and that it has a $4k(k+1)$ kernel and a search algorithm that runs in $O(2^{O(k^2)})$.

Lokshtanov and Marx [29] presented a clustering algorithm that partitions the vertices by respecting some local constraints on each cluster instead of the global constraint on the whole graph which is optimizing the number of operations needed to cluster the graph. The local constraints set on the clusters limit the number of vertices and missing edges in each cluster, the number of edges that can leave a cluster, and the number of non-neighbours a vertex can have in the cluster. The algorithm was proved fixed-parameter tractable by these local bounds.

In this thesis, we study the *Multi-Parameterized Cluster Editing with Vertex Splitting* problem, and we introduce a heuristic algorithm and a semi-exact algorithm to solve it. It is a multi-parameterized approach allowing each vertex to split into more than one copy in order to join more than one cluster. It limits the number of allowed edge additions, edge deletions, and vertex splittings to be performed by a single vertex. In addition, it limits the total number of operations and vertex splittings that can be done to cluster the graph. We also introduce a greedy heuristic for cluster editing with vertex splitting, and a heuristic algorithm that decides on how to partition the neighborhood of a vertex, upon splitting it, into two parts each of which to be the neighborhood of one of its splits. In order to evaluate our clustering algorithms and show the effectiveness of the splitting operation, we compare our semi-exact algorithm to another semi-exact algorithm based on the multi-parameterized approach in [2] that does not allow vertex splitting. We also compare the results of our greedy heuristic when allowing splitting to those when splitting is not allowed. The experimental results show that our algorithms yield near-to-optimal results and that clustering with vertex splitting gives better clustering results because vertices are allowed to belong to more than one cluster due to their similarity to a variety of vertices. This can be seen by the higher intra-cluster similarity achieved by allowing splitting.

The remaining of this thesis is organized as follows: Chapter II presents some preliminaries, Chapter III explains our work, Chapter IV discusses the heuristic algorithm

used to decide on the neighborhood of the splits that will be generated due to every vertex splitting operation, in Chapter V we present our clustering algorithms. The results from our experiments are reported in Chapter VI, and we conclude and present some future work in Chapter VII.

# Chapter Two

# Preliminaries

## 2.1 Terminology

An *undirected* graph $G$ is defined by a pair $(V,E)$ where $V$ is the set of vertices and $E$ is the set of edges. If there is an edge between $u$ and $v$, then it connects $u$ to $v$ and $v$ to $u$. Two vertices $u$ and $v$ are *adjacent* if there is an edge connecting them. Two vertices $u$ and $v$ are said to be *neighbours* if they are adjacent. The *neighborhood* $N(v)$ of a vertex $v$ is the set of vertices adjacent to it. The *degree* of a vertex $v$, $d(v)$, is the number of neighbours of $v$.

A *subgraph* $G'$ of a graph $G(V,E)$ is a graph $(V',E')$ such that $V' \subset V$ and $E' \subset E$. An *induced subgraph* $G[V']$ is a *subgraph* $G'(V',E')$ such that $E'$ consists of all the edges $uv$ in $E$ such that $u$ and $v$ belong to $V'$. A *complete graph* is an undirected graph such that any two vertices are connected by an edge. A *clique(cluster)* is a *complete subgraph*. A *path* is a graph (or subgraph) whose vertices can be ordered so that two vertices are adjacent if and only if they are consecutive in the list. The endpoints of a *path* are the pair of vertices that belong to only one edge in the path. An undirected graph is *connected* if every vertex has a path between it and every other vertex in the graph. A *connected component* is a connected *subgraph* such that no vertex is connected to any other vertices in the *supergraph*.

In addition, graphs can be classified into *weighted* and *unweighted* graphs. *Weighted*

graphs have numbers or values assigned to each edge representing costs, distances, or any other values depending on the problem definition. *Unweighted* graphs do not have weights assigned to edges. Also, it can be considered as if all of the edges have the same weight. In our work, we consider simple unweighted and undirected graphs.

## 2.2 Cluster Editing

*Cluster Editing* asks, given a graph $G$ and an integer $k$, whether we can transform $G$ into a disjoint union of cliques by performing no more than $k$ *edge editing operations*. An operation may add an edge to the graph or delete one from it. We denote by *cluster graph* a graph that has all of its connected components as cliques. Given an input graph $G$ for the clustering problem, we denote by the *solution graph* the *cluster graph* that results from transforming $G$ into a disjoint union of cliques.

In correlation clustering, we are given a set of objects and a function defining the similarity between every pair of objects. The data can be represented by an undirected graph where each vertex represents an object. A *similarity graph* can be obtained by choosing a similarity threshold and setting an edge between every two vertices that have their similarity value greater than this threshold [31]. The goal is to group vertices into clusters while achieving high *intra-cluster similarity* (members of the same cluster are similar to each other) and low *inter-cluster similarity* (members of different clusters are dissimilar).

A *forbidden* edge is an edge that should not be in the solution graph. A *permanent* edge is an edge that should be in the solution graph. A *permanent clique* has all of its edges permanent. A *conflict triple* is a path of length two (three consecutive vertices) and it is named like this because it can never be part of a *solution graph* [2]. We denote by $(u,v,w)$ a *conflict triple* of edges $uv$ and $vw$ and with a missing edge $uw$. We call $v$ the *center of the conflict triple*. The first and simplest algorithm for *Cluster Editing* is a recursive algorithm that searches for a conflict triple $(u,v,w)$ and resolves it by branching on one of three possible branches: deleting one of the existing edges ($uv$ or

*vw*) or adding the missing edge *uw* [25].

The *Cluster Editing* problem allows each vertex to join only one cluster. *Clustering with Overlaps* is a variant of the problem that allows overlapping clusters in which a vertex can join more than one cluster. We employ the notion of *vertex splitting* which is an operation in which a vertex is allowed to split into more than one version. Each version can thus join a separate cluster. A *graph editing operation* can be an *edge editing operation* or a *vertex splitting operation*. A *split vertex* can split into multiple splits where each of them can be a member of a separate cluster.

We consider a multi-parameterized version of the problem that studies the problem based on more than one parameter. These parameters are: the maximum number of edge additions a single vertex can perform, the maximum number of edge deletions a single vertex can perform, the maximum number of times a vertex can split, the maximum number of *graph editing operations* allowed to be performed, the maximum number of splits allowed to be in the graph, and the minimum acceptable cluster size. Therefore, our approach aims at clustering by allowing overlaps and by performing a certain number of operations that include adding edges, deleting edges, and splitting vertices. Figure 1 shows the four possible graph editing operations that can be done to resolve a conflict triple $(u, v, w)$. The mark **P** denotes setting an edge to *permanent*, while a dashed edge denotes setting an edge to *forbidden*. The reason behind these markings is that each of the pairs $\{v_1, u\}$ and $\{v_2, w\}$ must form two separate clusters.

## 2.3 Parameterized Complexity

A problem is said to be *fixed-parameter tractable*, (FPT) with respect to a parameter $k$ if it can be solved by an algorithm that runs in $\mathcal{O}(f(k)n^c)$ where $n$ is the size of the input and $c$ is constant [19]. Implicitly, this gives rise to the notion a *fixed parameter* algorithm: an algorithm whose time complexity is measured as a function of the input length and a (potentially small) parameter independent of the input. Such an algorithm runs in polynomial time in the input length but possibly in exponential time (or

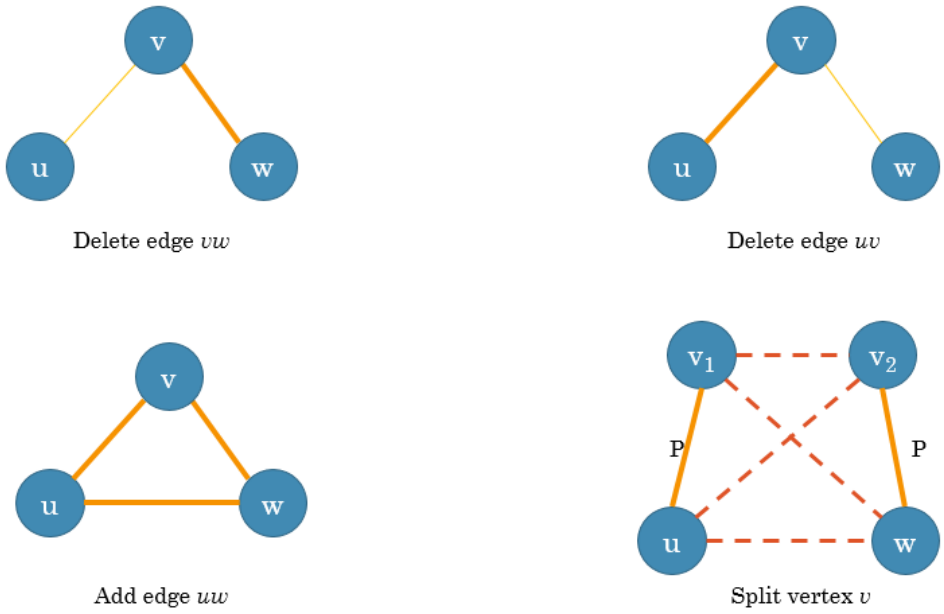


Figure 1: Graph Editing Operations

worse) in the parameter. Fixed-parameter algorithms are interesting when the complex instances of the corresponding problem have relatively small parameter values.

A *kernelization* is a polynomial time reduction procedure of an instance $(I,k)$ of a parameterized problem into another instance $(I',k')$ of the same problem such that:

1. $(I,k)$ is a YES-instance if and only if $(I',k')$ is a YES-instance
2. $k' \leq k$
3. $|I'| \leq g(k)$ for some computable function $g$

A problem has a *kernel* if and only if it is FPT [19], however not every FPT problem has a kernel of polynomial size [13].

## 2.4 Heuristic Algorithms

Heuristic algorithms are algorithms that trade optimality for time. They are based on shortcuts to speed up the decision making procedure. Such algorithms have practical applications especially when finding an optimal solution is impossible or computationally hard. The solution of a heuristic algorithm may sometimes be optimal, but in most

cases it may not. Instead, it can be satisfactory or close-to-optimal but computed in reasonable time.

# Chapter Three

# Our Work

## 3.1 Overview

In our work, we tackle the problem of *Clustering with Overlaps* by presenting new algorithms for a multi-parameterized version that allows vertex splitting. Our work is motivated mainly by previous work done in [2] and [6]. The work in [2] presented a multi-parameterized version of the *Cluster Editing* problem. This model is considered as multi-parameterized because it studies the problem by considering more than one parameter independent from the input. These parameters are: the maximum number of edge editing operations allowed to be performed in the whole graph, the maximum number of edge additions a single vertex can have, the maximum number of edge deletions a single vertex can have, and the minimum acceptable cluster size. It was suggested to allow the vertex splitting operation in applications where data elements can play roles in different clusters.

The work presented in [6] introduced a new variant of the Cluster Editing problem where clusters are allowed to have overlapping vertices. This model allows a vertex to split more than once, thus allowing each split to join a separate cluster. It was projected that better clustering results can be obtained by using the multi-parameterized model presented in [2]. In addition, the challenge of identifying the data sets that should be in common between different clusters was mentioned.

Motivated by the above mentioned research, our work combines the approaches done in [2] and [6] to study a multi-parameterized approach for Cluster Editing that allows vertex splitting. The problem can be then defined as follows:

**The Multi-Parameterized Cluster Editing With Vertex Splitting Problem**

**Input:** Given a graph $G$, and parameters $a$, $d$, $s$, $p$, and $k$

**Question:** Can $G$ be transformed into a cluster graph $G'$ by performing at most $k$ operations from the following:

1. Add an edge
2. Delete an edge
3. Split a vertex: For some vertex $v \in V$, partition $N(v)$ into two disjoint sets $U_1$ and $U_2$ such that $U_1 \cup U_2 = N(v)$. Then replace $v$ by $v_1$ and $v_2$ such that $N(v_1) = U_1$ and $N(v_2) = U_2$.

such that no vertex can gain more than $a$ edges or lose more than $d$ edges or have more than $s$ splits in the solution graph, and the solution graph will have no more than $p$ total splits?

## 3.2 Algorithmic and Implementation Details

During the clustering procedure of our algorithms, an edge may have one of the three following states: *permanent*, *forbidden*, or *unknown*. Initially, all edges will have an *unknown* state. Whenever an edge is set to *permanent*, it cannot be deleted anymore. Therefore, it will be part of the solution (if any exists). On the other hand, when an edge (or a pair of non-adjacent vertices) is set to $forbidden$, it cannot be added anymore. Thus, it will not be part of any (subsequent) potential solution.

Each vertex has the ability to gain and lose edges, as well as splitting into more than one version. The notion of vertex splitting is introduced as a parent-child relationship. For example, if a vertex $v$ will split into two vertices $v_1$ and $v_2$, $v$ will be considered as

the parent of $v_1$ and $v_2$. Equivalently, $v_1$ and $v_2$ will be considered as the children of $v$. Initially, the whole set of vertices in the input graph are parent vertices, and all of their splits that will be created subsequently will be their respective children. Every vertex in the initial set of vertices is given addition, deletion, and splitting budgets. Whenever a child needs to perform some operation, its parent should have enough local budget for that particular operation. Whenever a vertex performs some operation, its parent's local budget for that particular operation is decremented, as well as the global graph editing budget given to cluster the graph. We also note that we set a maximum number of splits to be allowed in the solution graph and every time a split is added, the splits' counter is incremented by 1. The algorithm can keep on branching on the splitting branch as long as the number of splits is still less than the maximum.

A simple branching algorithm to solve the Cluster Editing problem is to find a conflict triple $(u, v, w)$, of center $v$, and branch by either deleting one of the existing edges or adding the missing edge. However, our algorithm employs the utility of vertex splitting to have an additional branch on which the center vertex $v$ splits into two vertices $v_1$ and $v_2$ such that $N(v_1) \subset N(v) \setminus \{w\}$ and $N(v_2) \subset N(v) \setminus \{u\}$. The algorithm keeps on searching for conflict triples and branching on one of the four branches until the graph becomes free of conflict triples, and therefore a cluster graph. At each call for the branching algorithm, we first apply some preprocessing to the graph instance, part of it in the form of reduction rules. We will elaborate more on discussing these rules in the following sections. The pseudo-code of Algorithm 1 gives an overview about our multi-parameterized branching algorithm.

As can be seen in the above algorithm, the vertex splitting branch is considered, or explored, after all of the previous branches fail to contribute to the solution. This means that deleting any of the edges $uv$ and $vw$ will not lead to a solution, therefore we need to keep them in the graph. Moreover, adding the edge $uw$ will not lead to a solution as well, therefore we do not need it in the *solution graph*. More specifically, $v$ has to be in the two clusters with each of $u$ and $w$, but none of them can be in clusters with each other. In the vertex splitting operation shown in Figure 1, the mark

**Algorithm 1** Multi-Parameterized Cluster Editing with Vertex Splitting

```
function MPCEVS(G, A[], D[], S[], p, k)
    apply reduction rules
    pick a conflict triple (u, v, w)
    if no conflict triples in G then
        return Yes
    if uv is not permanent then
        D[parentOf(u)]−−;
        D[parentOf(v)]−−;
        if MPCEVS(G − uv, A[], D[], S[], p, k − 1) then
            return Yes
    set uv to permanent
    if vw is not permanent then
        D[parentOf(v)]−−;
        D[parentOf(w)]−−;
        if MPCEVS(G − vw, A[], D[], S[], p, k − 1) then
            return Yes
    set vw to permanent
    if uw is not fobidden then
        A[parentOf(u)]−−;
        A[parentOf(w)]−−;
        if MPCEVS(G + uw, A[], D[], S[], p, k − 1) then
            return Yes
    set uw to forbidden
    S[parentOf(v)]−−;
    split vertex v into v1 and v2
    if MPCEVS(G − v + v1 + v2, A[], D[], S[], p + 1, k − 1) then
        return Yes;
    return No;
end function
```

**P** denotes setting an edge to permanent, while the dashed line denotes setting an edge to forbidden. The edges $v_1v_2$, $uw$, $v_1w$, and $v_2u$ are set to forbidden. Otherwise, the rule of $v_1$ and $v_2$ belonging to separate clusters will be violated. Moreover, with the relaxation of overlapping clusters, splitting $v$ permits the clustering process to continue by allowing $v$ to join separate clusters with each of $u$ and $w$ via its children $v_1$ and $v_2$. We would like to note that splitting any of $u$ or $w$ will not help in resolving the conflict triple.

We present a semi-exact algorithm for the multi-parameterized approach that applies a heuristic preprocessing method to (try to) predict permanent and forbidden edges before branching in order to speed up the clustering process. We set a permanent edge between two vertices having the number of their common neighbours above a certain threshold, and a forbidden edge between two vertices having the number of their common neighbours less than a certain edge.

We also present a greedy heuristic algorithm, that does not apply a branching procedure. Instead, it keeps on searching for conflict triples and resolving them instantly by applying one of the four possible editing operations. In case vertex splitting is allowed, the heuristic applies our thresholding procedure to decide whether the conflict triple should be resolved by splitting or not. Otherwise, the algorithm applies greedily the operation that has the least cost. We explain the strategy of determining the cost of an editing operation in the following section. The algorithm keeps on iterating until the graph becomes free of conflict triples.

## 3.3 Cost of an Edge Editing Operation

In our work, we introduce a greedy heuristic algorithm that resolves conflict triples instantly by splitting vertices or applying the cheapest edge editing operation possible. We thus employ a notion of costs of editing operations inspired by the data polishing idea introduced in [35]. We consider the costs of editing operations as follows:

- the cost of adding an edge $uv$ is the number of operations needed to connect the

neighbours of each of $u$ and $v$ with edges

- the cost of deleting an edge $uv$ is the number of operations needed to disconnect the common neighbours of $u$ and $v$

## 3.4 Reduction Rules

In every call to our branching algorithm, we apply some preprocessing operations in the form of reduction rules. These reduction rules give us a reduced instance of the input graph that is equivalent to it. By equivalent we mean that the decision result of the branching algorithm on this reduced instance will be the same as the decision result on the original instance. In this context, we employ some reduction rules influenced by the rules introduced in [2]. We note that the parameter $s$ in [2] denotes the minimum acceptable cluster size. However, in our model, $s$ corresponds to the maximum allowable splitting operations that a single vertex can perform. We denote by $a_v$, $d_v$ and $s_v$ the remaining budgets for adding an edge, deleting an edge, or splitting for an arbitrary vertex $v$.

***Reduction rule 1:*** *If any of k, $a_v$, $d_v$, or $s_v$ drops below zero for any vertex v, then halt and report a no-instance.*

***Reduction rule 2:*** *If $d_v = 0$, for any vertex v, then set all the edges between v and its neighbours to permanent.*

***Reduction rule 3:*** *If $a_u = 0$, for any vertex v, then set all the edges between v and the vertices in $V(G) \setminus N(u)$ to forbidden.*

***Reduction rule 4:*** *If uv and uw are permanent edges and vw is forbidden then split u to resolve the conflict triple.*

## 3.5 Preprocessing Rules

As we mentioned previously, we perform preprocessing operations at the beginning of each branching step in our algorithm. In this section we introduce two preprocessing methods that we apply in addition to the ones of the previous section. One of these methods is used to monitor the number of splits in the graph and make sure that it does not exceed the maximum allowable number, while the other method explains the heuristic approach that we use to speed up the cluster editing process by trying to predict with edges should be set to permanent and which could be forbidden. The two preprocessing procedures are the following.

**1. *Check Splits Count:*** *The reduction algorithm terminates and reports a no-instance whenever the number of the splits exceeds the maximum number allowed.*

This preprocessing method is applied in order to make sure that the graph will not contain more splits than the maximum allowed number.

**2. *Find Edge States:***

For every pair of vertices $u$ and $v$:

- if $\frac{common\_neighbours(u,v)}{N(u)} \geq$ P_Th and $\frac{common\_neighbours(u,v)}{N(v)} \geq$ P_Th then set $uv$ to $permanent$
- if $\frac{common\_neighbours(u,v)}{N(u)} \leq$ F_Th and $\frac{common\_neighbours(u,v)}{N(v)} \leq$ F_Th then set $uv$ to $forbidden$

This preprocessing method tries to find some potential permanent and forbidden edges at an early stage in the branching process using two predetermined thresholds, namely P_Th and F_Th. These two thresholds are used to measure the similarity of two arbitrary vertices. If each of the ratios of the number of common neighbours between $u$ and $v$ to the number of neighbours of each of $u$ and $v$ is greater than P_Th, then $u$ and

*v* are considered *too-close to be in different clusters* (or similar) and the edge between them should be set to permanent. On the other hand, if the value of each of these ratios is less than F_Th, then *u* and *v* are considered dissimilar and the edge between them ought to be set to forbidden. This process of marking edges early as permanent or forbidden helps in speeding up the clustering process because it helps pruning the search tree at early stages, resulting in a faster cluster editing.

# Chapter Four

# A Neighborhood Splitting Heuristic Algorithm

## 4.1 Neighborhood Uncertainty for Splits

In our input graph, we know that each vertex $v$ is connected to a certain set of vertices ($N(v)$), which could be an empty set. Our algorithms employ vertex splitting as an additional branch in order to resolve conflict triples. A vertex $v$ splits into two vertices $v_1$ and $v_2$ due to a conflict triple $(u, v, w)$ that was not resolved by performing any of the other edge editing operations. As a result, $v_1$ can be put in a cluster with $u$ but not with $w$ and $v_2$ can be put in a cluster with $w$ but not with $u$. We consider $v$ to be the parent vertex, and $v_1$ and $v_2$ as its children. At this stage, we do not have any information about the neighborhood of these two newly added vertices (splits). The only information we have about the splits' neighborhoods is that the two pairs $\{u, v_1\}$ and $\{w, v_2\}$ must be in (exactly) two disjoint cliques. This raises the following question: which vertices should also belong to the neighborhoods of $v_1$ and $v_2$? In other words: how to partition $N(v)$ into exactly two subsets that form $N(v_1)$ and $N(v_2)$?.

A naive solution is to consider all of the remaining vertices from the neighborhood of the parent to belong to the neighborhood of each of its splits, and postpone the decision until the neighborhood becomes a disjoint union of cliques. If this approach

is adopted, and $v$ is to be split due to a conflict triple $(u, v, w)$, the neighborhoods of the splits would be as follows: $N(v_1) = N(v) \setminus \{w\}$ and $N(v_2) = N(v) \setminus \{u\}$, as shown in Figure 2. Unfortunately, this simple solution may cause the splits to have edges with vertices that should not belong to their neighborhoods. Therefore, a split might need a smaller subset of its parent's neighborhood. Moreover, this naive approach will make the search space for the branching algorithm wider due to the unwanted vertices in $N(v_1)$ and $N(v_2)$. As a consequence, clustering will require additional budget for the graph editing operations, and obviously more time.

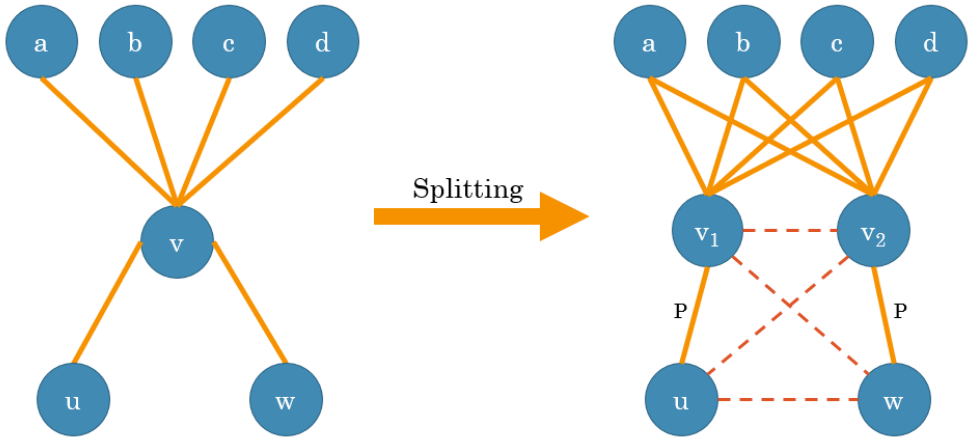


Figure 2: Naive Splitting Operation.

## 4.2 A Neighborhood Splitting Method

In order to solve the challenge of the neighborhoods' uncertainty for the splits and partitioning the neighborhood of a splitting vertex in a better way than that of the naive approach, we propose the *neighborhood splitting method*. This method tends to resolve conflict triples by splitting vertices while trying to keep vertices that have a high potential to belong to the same cluster connected to each other, thus achieving high intra-cluster similarity. In the following, we explain the steps taken by the *neighborhood splitting method*. Consider the example in Figure 3 as a subgraph of an input graph for the clustering problem. Consider in this subgraph, the conflict triple $(u, v, w)$

that should be resolved by applying one of the four possible graph editing operations. Suppose that $(u, v, w)$ could not be resolved by applying any of the edge editing operations and that we are only left with splitting $v$. This implies that $v$ could join separate clusters with each of $u$ and $w$, and that $u$ and $w$ cannot belong to the same cluster.

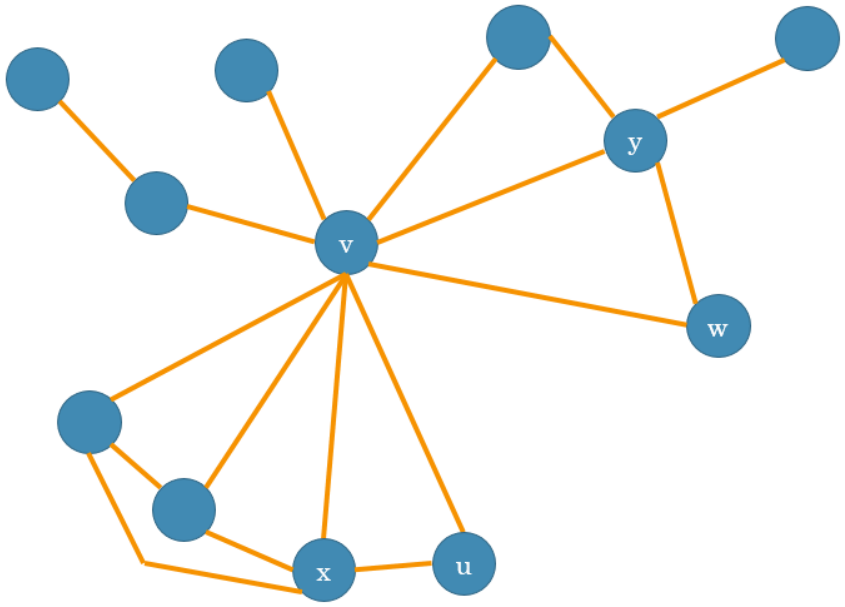


Figure 3: Neighborhood Splitting Method: An Example.

The *neighborhood splitting method* computes the induced subgraph $G[N(v)]$, as shown in *Figure 4*, that contains only the neighbours of $v$ and the edges between them. The method then computes the connected components in $G[N(v)]$. Let $C_u$ and $C_w$ be the connected components to which $u$ and $w$ belong in $G[N(v)]$, respectively. Assume that $G[N(v)]$ has more than one connected component and that $C_u$ and $C_w$ are two separate connected components. We pick among them the component with more vertices. In case they have an equal number of vertices, we pick anyone of them. For our example, Figure 4 shows that $C_u$ is the connected component with the greater number of vertices. As a result, one of the splits will have the vertices of $C_u$ as its neighborhood, while the other split will have the vertices of $N(v) \setminus C_u$ as its neighborhood. Figure 5 shows the resulting subgraph after splitting $v$ into $v_1$ and $v_2$ with $N(v_1) = C_u$ and $N(v_2) = N(v) \setminus C_u$.

On the other hand, consider the case in which $u$ and $w$ belong to the same connected

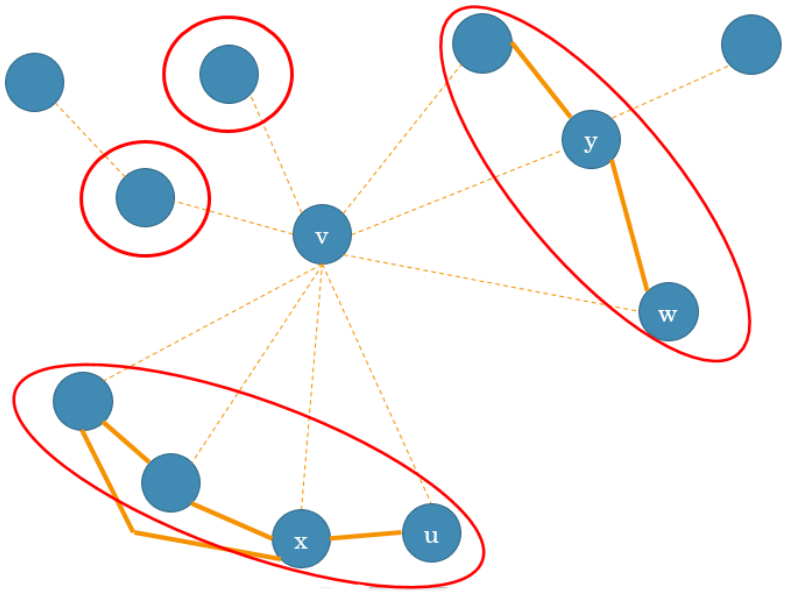


Figure 4: Connected Components of $G[N(v)]$ in the example of Figure 3.

component, $C_{uw}$ (which may be the only connected component in $G[N(v)]$), as illustrated in Figure 6. We pick among $u$ and $w$ the vertex having more neighbours in $C_{uw}$. In case they have an equal number of neighbours in $C_{uw}$, we pick anyone of them. For the example in Figure 6, $u$ is the vertex with more neighbours in $C_{uw}$. As a result, one of the splits will have as its neighborhood $u$ and its neighbours in $C_{uw}$, while the other split will have the remaining vertices from $N(v)$ as its neighborhood. Figure 7 shows the resulting subgraph after splitting $v$ of the example in Figure 6.

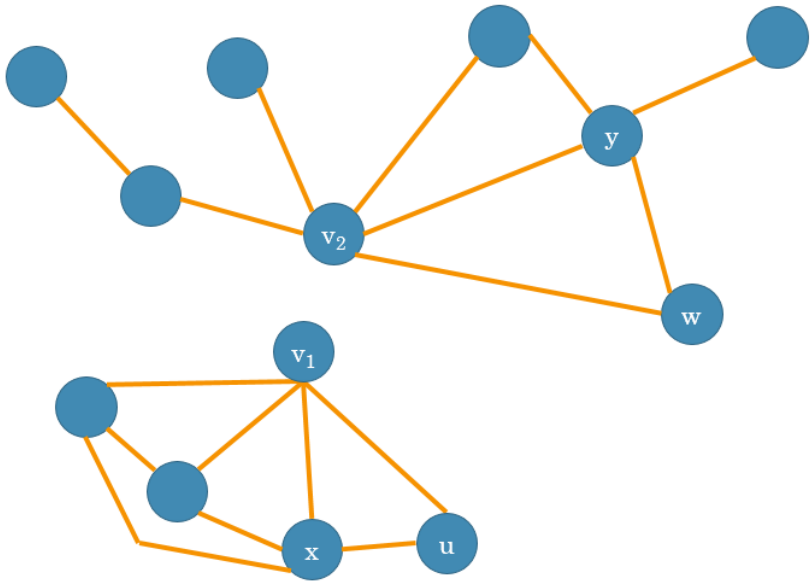


Figure 5: Neighborhood Splitting of $v$ in the example of Figure 3.

The splitting branch is taken after all of the edge editing operations fail to resolve the conflict triple $(u, v, w)$. Thus, $uv$ and $vw$ cannot be deleted, and $vw$ cannot be added. Consequently, $v$ is split into two vertices where each can join a cluster with one of $u$ and $w$. The *neighborhood splitting method* splits $v$ into $v_1$ and $v_2$ taking into consideration the issue of the neighborhood uncertainty of the splits. It associates as a neighborhood to $v_1$ the vertex with the larger neighborhood in $G[N(v)]$, among $u$ and $w$, and its neighbours. In this way, it tends to enable these vertices of forming a cluster together. On the other hand, it associates as a neighborhood to $v_2$ the remaining vertices from the neighborhood of $v$ tending to preserve the connections between $v$ and its neighbours. The pseudocode of this algorithm is presented below in Algorithm 2.

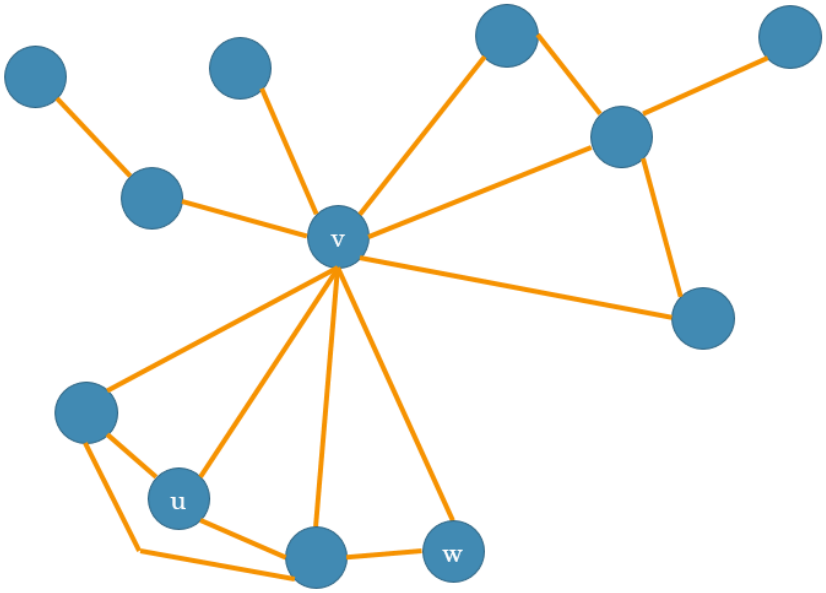


Figure 6: Case where $u$ and $w$ belong to the same connected component in $G[N(v)]$.

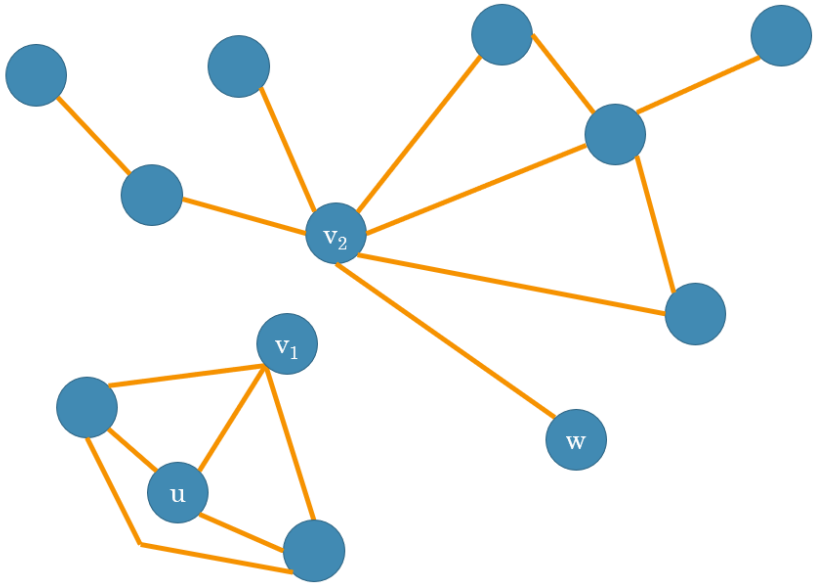


Figure 7: Neighborhood Splitting of $v$ in the example of Figure 6

**Algorithm 2** The Neighborhood Splitting Method

**function** NEIGHBORHOODSPLITTING($G$, $u$, $v$, $w$)
  add vertices $v_1$ and $v_2$
  set $v_1u$ and $v_2w$ to permanent
  set $v_1v_2$ and $uw$ to forbidden
  set $v_1w$ and $v_2u$ to forbidden
  $G' \longleftarrow G[N(v)]$
  $C_u \longleftarrow$ the connected component of $G'$ that contains $u$
  $C_w \longleftarrow$ the connected component of $G'$ that contains $w$
  **if** $C_u == C_w$ **then**
    //$C_u$ and $C_w$ are the same connected component
    $N_u \longleftarrow u$ and its neighbours in $C_u$
    $N_w \longleftarrow w$ and its neighbours in $C_w$
    $N \longleftarrow$ the set having more vertices between $N_u$ and $N_w$
    $N(v_1) = N$
    $N(v_2) = N(v) \setminus N$
    **else**
    //$C_u$ and $C_w$ are different components
    $C \longleftarrow$ the connected component having more vertices between $C_u$ and $C_w$
    $N(v_1) = C$
    $N(v_2) = N(v) \setminus$C
  delete $v$ from $G$
**end function**

# Chapter Five

# Our Algorithms

In this chapter, we present our semi-exact algorithm for multi-parameterized cluster editing with vertex splitting, *MPCEVS_SE*, and our greedy heuristic for the cluster editing problem also allowing vertex splitting, *GHCEVS*. We also present a semi-exact algorithm for multi-parameterized cluster editing, but not allowing vertex splitting, based on the model represented in [2]. We denote this semi-exact algorithm by *MPCE_SE* and we use it for comparison purposes to compare its results with those of our algorithm *MPCEVS_SE*.

## 5.1 Multi-Parameterized Cluster Editing with Vertex Splitting: A Semi-Exact Algorithm

In this section, we introduce our semi-exact algorithm for the multi-parameterized version of the problem allowing vertex splitting. The algorithm takes as input the graph to be clustered, the remaining budgets for edge additions, edge deletions, and vertex splittings corresponding to each parent vertex, and the maximum allowable number of editing operations $k$. In addition, it takes the maximum number of splits, $p$, allowed to be in the solution graph, and the values of the thresholds $P\_Th$ and $F\_Th$ used by the preprocessing method "$findEdgeStates$" to find permanent and forbidden edges. The psuedocode is shown in Algorithm 3.

**Algorithm 3** Semi-Exact Multi-Parameterized Cluster Editing with Vertex Splitting

```
function MPCEVS_SE(G, A[], D[], S[], p, k, P_TH, F_TH)
    checkSplitsCount(p)
    findEdgeStates(P_TH, F_TH)
    apply reduction rules
    pick a conflict triple (u, v, w)
    if no conflict triples in G then
        return Yes
    if uv is not permanent then
        D[parentOf(u)]−−;
        D[parentOf(v)]−−;
        if MPCEVS_SE(G − uv, A[], D[], S[], p, k − 1) then
            return Yes
    set uv to permanent
    if vw is not permanent then
        D[parentOf(v)]−−;
        D[parentOf(w)]−−;
        if MPCEVS_SE(G − vw, A[], D[], S[], p, k − 1) then
            return Yes
    set vw to permanent
    if uw is not fobidden then
        A[parentOf(u)]−−;
        A[parentOf(w)]−−;
        if MPCEVS_SE(G + uw, A[], D[], S[], p, k − 1) then
            return Yes
    set uw to forbidden
    S[parentOf(v)]−−;
    NEIGHBORHOODSPLITTING(G, u, v, w)
    if MPCEVS_SE(G − v + v1 + v2, A[], D[], S[], p + 1, k − 1) then
        return Yes;
    return No;
end function
```

The algorithm is a recursive branching algorithm. At each branch, we apply the reduction and preprocessing methods that were discussed in the previous chapter. We keep on searching for conflict triples $(u, v, w)$ in the graph to resolve them using one of the four possible editing operations. At first, we try to delete one of the existing edges ($uv$ and $vw$) and decrement $k$ and the deletion budgets of the involved vertices; if it does not work, we set these edges to permanent as they must be part of the solution. Then, we try to add the missing edge $uw$ and decremenet $k$ and the addition budget of the involved vertices; if it does not work, we set the edge to forbidden as it can not be part of the solution. Consequently, after none of the previous operations works, we try splitting vertex $v$ into two vertices $v_1$ and $v_2$, decrement $k$ and the splitting budget of $v$, and increment the counter of the number of splits in the graph. The algorithm keeps on running until we get a cluster graph or a no-instance.

## 5.2 A Greedy Cluster Editing Heuristic Algorithm

Given a graph $G$, our algorithm performs a number of editing operations: adding an edge, deleting an edge and splitting a vertex. Each operation is performed based on some likelihood measure. For example, an edge is added to connect two vertices $u$ and $v$ if they (simply) share "many" common neighbours. To explicate:

For every pair of vertices $u$ and $v$:

- if $\frac{common\_neighbours(u,v)}{N(u)} \geq$ P_Th and $\frac{common\_neighbours(u,v)}{N(v)} \geq$ P_Th then set $uv$ as $permanent$
- if $\frac{common\_neighbours(u,v)}{N(u)} \leq$ F_Th and $\frac{common\_neighbours(u,v)}{N(v)} \leq$ F_Th then set $uv$ as $forbidden$

The algorithm keeps on finding conflict triples and resolving them by applying one of the four possible operations. A pseudo-code is shown below, in Algorithm 4. We assume that in addition to the input graph $G$, the algorithm takes the following as input: a boolean value $allows\_splitting$ that determines if the algorithm is allowing

vertex splitting or not, and two thresholds $P\_Th$ and $F\_Th$ that are used to determine potential permanent and forbidden edges based on the above rules.

If a conflict triple should be resolved by splitting a vertex $v$ that is aprt of a conflict triple $(u,v,w)$, then our algorithm does so by splitting it into $v_1$ and $v_2$, using the *neighborhood splitting method* introduced in Chapter IV, each of which can form a separate cluster with $u$ or $w$. Splitting $v$ should occur if $uv$ and $vw$ are considered as potential permanent edges and $uw$ is considered as a potential forbidden edge by the thresholding technique shown above. Otherwise, if the algorithm is not allowing vertex splitting or the conflict triple should not be resolved by splitting $v$, we perform the edge editing operation with the lowest cost. We employ the notion of costs of editing operations introduced in Chapter III. We keep on resolving conflict triples using this strategy until the graph becomes free of conflict triples and thus a cluster graph.

**Algorithm 4** Greedy Heuristic for Cluster Editing with Vertex Splitting

```
function GHCEVS(G, allows_splitting, P_Th, F_Th)
    do
        find a conflict triple (u,v,w)
        if allows_splitting AND (u,v,w) is a splitting conflict triple then
            split v into v1 and v2
        else
            if adding uw has the lowest cost then
                add uw
            else
                if deleting uv has the lowest cost then
                    delete uv
                else
                    delete vw
                end
            end
        end
    while there are conflict triples
end function
```

## 5.3 A Semi-Exact Algorithm for Multi-Parameterized Cluster Editing

In this section, we present a semi-exact algorithm for a multi-parameterized cluster editing approach that does not allow vertex splitting, and we denote it by *MPCE_SE*. It is based on the exact algorithm presented in [2]. The only difference is that we apply the preprocessing method *"findEdgeStates"* at each branching step in order to speed up the procedure. We use this algorithm for comparison purposes in our experiments to show the effectiveness of splitting by comparing its results to the results of our semi-exact algorithm *MPCEVS_SE* (which allows splitting). A pseudo-code is presented in Algorithm 5.

**Algorithm 5** Multi-Parameterized Cluster Editing Semi-Exact Algorithm

```
function MPCE_SE(G, A[], D[], k, P_Th, F_Th)
    findEdgeStates(P_TH, F_TH)
    apply reduction rules
    pick a conflict triple (u,v,w)
    if no conflict triples in G then
        return Yes
    if uv is not permanent then
        D[parentOf(u)]−−;
        D[parentOf(v)]−−;
        if MPCE_SE(G − uv, A[], D[], k − 1) then
            return Yes
    set uv to permanent
    if vw is not permanent then
        D[parentOf(v)]−−;
        D[parentOf(w)]−−;
        if MPCE_SE(G − vw, A[], D[], k − 1) then
            return Yes
    set vw to permanent
    if uw is not fobidden then
        A[parentOf(u)]−−;
        A[parentOf(w)]−−;
        if MPCE_SE(G + uw, A[], D[], k − 1) then
            return Yes
    return No;
end function
```

The algorithm takes as input the graph to be clustered, the remaining budgets for the

edge editing operations corresponding to each vertex, the maximum allowable number of editing operations $k$, and the values of the thresholds $P\_Th$ and $F\_Th$ used by *findEdgeStates* to find permanent and forbidden edges. The algorithm keeps on finding conflict triples and resolving them by recursive branching on one of the three possible branches while decrementing the number of operations and editing budgets corresponding to each vertex.

# Chapter Six

# Experimental Analysis

In this chapter we discuss our experimental analysis based on the results that we obtained from running our algorithms on various real and synthetic graphs. The algorithms were implemented in Java. The aim of our experiments was to measure the performance of our clustering algorithms and to measure the effectiveness of the splitting operation. We thus compared the results of our algorithms with the results obtained from not allowing splitting and the KMeans clustering algorithm.

For synthetic graphs generation, we implemented a graph generator that outputs a graph and determines the values of the editing budgets needed to transform it into a cluster graph. The generator takes as input the number of vertices and clusters needed in the graph to be generated. It randomly assigns each vertex to a cluster and adds edges between vertices that are chosen to be in the same cluster. The graph will hence be a cluster graph. After that, the generator performs $k$ editing operations from addition, deletion, and splitting. Therefore, we will know the minimum number of operations needed to transform the graph into a cluster graph, in addition to the maximum number of edge additions, deletions, and vertex splittings that can be performed at a single vertex. This procedure makes the ground truth of the generated graphs known.

In order to evaluate the clustering results, we used the "within-cluster average distance" metric, denoted by *WCAD*. This metric measures intra-cluster similarity as it computes the average distance between every pair of vertices belonging to the same cluster. We

consider the number of common neighbours between two vertices as a measure of how similar they are. Equivalently, we consider, as a distance between two vertices, the total number of vertices in the graph minus the number of common neighbours between these vertices. In addition, we used the *FBCubed* metric [8] that is known to be used for evaluating clustering with overlaps [16]. The *FBCubed* metric gives a real value belonging to the interval [0, 1]. The closer the value to 1, the better the clustering is, while the closer the value to 0 the worse the clustering is, corresponding to the ground truth [16]. A value of 1 represents the best or optimal clustering.

In order to test the performance of our algorithms and the effectiveness of vertex splitting in clustering, we ran several experiments on graphs with various sizes and compared the results of allowing splitting to those of not allowing it. We notice that vertex splitting gives a better clustering and higher intra-cluster similarity. We represent in the following, the characteristics of the tested graphs and the values of the considered measures. For each column in the following tables, we ran the corresponding algorithm on 20 graphs and reported their average values. The values "time" and "$k$" represent the average time and number of operations that were needed to cluster the graph. The values "WCAD" and "FBCubed" represent the average values of the "within-cluster average distance" and FBCubed metric for each solution graph. In addition, "Splits" and "Splits Range" show the average number of splits in the resulting graph and the range to which this number belongs. As for the values of the thresholds, we set $P\_Th$ to 0.75 and $F\_Th$ to 0.25 in all the experiments.

## 6.1 Experiment 1: Performance of the Semi-Exact Algorithm on Synthetic Data

In this experiment we ran synthetic graphs of orders ranging between 50 and 150 with the optimal value of $k$ set to 0.30|E|. The results of the clustering measures are reported in Table 1. We notice that the value of the FBCubed metric is always higher (closer to 1) when vertex splitting is allowed, which means that allowing vertex splitting results

| Graph Order | With Splitting | | | | Without Splitting | |
|---|---|---|---|---|---|---|
| | Splits Range | Splits | WCAD | FBCubed | WCAD | FBCubed |
| 50-75 | 2-7 | 5.1 | 57.81 | 0.95 | 59.91 | 0.89 |
| 75-100 | 5-11 | 8.43 | 82.69 | 0.98 | 85.79 | 0.91 |
| 100-125 | 8-13 | 10.54 | 104.15 | 0.96 | 107.72 | 0.90 |
| 125-150 | 8-16 | 12.28 | 132.12 | 0.96 | 134.62 | 0.87 |

Table 1: Experiment 1. Clustering Results

| Graph Order | With Splitting | | Without Splitting | |
|---|---|---|---|---|
| | Time | k | Time | k |
| 50-75 | 14.61 sec | 51.3 | 2.53 sec | 65.6 |
| 75-100 | 1.11 min | 64.7 | 6.17 sec | 82.9 |
| 100-125 | 12.84 min | 96 | 2.60 min | 115.86 |
| 125-150 | 26.84 min | 107.9 | 4.23 min | 129.54 |

Table 2: Experiment 1. Consumed Budgets

in more accurate clustering. Also, we notice that the value of the WCAD metric is always less when vertex splitting is allowed, which means that we had less distances between elements of the same clusters. In addition, we show that allowing vertex splitting revealed significant numbers of splits in the solution graphs which shows that a significant number of vertices can belong to multiple clusters.

Moreover, Table 2 shows the values of the consumed budgets to cluster the graphs. It is clear that clustering with vertex splitting was taking less values of $k$ but more time and that the difference in the running times between allowing and not allowing splitting increases significantly as the graph order increases. This is due to the additional splitting branch of the exponential-time branching algorithm.

## 6.2 Experiment 2: Performance of the Greedy Heuristic on Synthetic Data

The aim of this experiment was to evaluate the performance of our greedy algorithm on synthetic graphs having 100 to 1000 vertices and with an optimal value of $k$ set to $0.30|E|$. Table 3 shows the clustering results that were obtained from running the

| Graph Order | With Splitting | | | | Without Splitting | |
|---|---|---|---|---|---|---|
| | Splits Range | Splits | WCAD | FBCubed | WCAD | FBCubed |
| 100-250 | 7-17 | 10.2 | 178.48 | 0.95 | 183.14 | 0.87 |
| 250-500 | 10-48 | 23.66 | 357.33 | 0.95 | 359.19 | 0.87 |
| 500-750 | 25-66 | 40.2 | 615.63 | 0.93 | 617.52 | 0.86 |
| 750-1000 | 32-89 | 67.23 | 806.55 | 0.94 | 808.23 | 0.87 |

Table 3: Experiment 2. Clustering Results

| Graph Order | With Splitting | | Without Splitting | |
|---|---|---|---|---|
| | Time | k | Time | k |
| 100-250 | 6.19 sec | 296.2 | 5.37 sec | 384 |
| 250-500 | 1.72 min | 658.7 | 1.17 min | 882.6 |
| 500-750 | 12.99 min | 1308.7 | 8.98 min | 1515.8 |
| 750-1000 | 17.91 min | 1643.16 | 13 min | 2239 |

Table 4: Experiment 2. Consumed Budgets

experiments. Obviously, allowing vertex splitting resulted in more accurate clustering due to the higher values of the FBCubed metric, and less distances between elements of the same clusters due to lower values of the WCAD metric. In addition, the solution graphs had notable numbers of splits, which uncover information about vertices that should belong to more than one clusters.

Table 4 shows that clustering with allowing vertex splitting took less values of $k$ but, as expected, more time to cluster the graphs and that the values increased as the number of vertices increased.

## 6.3 Experiment 3: Performance of the Greedy Heurstic on Protein Similarity Data

In this experiment we employed our greedy algorithm to cluster real graphs derived from the COG protein similarity data [31] [11]. The COG graphs are weighted graphs where weights represent similarities between protein data. Since our model applies to unweighted graphs, we ran experiments on similarity graphs obtained from the COG graphs by choosing a similarity threshold and adding (unweighted) edges between

| Graph Order | With Splitting | | | Without Splitting |
|---|---|---|---|---|
| | Splits Range | Splits | WCAD | WCAD |
| 75-100 | 6-13 | 9 | 75.14 | 77.27 |
| 100-150 | 8-21 | 12.28 | 114.59 | 116.71 |
| 150-200 | 11-25 | 16.55 | 159.47 | 162.91 |
| 200-250 | 16-22 | 17.44 | 217.19 | 219.23 |
| 250-300 | 9-26 | 17.25 | 268.78 | 271.16 |
| 300-350 | 9-36 | 19.33 | 320.67 | 323.48 |

Table 5: Experiment 3. Clustering Results

| Graph Order | With Splitting | | Without Splitting | |
|---|---|---|---|---|
| | Time | k | Time | k |
| 75-100 | 4.42 sec | 764.4 | 4.35 sec | 756.5 |
| 100-150 | 8.11 sec | 775.4 | 9.04 sec | 761.8 |
| 150-200 | 10.96 sec | 1306.3 | 12.56 sec | 1280.6 |
| 200-250 | 24.57 sec | 1907.77 | 24.78 sec | 1881.77 |
| 250-300 | 41.22 sec | 2490.5 | 51.73 sec | 2473.4 |
| 300-350 | 100.64 sec | 3066.8 | 88.34 sec | 3069.4 |

Table 6: Experiment 3. Consumed Budgets

vertices that have their similarity greater than the chosen threshold.

The clustering results reported in Table 5 show that higher intra-cluster similarity was obtained by allowing vertex splitting. This can be seen in the lower values of the WCAD metric that shows less distances between elements of the same clusters. In addition, vertex splitting disclosed remarkable values of splits in the solution graphs, which shows that some proteins can belong to multiple clusters with other protein structures.

Table 6 reports the number of operations and running times consumed to cluster the graphs. We can observe that allowing vertex splitting was taking higher values of $k$ or time on some data, and less values on others. Therefore, there is no clear advantage for the non-splitting variant when it comes to efficiency. Yet, the effectiveness of splitting was obvious, as reported in Table 5.

| Graph Order | MPCEVS_SE | | KMeans | |
|---|---|---|---|---|
| | FBCubed | Time | FBCubed | Time |
| 50-75 | 0.95 | 14.61 sec | 0.81 | 0.37 sec |
| 75-100 | 0.98 | 1.11 min | 0.79 | 0.44 sec |
| 100-125 | 0.96 | 12.84 min | 0.75 | 0.66 sec |
| 125-150 | 0.96 | 26.84 min | 0.76 | 1.21 sec |

Table 7: Experiment 4. Comparing MPCEVS_SE with KMeans Clustering

| Graph Order | GHCEVS | | KMeans | |
|---|---|---|---|---|
| | FBCubed | Time | FBCubed | Time |
| 100-250 | 0.95 | 6.19 sec | 0.82 | 2.42 sec |
| 250-500 | 0.95 | 1.72 min | 0.80 | 11.96 sec |
| 500-750 | 0.93 | 12.99 min | 0.77 | 50.26 sec |
| 750-1000 | 0.94 | 17.91 min | 0.78 | 2.48 min |

Table 8: Experiment 4. Comparing GHCEVS with KMeans Clustering

## 6.4 Experiment 4: Comparison with KMeans Clustering

The objective of this experiment was to compare our clustering algorithms to the well known *KMeans* clustering algorithm. Table 7 shows a comparison between our semi-exact algorithm *MPCEVS_SE* and *KMeans* on the FBCubed metric and the running time. Table 8 compares our greedy heuristic to *KMeans* on the same measures. The results show that our algorithms take more time but yield much better clustering due to the higher values of the FBCubed metric. This gives further evidence of the effectiveness of vertex splitting in practice.

# Chapter Seven

# Conclusion and Future Work

In this thesis, we tackled the problem of cluster editing in graphs. The problem models unsupervised correlation clustering that is used in many fields such as machine learning and data mining. The operations allowed in cluster editing are usually addition and deletion of edges. Vertices are limited each to belong to only a single cluster. In our work, we allow an additional operation: vertex splitting. A vertex is allowed to split into more than one version (or copy) and thus allowed to belong to more than one cluster. This operation was proposed in [6] as a generalized version of the problem with a main objective to allow overlapping clusters.

We addressed the difficulty of how to split the neighborhood of a vertex once it is known to be a split-vertex, and presented the *neighborhood splitting method* to solve this problem. Moreover, we presented a multi-parameterized heuristic algorithm that allows each vertex to split into more than one version and bounds the global number of allowed editing operations, global number of vertex splitting, the local numbers of edge editing operations and vertex splittings allowed (per vertex). In addition, we presented a greedy heuristic that allows splitting and resolves conflict triples instantly instead of employing a recursive branching technique.

In our experiments, we compared clustering with vertex splitting to clustering without vertex splitting. We also compared our algorithms with the well known KMeans clustering algorithm that allows each vertex to belong to only one cluster. The results

showed that clustering with vertex splitting may take additional time, but yield better clustering results and higher intra-cluster similarity. In addition, splitting gives insights about data elements that should belong to more than one cluster, or can play roles in more than one group. This information can be easily lost when each data element is allowed to belong to only one cluster. Moreover, we note a very important outcome from our experiments that shows that allowing vertex splitting can decrease the number of operations needed to cluster the graph.

Future work on the clustering with overlaps problem can potentially make good use of crown decomposition (introduced in [7] and [18]) and corresponding parameterized kernelization algorithms [20]. Moreover, the recent work on dynamic parameterized problems can be possibly applied to our parameterized clustering with overlaps problem (see [5, 30]) and a plausible future direction is to try the turbo-charging method to enhance heuristic techniques [4], such as the greedy heuristic approach presented in this thesis. In addition, future work can consider other methods to split the neighbourhood of a splitting vertex. Finally, it remains open whether the Cluster Editing with Overlaps problem can be solved by a fixed-parameter algorithm with a running time in $\mathcal{O}(c^k)$, for some small constant $c$.

# Bibliography


[1] M. Aamir and S. M. A. Zaidi. Clustering based semi-supervised machine learning for ddos attack classification. *Journal of King Saud University-Computer and Information Sciences*, 2019.

[2] F. N. Abu-Khzam. On the complexity of multi-parameterized cluster editing. *Journal of Discrete Algorithms*, 45:26–34, 2017.

[3] F. N. Abu-Khzam, N. E. Baldwin, M. A. Langston, and N. F. Samatova. On the relative efficiency of maximal clique enumeration algorithms, with application to high-throughput computational biology. 2005.

[4] F. N. Abu-Khzam, S. Cai, J. Egan, P. Shaw, and K. Wang. Turbo-charging dominating set with an FPT subroutine: Further improvements and experimental analysis. In T. V. Gopal, G. Jäger, and S. Steila, editors, *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, volume 10185 of *Lecture Notes in Computer Science*, pages 59–70, 2017.

[5] F. N. Abu-Khzam, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. On the parameterized complexity of dynamic problems. *Theor. Comput. Sci.*, 607:426–434, 2015.

[6] F. N. Abu-Khzam, J. Egan, S. Gaspers, A. Shaw, and P. Shaw. Cluster editing with vertex splitting. In J. Lee, G. Rinaldi, and A. R. Mahjoub, editors, *Combinatorial Optimization - 5th International Symposium, ISCO 2018, Marrakesh, Morocco,*

*April 11-13, 2018, Revised Selected Papers*, volume 10856 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2018.

[7] F. N. Abu-Khzam, M. A. Langston, and W. H. Suters. Fast, effective vertex cover kernelization: a tale of two algorithms. In *The 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.*, page 16. IEEE, 2005.

[8] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information retrieval*, 12(4):461–486, 2009.

[9] D. M. S. Bhatti, S. Ahmed, A. S. Chan, and K. Saleem. Clustering formation in cognitive radio networks using machine learning. *AEU-International Journal of Electronics and Communications*, 114:152994, 2020.

[10] S. Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

[11] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. TRUB. A fixed-parameter approach for weighted cluster editing. In *Proceedings Of The 6th Asia-Pacific Bioinformatics Conference*, pages 211–220. World Scientific, 2008.

[12] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.

[13] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[14] G. M. Borkar, L. H. Patil, D. Dalgade, and A. Hutke. A novel clustering approach and adaptive svm classifier for intrusion detection in wsn: a data mining concept. *Sustainable Computing: Informatics and Systems*, 23:120–135, 2019.

[15] P. Braun, A. Cuzzocrea, T. D. Keding, C. K. Leung, A. G. Padzor, and D. Sayson. Game data mining: clustering and visualization of online game data in cyber-physical worlds. *Procedia Computer Science*, 112:2259–2268, 2017.

[16] G. O. Chagas, L. A. N. Lorena, and R. D. C. dos Santos. A hybrid heuristic for the overlapping cluster editing problem. *Applied Soft Computing*, 81:105482, 2019.

[17] J. Chen and J. Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

[18] B. Chor, M. Fellows, and D. Juedes. Linear kernels in linear time, or how to save k colors in o (n 2) steps. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 257–269. Springer, 2004.

[19] R. G. Downey and M. R. Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

[20] M. Fellows, M. Langston, F. Rosamond, and P. Shaw. Efficient parameterized preprocessing for cluster editing. In *International Symposium on Fundamentals of Computation Theory*, pages 312–321. Springer, 2007.

[21] M. R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *International Workshop on Parameterized and Exact Computation*, pages 276–277. Springer, 2006.

[22] M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2 – 17, 2011. Parameterized Complexity of Discrete Optimization.

[23] K.-i. Fukui, Y. Okada, K. Satoh, and M. Numao. Cluster sequence mining from event sequence data and its application to damage correlation analysis. *Knowledge-Based Systems*, 179:136–144, 2019.

[24] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.

[25] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

[26] J. Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009.

[27] G. Kikugawa, Y. Nishimura, K. Shimoyama, T. Ohara, T. Okabe, and F. S. Ohuchi. Data analysis of multi-dimensional thermophysical properties of liquid substances based on clustering approach of machine learning. *Chemical Physics Letters*, 728:109–114, 2019.

[28] M. Křivánek and J. Morávek. Np-hard problems in hierarchical-tree clustering. *Acta informatica*, 23(3):311–323, 1986.

[29] D. Lokshtanov and D. Marx. Clustering with local restrictions. *Information and Computation*, 222:278–292, 2013.

[30] J. Luo, H. Molter, A. Nichterlein, and R. Niedermeier. Parameterized dynamic cluster editing. In S. Ganguly and P. K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 46:1–46:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[31] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truss, and S. Böcker. Exact and heuristic algorithms for weighted cluster editing. In *Computational Systems Bioinformatics: (Volume 6)*, pages 391–401. World Scientific, 2007.

[32] J. P. Sarkar, I. Saha, S. Chakraborty, and U. Maulik. Machine learning integrated credibilistic semi supervised clustering for categorical data. *Applied Soft Computing*, 86:105871, 2020.

[33] Y. Sato, K. Izui, T. Yamada, and S. Nishiwaki. Data mining based on clustering and association rule analysis for knowledge discovery in multiobjective topology optimization. *Expert Systems with Applications*, 119:247–261, 2019.

[34] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[35] T. Uno, H. Maegawa, T. Nakahara, Y. Hamuro, R. Yoshinaka, and M. Tatsuta. Micro-clustering by data polishing. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pages 1012–1018, 2017.