

LEBANESE AMERICAN UNIVERSITY

Correlation Clustering via s-Club Cluster Edge Deletion: Theory and
Experiments

By

Norma Makarem

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 2023

THESIS APPROVAL FORM

Student Name: NORMA MAKAREM I.D. #: 200202205

Thesis Title: Correlation Clustering via s-Club Cluster Edge Deletion: Theory and Experiments


Program: MS in Computer Science


Department: Computer Science and Mathematics

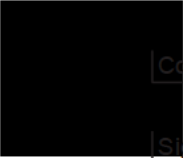

School: Arts and Sciences

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: 	Date: 19 / 05 / 2023 <small>Day Month Year</small>

Committee Member's Name: Dr. Eileen-Marie Hanna	
Signatur: 	Date: 19 / 05 / 2023 <small>Day Month Year</small>

 Committee Member's Name: Dr. Ramzi Haraty	
Signature: 	Date: 19 / 05 / 202 <small>Day Month Year</small>

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: NORMA MAKAREM

Signature:



Date: 05 / 07 / 2023

Day Month Year

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: Norma Makarem

Signature:



Date: 05 / 07 / 2023

Day Month Year

ACKNOWLEDGMENT

I would like to take this chapter as a window of opportunity to highlight my deepest gratitude for my advisor Dr. Faisal Abu Khzam that has led and inspired many generations in this field, where I am lucky to have been a member. In particular, his review and ongoing advice in this work. An additional appreciation to the committee members Dr. Ramzi Haraty and Dr. Eileen-Marie Hanna for their stand for my achievement of this work. Finally, I could not have undertaken this journey without the support of my husband and kids that are proud of my achievement. My final and special thanks go to the institution that I have been part of and proud of since 2004 the Lebanese American University.

Correlation Clustering via s-Club Cluster Edge Deletion: Theory and Experiments

Norma M. Makarem

ABSTRACT

Cluster Editing, a known model for correlation clustering, has garnered significant consideration in the parameterized complexity area and has been utilized in a range of practical contexts. In certain situations, the requirement for clusters to be cliques was deemed excessively stringent, leading to the proposal of alternative relaxed clique models for dense subgraphs, such as s-club. In this work, we implement three approaches to tackle the 2-club clustering via edge deletion: a heuristic approach based on the influence of the edge to resolve maximum conflicts, a parameterized algorithm in which by deleting a maximum of k edges, the graph can be transformed into a 2-club cluster based on a branching algorithm, and the approach in Integer Linear Programming to find the optimized solution in an integer formulation. We run these algorithms and cross-compare the three experiment results to the fastest Cluster Editing algorithm which runs in $O(1.62^k)$ time. Our 2-club Cluster Edge Deletion approach showed better performance than Cluster Editing in terms of running time, and significantly less cost of modifications while preserving the information of the underlying structure. Finally, we consider the 3-clubs Cluster Edge Deletion which did not have much focus in the literature. The problem can be solved in time $O(4^k)$. We present a first fixed-parameter algorithm that breaks this $4k$ barrier by solving the problem in $O(3.65^k)$ time. In addition, we implement this 3CCED algorithm in the heuristic approach and compare the experiments to Cluster Editing and 2CCED. The results show that the 3-club Cluster Edge Deletion presents the highest performance

in terms of running time and cost of modifications to reach the desired results. This shows that deletion into a graph whose components are of bounded diameter can be a better model for correlation clustering.

Keywords: Correlation Clustering, Cluster Editing, s-Club Cluster Edge Deletion, 2-club, 3-club.

TABLE OF CONTENTS

Chapter	Page
I- Introduction	1
II- Preliminaries	5
2.1 Terminology	5
2.2 Cluster Editing	6
2.3 Parameterized Complexity	7
2.4 Heuristic Algorithm	7
2.5 2-Club Cluster Edge Deletion	8
III- A Heuristic Approach for 2-Club Cluster Edge Deletion	9
3.1 Implementation.....	10
IV- A Fixed-Parameter Algorithm and its Implementation	12
4.1 Pre-processing Techniques	13
4.2 A Simple Branching Algorithm.....	13
4.3 Implementing the Fixed-parameter Algorithm	14
V- Experimental Analysis	17
5.1 Heuristic vs. Parameterized 2CCED Results.....	17
5.2 Heuristic vs. Cluster Editing.....	18
5.3 2CCED vs. Cluster Editing.....	19
VI- Integer Linear Programming Formulation	21
6.1 Formulation	21

6.2	Experimental Results	22
VII-	Introducing 3-Clubs Cluster Edge Deletion	23
7.1	Problem Identification	24
7.2	A Reduction Procedure.....	25
7.3	Branching Rules	26
7.4	A Heuristic Approach for 3-Club Cluster Edge Deletion	35
7.5	Experimental Analysis.....	37
VIII-	Summary and Future Work	39
	Bibliography.....	41-46

LIST OF TABLES

Table		Page
1	Experiments for Heuristic and 2CCED algorithms.....	18
2	Experiments for Heuristic and Cluster Editing algorithms	19
3	Experiments for 2CCED and Cluster Editing algorithms	19
4	Experiments for 3CCED heuristic and Cluster Editing algorithms.....	38
5	Experiments for 3CCED heuristic and the 2CCED algorithms	38

LIST OF FIGURES

Figure		Page
1	A conflict quadruple also known as a P_3	14
2	Induced path and an edge in common.....	14
3	A tail form of length four.....	26
4	Conflict quintuple with pendant endpoints	26
5	Conflict quintuple with pendant endpoints	27
6	b has a neighbor at a distance two from e	29
7	Neighbor of b is at distance of three from e	29
8	Neighbor of c at distance three from e passing through d	30
9	Neighbor of c at distance three from e not passing through c and d	31
10	Neighbor of c at distance two from a	32
11	Neighbor of c at distance one from a and two from d	33
12	Neighbor of c at distance two from a and two from d	34

CHAPTER ONE

INTRODUCTION

Correlation Clustering also referred to as Cluster Editing is an optimization problem that was extensively examined in various research works [1, 5, 3, 15, 16, 26, 28, 30, 34, 43], surveys [20], and practical settings [12, 13]. The technique entails dividing a group of elements into uniform and distinct subsets. The vertices in this problem represent the data elements, while the edges between the vertices represent their similarity. The permitted number of modifications varies based on the particular problem variation and its intended application. This problem has various applications, including computational biology [43, 44], machine learning [11], bioinformatics [14], and psychology [46].

Cluster editing from a graph theoretic view is an NP-Complete problem that focuses on transforming graph G into G^t by applying addition and deletion operations on a maximum of k edges, and the resulting graph contains only cliques as connected components. To solve the problem, we must look for the lowest number of edge adjustments required. When the edges are weighted, the modifications will be given a certain cost that corresponds to the weights of the edges modified. In this latter case, the problem would be named Weighted Cluster Editing, and the main objective is to find the edge adjustments having in total the smallest weight.

In [1], a modified version was presented that incorporates multiple parameters. This revised version establishes a minimum size for the clusters and also applies a

boundary on how many edges can be added or removed from a vertex. With these modifications, the author was able to tackle this clustering approach in polynomial time, provided that the total edge modifications for each vertex does not exceed half of the minimum cluster size. However, they proved that even when only one edge deletion or two edge deletions are allowed per vertex this problem is still NP-hard. By limiting false positive and/or false negative correlations attributed to a specific variable, this method enhances the efficiency of obtaining optimal correlation clustering solutions.

Another version of the graph clustering problem was studied in which every cluster must fulfill a specific local constraint [37]. They focused on the cluster's non-edges count, the number of vertices in each cluster, and the number of non-neighboring vertices that each vertex is allowed to have within its cluster. Additionally, they demonstrated that the algorithm's fixed-parameter tractability was determined by these local constraints.

As the Cluster Editing problem imposes that each node is in a single cluster, yet a member in a certain real-life application can have roles in multiple clusters, an example is discussed in gene regulatory networks [2]. Hence, the clustering with overlaps problem was introduced [23] and achieved fixed-parameter tractability and parameterized hardness under certain constraints. The graph produced includes maximal cliques that may overlap depending on the overlap parameter s . Hence, their application on s -edge-overlap and s -vertex-overlap. In the same focus, a more recent variant was introduced

[5] covering the problem of clustering with Vertex Splitting whereby allowing a single vertex to be a member of many clusters after it is split into many vertices. This work demonstrates that it is FPT (fixed-parameter tractable) if bounded by the count of edge editing and vertex division operations.

An introduction to the problem of clustering via Vertex Deletion was initiated in [27] where they allowed vertex deletion as the form of modifications and expected in practical applications that this measure to be much lower than the edge-editing distance because deleting a vertex would remove all its linked edges as well. An improvement

in its branching time is shown [17] by presenting a number of observations incorporated into the algorithm. The variant of modifying the graph by deleting vertices has attracted other considerable work [21, 31, 45].

The PACE Challenge which stands for "Parameterized Algorithms and Computational Experiments" for the year 2021 [32] was focused on algorithms that tackle the problem of Cluster Editing, and the majority of the submissions employed a *branch and bound* algorithm, data reduction rules, bounded tree search pruning, and local search. Out of all the participants, Tobias Heuer [33] emerged as the top-ranked contestant, and his contribution will be referenced later in this work along with the dataset used in this competition. The graphs used are gathered from different data sources [19, 41, 49, 42, 35] to cover multiple areas, sizes, and complexity in Bioinformatics, newsgroups posts, and social networks.

For many applications, It was discovered that the requirement for clusters to be in the form of cliques was limiting and strict. Therefore, the use of clique relaxation as a mean of identifying clusters obtained popularity in graph-based data clustering [29] and alternative models to strict cliques have been proposed including quasi-clique, s -plex, and s -club. These relaxed clique models are applicable to dense subgraphs and involve defining vertex subsets that induce subgraphs with diameters no greater than specified parameter s .

Allowing the model to have a more relaxed cluster is an advantage of s -clubs for $s \geq 2$, better-reflecting inaccuracies of the input data. In practical applications, the low-diameter clusters have high significance, which has led to the utilization of this model in various fields such as protein interaction networks [10] and social networks [7].

The concept was introduced to model not only connected subgroups in social networks [39] but "acquaintance groups" as well which are less tightly-knit, homogeneous social groups, where each pair of members who are not in direct contact have mutual acquaintances or share common third contacts.

Numerous types of NP-Complete graph editing problems [36] have been explored

with the goal of transforming a graph into disconnected groups of 2-clubs, including 2-Clubs Cluster Editing, 2-Club Cluster Edge Deletion, and 2-Club Cluster Vertex Deletion. Additionally, according to [24], it was demonstrated that 2-Club Clustering is $W[2]$ -hard based on the number of altered edges, which implies that it is probably not FPT. Moreover, the vertex deletion approach was proved to be FPT but not poly-kernelizable only if $NP \subset co-NP/poly$.

In this work, we are tackling the 2-CLUB CLUSTER EDGE DELETION version which we refer to as 2CCED. We present a heuristic approach and an exact parameterized algorithm while testing their implementation with experience results. We then cross-compare the results of both the beforementioned approaches and the Cluster Editing algorithm. Moreover, we introduce a theoretical study of 3-club cluster edge deletion, in addition to its experimental results of applying a heuristic approach and presenting its comparative analysis with the 2-Club Edge Deletion and Cluster Editing approaches.

The arrangement of the thesis goes as follows: In Chapter 2 we outline the preliminaries; Chapter 3 illustrates the Heuristic approach, along with the 2CCED algorithm; Chapter 4 presents the fixed Parameter Algorithm and its Implementation; Chapter 5 consists of the Experimental Analysis; Chapter 6 presents the problem formulation in Integer Linear Programming, Chapter 7 introduces the 3-clubs Edge Deletion approach and its experimental results, while Chapter 8 provides the concluding Summary and Future Work.

CHAPTER TWO

PRELIMINARIES

2.1 Terminology

Consider G , an undirected graph with no weights. We often write $G = (V, E)$, with V representing the set of G 's vertices and E for its edges. No self-loops or multi-edges are allowed in G . The distance from vertex u to w is the length of the smallest route to go from one vertex to another and is represented as $d(u, w)$. The longest route, on the other hand, would be the graph's diameter. $N(w)$ being the set of neighbors for vertex w , and $degree(w) = |N(w)|$ is hence its degree. If $degree(w) = 1$ then we call vertex w a pendant one. We utilize the standard terminology of graph theory as present in [48].

A complete graph is defined as undirected where one cannot find any two vertices that are not connected by an edge. Similarly, a clique is a complete subgraph. A subgraph $G^t(V^t, E^t)$ of G has V^t and E^t as a subset of V and E , in G , respectively.

A graph is said to be undirected if the edges have no direction or orientation, and unweighted if the edges are not assigned any values or weights. A graph is considered connected when every vertex is connected to (or reachable from) all other vertices in the graph. The use of weights in graphs typically involves assigning values to edges, which may represent costs, distances, or other problem-specific values. However, in this work, we do not consider weighted graphs, as we tackle only the simplest form of unweighted graphs.

Furthermore, a path can be defined as a graph or subgraph in which its vertices

can be arranged in an order that ensures that two vertices are adjacent only if they are adjacent (or consecutive) in the sequence. A path P is a series of distinct vertices $(v_1, v_2, v_3, \dots, v_t)$ where $v_i v_{i+1} \in E$ for all $i \in \{1, \dots, t\}$. When the length of a path P is $k - 1$, P is considered to be of length l , denoted as P_l (where P_l has $l + 1$ vertices). A P_3 is an induced path that has four distinct vertices. When given a $P_3 abcd$, we refer to a and d as the end vertices of this P_3 , and b and c as its internal vertices.

An s -club is a subgraph with a maximum diameter of s . Consequently, a clique is obtained when $s = 1$. Hence, an s -club cluster graph has only s -clubs as components. In this work, we focus on 2-clubs and 3-clubs graphs which are components with vertices of distance at most 2 and 3 respectively.

2.2 Cluster Editing

Cluster Editing considers a graph G with an integer k , then seeks to determine if G can be modified using k or less edge editing operations, to become a set of cliques. The operations refer to either adding or deleting an edge from the graph. If each connected component of G is of a clique form, then the graph is a cluster graph. Forbidden edges are those that should not be included in the resulting graph, while permanent edges must be present in the final graph.

A conflict triple refers to a sequence of three adjacent vertices forming a path of length two. Its existence, as the name represents, forbids the solution to be disconnected groups of cliques. We denote a conflict triple by (abc) of edges ab and bc and nothing connecting a to c , and b is the center of the conflict triple. The most basic approach to solving the Cluster Editing problem involves identifying a conflict triple within the given graph. From there, the algorithm considers two possible scenarios: removing one edge or adding an absent one to resolve the conflict. As such, this algorithm runs in $O^*(3^k)$. Notably, this problem limits each vertex to belong to a single cluster. Other variants of the problem, such as clustering with Overlaps and vertex splitting, allow a vertex to be part of more than one cluster.

2.3 Parameterized Complexity

Efficient and exact solving algorithms for NP-complete or NP-hard problems are unlikely. However, some problems are solved in time that is a polynomial function of the input size, while having an exponential size of the associated (assumed to be small or fixed) parameter. These algorithms are known as fixed-parameter tractable (FPT), and they are effective in solving problems for (fixed) parameters of a small value. If a parameterized problem that has a fixed parameter k can be solved using an FPT approach, it is considered a fixed-parameter tractable problem. Parameterized algorithms require a cost limit k to be specified beforehand. When considering a minimization problem, if a solution is found with a budget at k or less, then the algorithm returns a positive answer and a solution is found; otherwise, it informs that no solution has been found.

Kernelization is the technique for efficient algorithms that use preprocessing in which inputs to the algorithm are substituted by a smaller input that is called a "kernel." It is usually easy to demonstrate that a kernel, which has predetermined constraints on its size, can be obtained within polynomial time. A reduction of a parameterized problem from an instance (I, k) into (I', k') by:

1. (I, k) instance has a positive answer if and only if (I', k') has a positive answer.
2. $k' \leq k$.
3. $|I'| \leq f(k)$ for a measurable function f .

2.4 Heuristic Algorithm

Heuristic algorithms are intended to solve problems more quickly and efficiently than traditional methods by prioritizing speed over optimality, accuracy, precision, or completeness. These algorithms can either generate a solution on their own or be combined with optimization algorithms to provide a solid starting point (see [4, 22] for more information on such hybrid methods). Heuristic algorithms are often the

preferred choice when approximate solutions are adequate and exact solutions are too computationally demanding.

2.5 2-Club Cluster Edge Deletion

The problem at hand involves a graph $G = (V, E)$ that is undirected and a $k \in \mathbb{N}$. We are looking to see if the graph is transformable to a set of disjoint 2-clubs by deleting no more than k edges. The diameter of a 2-clubs graph cannot exceed two. For any given vertices $u, v \in V$, this means that they are either adjacent or have a neighbor in common. A path $P_3 = stuv$ in G is called a conflict quadruple if $dist(s, v) = 3$. Here, P_3 represents the shortest path connecting s and v . Consequently, G is considered a 2-club cluster graph only if it does not contain any restricted P_3 .

CHAPTER THREE

A HEURISTIC APPROACH FOR 2- CLUB CLUSTER EDGE DELETION

Our work addresses the problem of 2-clubs Cluster Edge Deletion by first introducing a heuristic approach and its implementation. The objective of this approach is to develop an algorithm that can quickly produce a practical solution while compromising on optimality. The approach operates by removing the least number of edges that would eliminate all conflict quadruples in the graph, thus achieving a fast solution. Consequently, the algorithm leverages the *influence* level of an edge, which indicates the number of P_3 s passing through it, to determine which edges to cut.

In other words, an edge influence would indicate the number of P_3 s that decreases in the overall graph whenever this edge gets deleted. The higher the number of P_3 s it affects, the higher the edge's influence value is. Hence, by deleting the edges with the highest influence, we are minimizing the cost of edge deletion in the overall problem to transform graph G to a disjoint union of 2-clubs. To know the influence value for all edges, we apply the following. We assign a vertex as the source, then by using the Depth First Search method we traverse the graph from this vertex to its 3rd-degree neighbors. We count the number of P_3 s it is part of and increase its weight by this

number. We loop through all the vertices, with each vertex being a source and depth-first search is applied to find P_3 s and update the edges' weights accordingly. When the final edge weights are allocated then begins the edge deletion process. Starting with the highest weight, its edge is deleted which will decrease the highest number of P_3 s in this current graph version. Because of this deletion, edges' weights would be affected, hence a recalculation of weights is done after each edge deletion, only to the affected edges and not all graph edges. The affected edges when a deletion is applied are the ones connecting the vertices of the direct neighborhood and second-degree neighborhood. The algorithm will continue to iterate until the graph no longer contains any conflict quadruples. The algorithm does not make any prior assumptions on the number of clusters or their structure while assuming the input graph is undirected and unweighted.

3.1 Implementation

We have implemented the algorithm on the PACE dataset described earlier and the experimental results are presented in the following chapters. The implementation was written in CPP programming language.

We first initialize all edge weights to 0. Weights represent the influence level in this approach. Then we traverse the graph using depth-first search to store the weight of each edge representing the number of P_3 s it went through, hence its influence. We traverse the graph by looping through all the vertices as the source. Accordingly, we then delete the edge with the highest weight. Then, to optimize on time, we update the weights on the affected edges only which are linked to the vertices at distances 1 and 2. Then repeat the last two steps until all edge weights are 0, hence no P_3 is found in graph G which is converted to a disjoint union of 2-clubs.

Algorithm 1: Heuristic 2-clubs cluster edge deletion pseudo code

Input : A graph $G = (V, E)$

Output: The minimum edge deletions

$w_i \leftarrow 0$

initialize all edges weight to 0;

for each $v \in V$ **do**

 DFS from source $v \in V$.

if edge e is part of a P_3 **then**

 | $w(e) = w(e)+1$

else

 | Do nothing

end

end

while $w(e) > 0$ **do**

 Delete the highest weighted edge. Recalculate the weights of the nearest edges only which are linked to vertices at distances 1 and 2 from the deleted edge.

end

CHAPTER FOUR

A FIXED-PARAMETER ALGORITHM AND ITS IMPLEMENTATION

We tackle the 2CCED problem by presenting a fixed-parameter approach and its implementation. Starting from an undirected and unweighted graph, and an integer value of k that indicates the budget of edge deletion we have. In other words, it is the maximum number of edge deletion operations we can apply before we can say "no solution". This algorithm is based on the approach discussed previously [6] that resolves the conflict quadruples by removing one of its three edges. With each removal, which we call the branch in this work, we decrease the k parameter by one. However, during some branching, we further decrease k where more than one conflict intersects in a certain way. In addition, we describe cases where we can identify the edges that should be deleted without loss of optimality. We tackle these cases in polynomial time by applying reduction rules.

In this chapter, we will discuss the implementation details of our (previously published) fixed-parameter 2-clubs cluster edge deletion algorithm. For the sake of completeness, we present the pre-processing techniques and well as the main strategy (or logic) behind the branching algorithm, and finally the algorithm method.

4.1 Pre-processing Techniques

During the search process, We apply pre-processing actions by reduction rules on the input graph G , in polynomial time, whenever applicable.

Rule 1. The algorithm ends without a solution when k reaches a value below 0.

Rule 2. When the graph is empty, the algorithm ends and gives a solution-found result.

Rule 3. Delete all connected components that represent a 2-club sub-graph which includes all singletons.

Rule 4. Search for two non-adjacent vertices u and w with common neighbors more than k , then remove the connections between u and w to $N(u) \setminus N_2[w]$ and $N(w) \setminus N_2[u]$ respectively.

Rule 5. Search for connected components C with the highest degree is two, then C can optimally be altered into a subgraph of 2-clubs. We then decrease k : $k = k - d$ where d is the count of edges removed.

A tail is nothing but an induced path with a pendant endpoint. A 3-tail is a tail of length three.

Rule 6. Search for a 3-tail $T = (a, b, c, d)$, then remove the edge ab and reduce $k = k - 1$.

4.2 A Simple Branching Algorithm

As our main objective is to transform graph G into 2-clubs connected components, hence the primary structure that forbids this club from forming is what we call a P_3 , also known as conflict quadruple, which is a length three path and the distance between its endpoints is exactly three as shown in Figure 1 where $abcd$ form a P_3 where the distance between a and d is exactly 3 in G . The solution to our problem will result from searching for each P_3 and resolving it by deleting one of its three edges ab , bc , or cd . This algorithm runs in a $O^*(3^k)$.

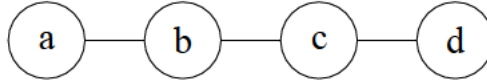


Figure 1: A conflict quadruple also known as a P_3

An Enhancement has been presented [6] of an algorithm that runs in $O^*(2.695^k)$ when applying branching rules on specific cases which we are implementing in the following chapter.

4.3 Implementing the Fixed-parameter Algorithm

The method begins with reading the graph and initializing the starting vertex to start our search algorithm. By traversing the graph using a Depth First Search method, we aim to find a conflict quadruple as defined earlier. When found, we check its form (its neighbors, common vertices, if it belongs to a cycle) in order to identify to which case it belongs, then we branch accordingly. The cases are special forms that this P_3 belongs to for example as presented in Figure 2 the conflict quadruple $abcd$ belongs to a special case where the vertex b and vertex d have a common neighbor w .

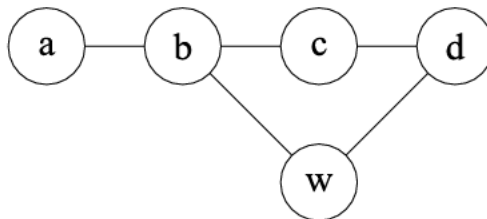


Figure 2: Induced path and an edge in common.

Hence it has a special branching logic which is:

- remove edge ab ;
- remove edges bc and bw ;
- remove edges cd and dw ;
- remove edges bc and dw ;
- remove edges cd and bw .

A similar approach is applied to all special cases discussed in [6] where we delete edges in a specific order to get the most efficient results.

The method can be viewed as a search-tree traverse and works in a recursive manner. Therefore, the running time is proportional to the number of recursive calls. For this matter, we use the O^* notation, which indicated the count of recursive calls and conceals polynomial factors.

An edge is marked permanent if all the searches following its selection have returned a no-solution answer. Then this edge is not part of the solution.

The algorithm converges to an end when there is no P_3 found with a 0 or positive k which indicated a solution found, or the search has been completed with all instances resulting in a negative parameter k .

The algorithm does not make any prior assumptions on the number of clusters or their structure while assuming the input graph is undirected and unweighted, parameter k (positive integer). We also consider that an instance (G, k) was preprocessed by comprehensively applying the reduction process.

Algorithm 2: Parametrized 2-clubs cluster edge deletion pseudo code

Input : A graph $G = (V, E)$ preprocessed, and integer k

Output: A disjoint union of 2-clubs clusters, count of deletion operations

while $k \geq 0$ **do**

for each $v \in V$ **as source do**

 Search for a P_3 ; **if no** P_3 **was found for all** v **then**

 | Return True;

else

 compare the selected P_3 form to the special cases; **if the selected** P_3
 form is found to be similar to a special case **then**

 | branch accordingly;

else

 | branch exhaustively going through the 3 edges;

end

 After each branch, $k = k$ -number of deletions;

end

end

end

CHAPTER FIVE

EXPERIMENTAL ANALYSIS

In this chapter, we discuss the findings of running our algorithms on a set of different-size graph networks that were used by the exact cluster editing solver ranked first [25] by the PACE Challenge 2021 (Parameterized Algorithms and Computational Experiments). The algorithms built for this work were implemented using the CPP language and run on a PC Intel Core i5, 1.19 GHz with 8 GB RAM. The objective behind these applications is to show the results of the parameterized approach and compare it to different approaches such as the Heuristic approach of a 2-club cluster editing and the cluster editing presented in [25].

5.1 Heuristic vs. Parameterized 2CCED Results

We work with graphs with vertices ranging between 20 and 350. The results are reported in Table 1 below with a comparison between the Heuristic 2-clubs cluster editing and the exact parameterized approaches implemented and discussed earlier in this work. In the results, the metrics measured are:

- (i) Time of the job running in milliseconds;
- (ii) Cost: is the count of deletions during the process; and
- (iii) Clusters: the number of components in the resulting graph including singletons.

Table 1: Experiments on different graphs for Heuristic and 2CCED algorithms

Graph		Heuristic 2CCED			2CCED		
Nodes	Edges	Time(ms)	Cost	Clusters	Time(ms)	Cost	Clusters
20	97	0.17	1	2	0.31	2	3
144	1191	6.55	8	40	405.02	8	40
144	3089	10.39	6	14	261.89	6	14
159	425	2.97	7	57	210.13	7	57
184	441	3.88	8	53	1250.34	8	54
216	3054	15.08	14	50	74.67	13	51
232	5582	16.02	4	32	17.56	1	32
263	1288	8.69	8	99	18.50	5	99
313	1764	12.57	3	109	40.18	3	109
350	1770	14.70	10	146	1103.05	9	146

Table 1 results show that, on average, the heuristic approach is 4.5 times faster in terms of running time, but might not be optimal in terms of effectiveness. We can observe that the parameterized approach takes relatively more running time but proves effectiveness in its final clusters and its lower cost of edge deletion.

5.2 Heuristic vs. Cluster Editing

The experiment was conducted on the same set of graphs as described earlier but with different approaches to cluster editing. The comparison was made between the Heuristic 2-clubs clustering and the fixed-parameter cluster editing algorithm [25], which clusters a graph with the smallest count of editing operations on the edges. The results, presented in 1, adopt the same metrics as in the previous experiment.

Table 2 results show that the cost is on average 10 times lower in the Heuristic 2CCED approach than the cluster editing while preserving an average of an equal number of resulting clusters. This proves that aiming for 2-clubs clustering instead of hard complete graphs has a much lower cost and maintains very close results in terms of connectivity and the number of clusters.

Table 2: Experiments on different graphs in Heuristic and Cluster Editing algorithm

Graph		Heuristic 2CCED			Cluster Editing		
Nodes	Edges	Time(ms)	Cost	Clusters	Time(ms)	Cost	Clusters
20	97	0.17	1	2	80	46	1
144	1191	6.55	8	40	60	132	43
144	3089	10.39	6	14	274	78	20
159	425	2.97	7	57	289	42	55
184	441	3.88	8	53	326	72	64
216	3054	15.08	14	50	908	134	56
232	5582	16.02	4	32	177	20	34
263	1288	8.69	8	99	822	56	107
313	1764	12.57	3	109	883	100	117
350	1770	14.70	10	146	873	106	153

5.3 2CCED vs. Cluster Editing

The aim of this experiment is to assess the effectiveness of the parameterized method for 2-clubs cluster edge deletion in comparison to the cluster editing algorithm [25] that was previously discussed. Graphs with orders varying from 20 to 350 were utilized for this purpose. The outcomes of the clustering were evaluated by applying both techniques to the same graphs that were utilized in the preceding experiments. The findings are presented in Table 3, which also adopts the metrics used in the previous experiments (time, cost and number of clusters found).

Table 3: Experiments on different graphs for 2CCED and Cluster Editing algorithms

Graph		2CCED			Cluster Editing		
Nodes	Edges	Time(ms)	Cost	Clusters	Time(ms)	Cost	Clusters
20	97	0.31	2	3	80	46	1
144	1191	405.02	8	40	60	132	43
144	3089	261.89	6	14	274	78	20
159	425	210.13	7	57	289	42	55
184	441	1250.34	8	54	326	72	64
216	3054	74.67	13	51	908	134	56
232	5582	17.56	1	32	177	20	34
263	1288	18.50	5	99	822	56	107
313	1764	40.18	3	109	883	100	117
350	1770	1103.05	9	146	873	106	153

Although the running time of the parameterized approach of the 2-clubs cluster edge deletion is on average 9 times higher (for 80 percent of the experiments), the cost of edge operations is 12 times lower than the Cluster Editing approach that gives a union of complete graphs, while maintaining a close output in terms of connectivity and the number of clusters.

CHAPTER SIX

INTEGER LINEAR PROGRAMMING FORMULATION

Integer programming models (ILP) have been part of many works to solve cluster editing and particularly 2-clubs clustering [10, 18] with a simple formulation, straightforward generalization of the classical maximum clique mainly focusing on vertex deletion operations. Another extension of this approach [47] introduced two new mixed-integer programming models.

6.1 Formulation

We provide in this section an ILP approach for the 2-Clubs Cluster Edge Deletion. The main requirement of this problem is that the resulting graph has to be a 2-club cluster graph which is a subgraph of maximum cardinality with a diameter at most two. To satisfy that condition, the resulting subgraph should include no conflict quadruple or as we note it in this work, induced P_3 s.

We denote by $N(i)$ the neighborhood of node $i \in V$, i.e., $N(i) = \{ j \in V \mid (i, j) \in E \}$. So, for any induced $P_3 \equiv abcd$ in graph G , if $N(a) \cap N(d) = \emptyset$; then at least one edge from ab , bc , or cd must be deleted for this conflict to be solved. We introduce a variable x_i for each edge $i \in E$. The value of this variable is 1 if i is in the set of the edges deleted, and it will take the value of 0 if i is not in the maximum 2-club solution

set. We shall label the edges ab , bc , and cd by u , v , and w respectively. This leads to the below ILP formulation. The aim is to find the minimal count of edges to be deleted for the solution to be a 2-club cluster graph, hence it is a minimization optimization with the edge deletion as the cost.

Hence, we need to find the minimal cost of the sum of all x_i where $i \in E$.

minimize:

$$\sum_{i \in E} x_i$$

subject to: $x_u + x_v + x_w \geq 1$ for all induced P_3 's $abcd$ in G .

$x_i \in \{0,1\}$ for all $i \in E$

6.2 Experimental Results

The algorithm above was coded in the CPP language and run on a PC Intel Core i5, 1.19 GHz with 8 GB RAM, and solved with ILOG/CPLEX 22.1. Many tools are focused on linear optimization problems and implement optimizers based on simplex algorithms such as Baron, Cplex, and Gurobi. In this work, we have chosen to work with the CPLEX tool.

The experiments were performed on the dataset used in the previous approaches and were compared against the heuristic approach. The results showed that the cost is almost the same on all tested graphs, proving the effectiveness of the 2CCED approach presented earlier in this work. For example, the graph with 159 nodes and 425 edges showed a cost of 7 deleted edges, and the graph with 184 nodes and 441 edges showed a cost output of 8 edge deletions which are equal to our 2CCED approach presented earlier. Graphs with more than 500 edges were too large to be completed within a reachable time because the ILP approach would run exhaustively on every path of 3 and study the cost of each of these 3 edges. Hence this conclusion was based on the smaller graphs that have been able to be completed within a reasonable time.

CHAPTER SEVEN

INTRODUCING 3-CLUBS CLUSTER

EDGE DELETION

In various practical cases, the requirement for clusters to be cliques was limiting and strict, subsequently, several alternative relaxed clique models were introduced. Noting that 2-clubs and 3-clubs are presented as the most rational diameter-relaxations of clique [10, 40], a few approaches were presented to deal with 3-club as well. Similar to the 2-clubs, 3-clubs clustering is proven to be run in polynomial time on a tree form graph [8]. Another work on the 3-club problem focuses on presenting an Integer formulation [9]. Their work shows that the compact approach has high competition with the methods presented prior to that work [18] that explores a branch and limit algorithm based on Vertex Deletion. The 3-Club 2-Cluster Edge Deletion approach [38] was also presented and proven that it cannot be solved in time $2^{o(k)}n^{O(1)}$ unless ETH fails.

A modified version of the 3-Clubs Clustering problem is here introduced that concentrates solely on the Edge Deletion operation. An algorithmic method is presented that involves progressively removing special setups, such as bounded-degree-three, a four-length tail, special three-length paths, and five-length paths, after implementing some reduction procedures that are presumed to be carried out during the search process and prior to making any deletion decisions. The findings reveal a runtime of $O(3.65^k)$.

7.1 Problem Identification

The 3-CLUB CLUSTER EDGE DELETION (3CCED) problem at hand involves an undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$. We are looking to delete at most k edges from G to convert it into a collection of disjoint 3-clubs.

When the highest degree in a graph is two or less, the 3CCED problem can be easily solved in polynomial time. When a path in the graph is identified as $P = (v_1, v_2, \dots, v_s)$, we can consecutively remove edges $v_i v_{i+1}$ for $i = 1, 2, \dots$, until there are no more 3-clubs left to remove. In this situation, this approach results in the optimal solution.

Taking another case of a cycle-connected component of length > 7 then we shall delete a random edge resulting in a path that is disconnected and can be resolved as described earlier.

A 3CCED problem produces a solution graph that comprises of connected components known as 3-clubs graph, which are subgraphs of diameter three. A path with a length of four and endpoints that are precisely four hops apart from each other will be denoted in this work as a *conflict quintuple*, and it is the primary structure that we need to destruct due to the fact that its existence hinders a graph from being a 3-clubs graph.

The method we use to solve the problem is to search for a conflict quintuple and attempt to resolve it by eliminating one of the four edges that form it. As we proceed, the parameter k is reduced by one in each branch. This produces a $O^*(4^k)$ approach. Nevertheless, there are instances where multiple conflicts overlap in a manner that permits us to decrease the parameter even more in certain branches. In addition, there are less complex scenarios where we are aware of the precise edge or set of edges to eliminate "without losing optimality." These situations can be addressed using polynomial-time procedures that rely on reduction rules.

7.2 A Reduction Procedure

Before executing the search-tree backtracking algorithm, we presume that the reduction procedure has been thoroughly implemented. Additionally, during the search procedure, the reduction rules are applied before any selection or determination is made by the search algorithm. We outline below the primary reduction rules which are expected to be executed sequentially.

Reduction Rule 1. Whenever the parameter k reaches a negative value, the algorithm terminates and outputs a message indicating the absence of an instance.

Reduction Rule 2. Assuming $k \geq 0$ as per the previous rule, the algorithm ends and returns a positive instance if the graph is empty.

Reduction Rule 3. If we find a 3-club connected component C in G , then we remove C .

In case G comprises a connected component C that meets the criteria of being a 3-club, then we delete this component.

Notice that applying Rule 3 exhaustively leads to the elimination of all singletons.

Reduction Rule 4. If a connected component D with a maximum degree of two is found in the graph, then D can be transformed into a 3-clubs subgraph in an optimal manner. k value will be reduced by the number of edges that were removed.

Soundness. A connected component with the highest degree equal to two would be a cycle or a path and can be resolved as explained earlier.

Reduction Rule 6. If a form of 4-tail $T = (a, b, c, d, e)$ was found in the graph (as depicted in Figure 3), then the edge ab will be removed, and k value decreased by 1.

Soundness. As the a and d vertices should be members of two distinct 3-clubs, then we must remove one of the three edges that make up T , because removing the edge ab creates an isolated 3-club which is the path consisting of b, c, d , and e . And cannot reach a suboptimal solution.

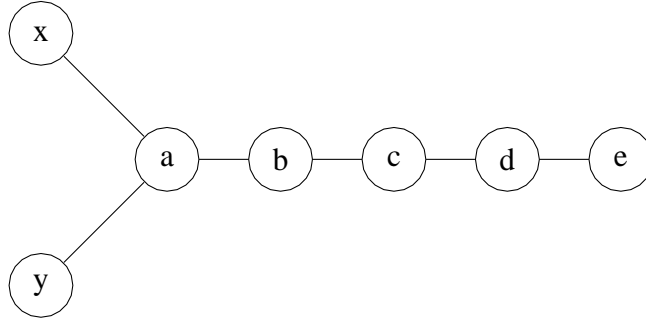


Figure 3: A tail form of length four.

7.3 Branching Rules

We introduce here our bounded search tree algorithm. It is a recursive procedure whose running time is directly proportional to the number of recursive calls. As a result, we employ the O^* notation that represents the aggregate count of recursive calls and conceals any polynomial factor.

We examine a pre-processed instance (G, k) of 3CCED, where reduction rules have been exhaustively applied. Hence, a solution is found when graph G is empty, or each connected component of G has one or more vertices with a degree of 3 or higher and at least two vertices situated at a distance of precisely four from each other. We apply this sequence of events to the following branching rules as well. Therefore, in every instance, we presume that none of the circumstances mentioned earlier apply.

Case 1. Neighbors of endpoints of a P_3 .

In case there exists an induced path $P = (a, b, c, d)$ of length three where $|N(a) \setminus N_3[d] \cup N(d) \setminus N_3[a]| \geq 2$, as shown in Figure 4, we can branch by deleting ab , bc , or cd , or all the vertices in $N(a) \setminus N_3[d] \cup N(d) \setminus N_3[a]$. This leads to a worst-case recurrence of $T(k) = 3T(k - 1) + T(k - 2)$, with a running time of $O^*(3.303^k)$.

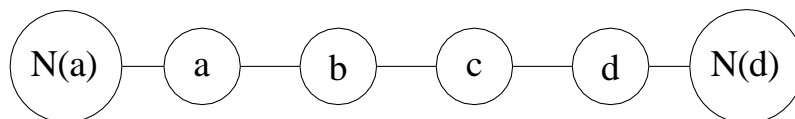


Figure 4: Conflict quintuple with pendant endpoints

Soundness. Concerning the first three branches, each branch deletes one of the four

edges of a conflict quintuple that includes a sub-path (a, b, c, d) . At the fourth branch, the edges ab , bc , and cd turn to a permanent state making it necessary to delete any neighbor of a that is at a distance of four from d , in addition to any neighbor of d that is at a distance of four from vertex a .

Remark. *We shall exclude two notable cases in the following discussion, where the above branching scenario implicitly applies.*

- *If a path (a, b, c, d, e) has internal vertices $(b, c$ and $d)$ of degree-two, then the distance from d to every neighbor of a is exactly four, and the same logic holds true for the situation of e and b . Therefore, we can apply Case 1 branching condition on the path (a, b, c, d) . We will implicitly exclude this case henceforth.*
- *When two vertices u and w where the distance $d(u, w) = 5$, then the four vertices between them on the shortest path will abide by Case 1 condition as well.*

Case 2. Conflict quintuple with pendant endpoints.

In this case, as shown in Figure 5 we have a conflict quintuple (a, b, c, d, e) where the degree of the endpoints vertices a and e is 1, and the internal vertices b and d have only one neighbor of degree-one (otherwise we will apply Case 1). Hence the deletion of edge bc or cd will solve only one conflict and might lead to more conflicts, but the deletion of the edge ab or de can solve one or more conflicts without creating additional conflict quintuples. For that reason, the branching in this particular case will be the deletion of either ab or de , which results in a running time of $O^*(2^k)$.

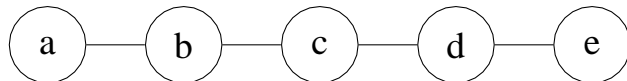


Figure 5: Conflict quintuple with pendant endpoints

Henceforth, we can assume that vertex e has one or more neighbors other than d . The distance between this neighbor of e and the internal vertex b is, therefore, two or

three, because if the distance between b and this neighbor is four, then we cannot in fact have a path (b, c, d, e) in the graph.

Case 3. b has a neighbor at distance two from e .

Let x and y be adjacent vertices and the neighbors of b and e respectively, as shown in Figure 6. The objective is to break the conflict quintuple $abcde$ but first deleting the edge ab , if that returned no solution then the edge ab becomes permanent. Our second option would be deleting edge bc which should coincide with deleting one of the 3 edges in the path $bxye$ in order for the distance between a and e not to be 4. Then we continue with the same logic with for edges cd and de . Hence, we branch as follows:

- delete edge ab ;
- delete edges bc and bx ;
- delete edges bc and xy ;
- delete edges bc and ey ;
- delete edges cd and bx ;
- delete edges cd and xy ;
- delete edges cd and ey .
- delete edges de and bx .
- delete edges de and xy .
- delete edges de and ey .

The recurrence of the above will be $T(k) = T(k - 1) + 9T(k - 2)$ and generates a running time of $O^*(3.541^k)$.

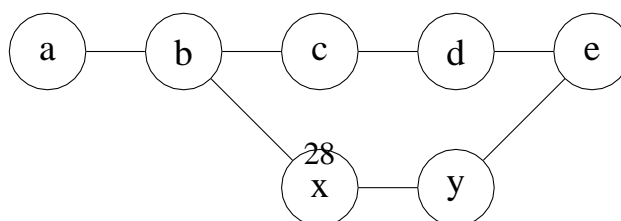


Figure 6: b has a neighbor at a distance two from e .

Case 4. b has a neighbor at distance three from e .

We consider an induced path $P4(b, w, x, y, e)$ that corresponds to this scenario, as illustrated in Figure 7. The distance between a and y will be four.

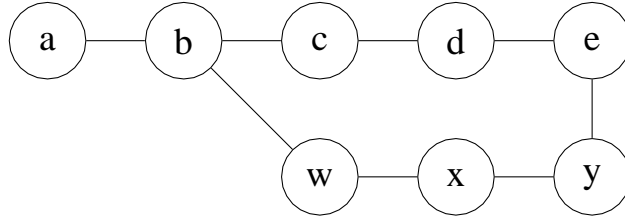


Figure 7: Neighbor of b is at distance of three from e .

If the distance between a and y is four then first we branch with ab , then we need to delete 2 edges, one of the 3 edges in $bcde$ and another from the 3 edges in $bwxy$, hence the branching will be as follows:

- delete ab ;
- delete bc and bw ;
- delete bc and wx ;
- delete bc and xy ;
- delete cd and bw ;
- delete cd and wx ;
- delete cd and xy ;
- delete de and bw ;
- delete de and wx ;
- delete de and xy ;

The recurrence of the above will be $T(k) = T(k - 1) + 9T(k - 2)$ and generates a running time of $O^*(3.541^k)$.

If the distance between a and y is three then we first need to delete ey then delete 3 branches to separate b from e and 3 branches to separate a from y . This will also lead to $T(k) = T(k - 1) + 9T(k - 2)$ and generates a running time of $O^*(3.541^k)$.

If the distance between b and y is three, we cannot assume $d(a,y) = 2$ because the distance between a and e would be less than four, hence no conflict quintuple.

From this point onward, we assume that b and d are of degree 2. Because if b has a neighbor of distance 1 from e then there is no conflict quintuple. if b has a neighbor of distance 2, 3, or 4 from e then we have solved it in the above cases. Similarly, it applies to its symmetric version on vertex d .

Case 5. c and a has a neighbor at distance one from a and three from e .

We consider in this case that vertices a and c has a common neighbor other than b , and the distance between this neighbor and vertex e is three passing through c and d as shown in 8.

Note. if the distance between this neighbor and e was less or equal to two than the conflict quintuple will not exist as $d(a,e)$ will be < 4 .

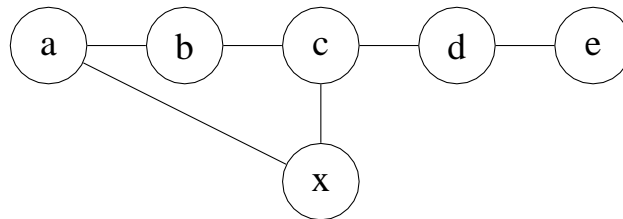


Figure 8: Neighbor of c at distance three from e passing through d .

To solve this conflict we will branch first by removing edges cd or de . Then when these edges become permanent, deleting ab will coincide with the deletion of ax or cx , and the same logic when branching with the edge bc .

For this case, the branching will be as follows:

- delete cd ;
- delete de ;
- delete ab and ax ;
- delete ab and cx ;
- delete bc and ax ;
- delete bc and cx .

The recurrence of the above will be $T(k) = 2T(k - 1) + 4T(k - 2)$ and generates a running time of $O^*(3.236^k)$.

Case 6. c has a neighbor at distance three from e from two paths.

We consider in this case that vertices a and c has a common neighbor other than b , and the distance between this neighbor and vertex e is three passing through two vertices other than c and d as shown in 9.

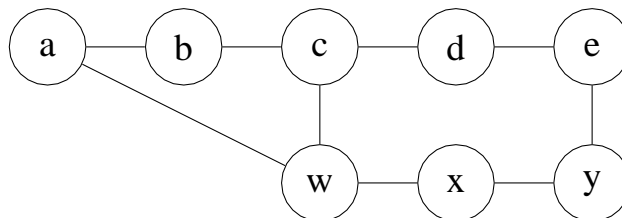


Figure 9: Neighbor of c at distance three from e not passing through c and d .

For this case, the branching will be as follows:

- delete ey and de ;
- delete ey and cd ;
- delete aw and ab ;
- delete aw and bc .

The recurrence of the above will be $T(k) = 4T(k - 2)$ and generates a running time

of $O^*(2^k)$.

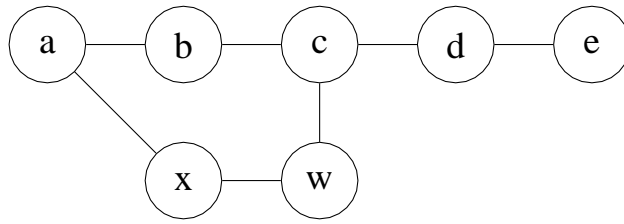


Figure 10: Neighbor of c at distance two from a .

Case 7. c has a neighbor at a distance two from a .

We consider in this case that c has a neighbor at a distance two from a and at a distance three from e as shown in 10.

For this case, the branching will be as follows:

- delete cd ;
- delete de ;
- delete ab and aw ;
- delete aw and wx ;
- delete aw and cx ;
- delete bc and aw ;
- delete bc and wx ;
- delete bc and cx ;

The recurrence of the above will be $T(k) = 2T(k - 1) + 6T(k - 2)$ and generates a running time of $O^*(3.646^k)$.

Case 8. c has a neighbor at distance one from a and three from e not passing through c .

We consider in this case that c has a neighbor at a distance one from a other than b and

at a distance two from d not passing through c as shown in 11.

For this case, the branching will be as follows:

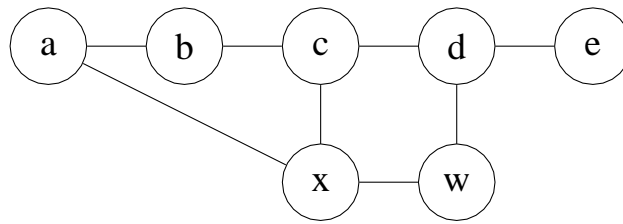


Figure 11: Neighbor of c at distance one from a and two from d .

- delete de ;
- delete ab and ax ;
- delete bc and ax ;
- delete cd and xw ;
- delete cd and dw ;

The recurrence of the above will be $T(k) = T(k - 1) + 4T(k - 2)$ and generates a running time of $O^*(2.562^k)$.

Case 9. c has a neighbor at distance two from a and two from d not passing through c .

We consider in this case that c has a neighbor at a distance two from a not passing through b and at a distance two from d not passing through c as shown in Figure 12.

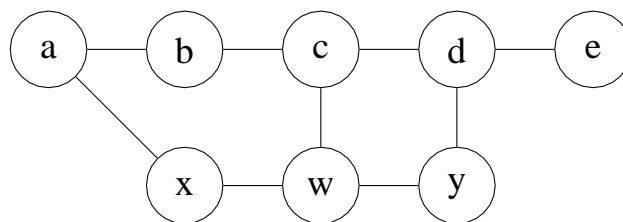


Figure 12: Neighbor of c at distance two from a and two from d .

For this case, the branching will be as follows:

- delete de ;
- delete ab and ax ;
- delete ab and xw ;
- delete bc and ax ;
- delete bc and xw ;
- delete cd and wy ;
- delete cd and dy ;

The recurrence of the above will be $T(k) = T(k - 1) + 6T(k - 2)$ and generates a running time of $O^*(3^k)$.

Taking all of the above branching cases into consideration, we conclude that:

Theorem 1. *The 3-Clubs Cluster Edge Deletion problem can be solved in $O^*(3.65^k)$.*

7.4 A Heuristic Approach for 3-Club Cluster Edge Deletion

In this chapter, we introduce and implement a heuristic approach to solve the 3-clubs Cluster Edge Deletion problem. The aim is to find a working solution within a reasonable time frame. The method to reach a solution is to cut as minimum edges as possible that would destroy all conflict quintuples in the graph. Therefore, this

algorithm is based on the *influence* level of an edge. The edge influence value indicates the number of P_4 s that this edge is part of. Hence, an edge influence designates the number of P_4 s that decreases in the overall graph whenever this edge gets deleted. Thus, to minimize the cost of edge deletions in the overall problem, we need to delete the edges with the highest influence first and subsequently transform graph G to a disjoint union of 3-clubs.

To calculate the influence value for the edges, we apply the following approach. We assign a vertex v as the source, then with Depth First Search we traverse the graph from v to its 4th-degree neighbors: vertices within distance 4 from v . For each edge, we count the number of P_4 s it is part of and increase its weight by this number. We loop through all the vertices, for each vertex being a source, we apply the depth-first search to count the number of P_4 s, then update the edges' weights accordingly. Obviously, this procedure takes quadratic time.

When the final weights have been allocated to the edges then the edge deletion process can begin. Starting with the highest weight edge deletion which will decrease the highest number of P_4 s in this current graph version. Because of this deletion, near edges' weights are affected, hence, after each deletion, a recalculation of weights is computed to the affected edges only and not to all graph edges. The affected edges when a deletion is applied are the ones connecting the vertices of the direct neighborhood, second-degree, and third-degree neighborhood. The algorithm will continue to iterate until the graph no longer contains conflict quintuples.

We do not assume in this algorithm any pre-existing assumptions about the number or structure of the clusters while assuming the input graph is undirected and unweighted.

Algorithm 3: Heuristic 3CCED Algorithm

Input: A graph $G = (V, E)$

Output: The minimum 3-club edge deletions

1: $\forall i : w_i \leftarrow 0$; //initialize all edges weight to 0.

- 2: Run a DFS algorithm for each $v \in V$.
 - 3: For each time a P_4 runs through an edge e , $w(e) = w(e)+1$.
 - 4: Delete the highest weighted edge.
 - 5: Recalculate the weights of the nearest edges only which are linked to vertices at distances 1,2, and 3 from the deleted edge.
 - 6: Repeat step 4 until no $w(e) > 0$ exist in the graph.
-

7.5 Experimental Analysis

The algorithm was implemented and tested on datasets used in the previous approaches of this work. The machine specs are the same as described previously. The implementation was written in CPP programming language, and the experimental results will follow.

In this experiment, the input graphs used are similar to the previous experiments, of orders ranging between 20 and 350. Table 4 below presents the results of the clustering measures with a comparison between the Heuristic 3-clubs clustering with edge deletion and the Cluster Editing [33] implemented and discussed earlier in this work. Again, in the presented results, the adopted metrics measured are:

- (i) Time of the job running in milliseconds;
- (ii) Cost: the number of edges deleted during the process; and
- (iii) Clusters: the number of components in the resulting graph, including singletons.

The results show that the 3-clubs clustering with edge deletion approach had close results in terms of the number of clusters in the output graphs, an average of 88% similarity with the Cluster Editing approach, yet the 3CCED running time was on average 4 times faster, and the cost of deleted edges was on average 17 times less than the Cluster Editing approach.

Another comparison was applied between the 3-clubs clustering with edge deletion algorithm and the previously discussed 2-clubs clustering with edge deletion FPT algorithm. Table 5 shows a 93% similarity (on average) in the resulting graphs' number of clusters and almost double the time and half the cost in favor of 3CCED being faster and less costly for the majority of the graphs.

We can conclude that adopting a more relaxed method of clustering than the disjoint union of cliques and even 2-clubs Clustering with Edge Deletion has reached close clustering results but with relatively faster time and lower cost.

Table 4: Experiments on different graphs using our 3CCED heuristic and the Cluster Editing algorithm

Graph		3CCED heuristic			Cluster Editing		
Nodes	Edges	Time(ms)	Cost	Clusters	Time(ms)	Cost	Clusters
20	97	10.2	1	2	80	46	1
144	1191	121	7	39	60	132	43
144	3089	271	15	13	274	78	20
159	425	51	6	56	289	42	55
184	441	54	6	51	326	72	64
216	3054	179	2	46	908	134	56
232	5582	334	0	31	177	20	34
263	1286	88	0	95	822	56	107
313	1764	121	0	106	883	100	117
350	1770	191	5	142	873	106	153

Table 5: Experiments on different graphs using our 3CCED heuristic and the 2CCED algorithms

Graph		3CCED heuristic			2CCED		
Nodes	Edges	Time(ms)	Cost	Clusters	Time(ms)	Cost	Clusters
20	97	10.2	1	2	0.311	2	3
144	1191	121	7	39	405.02	8	40
144	3089	271	15	13	261.89	6	14
159	425	51	6	56	210.13	7	57
184	441	54	6	51	1250.34	8	54
216	3054	179	2	46	74.67	13	51
232	5582	334	0	31	17.56	1	32
263	1286	88	0	95	18.50	5	99
313	1764	121	0	106	40.18	3	109
350	1770	191	5	142	1103.05	9	146

CHAPTER EIGHT

SUMMARY AND FUTURE WORK

In this work, we tackled the 2-CLUB CLUSTER EDGE DELETION in graphs which is the transformation of a graph G into a set of disjoint union of 2-clubs. The technical implementation of this work was based on the latest improved 2-CLUB CLUSTER EDGE DELETION algorithm which presented a running time of $O^*(2.695^k)$. This approach begins with the elimination of specific scenarios such as particular paths of length two or length four, bounded-degree-two, and the tail of length three, followed by the branching process. Branching is applied mainly by searching for specific cases and solving them faster than exhaustively. We also implemented a heuristic approach algorithm to transform graph G into a disjoint set of 2-clubs based on the edge's influence level. A performance cross-comparison has been applied between all three approaches: the implemented algorithms (heuristic and fixed-parameter), and the cluster editing algorithm that was top-ranked in Parameterized Algorithms and Computational Experiments Challenge (PACE) 2021. Results show that the heuristic approach compared to the 2CCED fixed-parameter algorithm is 4.5 times faster, on average, but might not be optimal in terms of effectiveness. But when compared to the Cluster Editing approach, the Heuristic is 30 times faster and 10 times lower in terms of cost while preserving similar resulting clusters. On the other hand, the 2CCED FPT algorithm is 9 times faster than the Cluster Editing approach with 12 times lower cost while maintaining a close output number of clusters.

In addition to the work on 2CCED, we have presented a new fixed-parameter

algorithm for 3-CLUB CLUSTER EDGE DELETION, which simply transforms a graph into a disjoint set of 3-clubs. Obviously, a 3CCED is the natural extension to the previous work on 2CCED, and a 3-club can be considered a reasonable diameter-relaxation of Clique. In this approach, we have reached a running time of $O^*(3.65^k)$. We also implemented a heuristic approach for the 3CCED problem using the proposed edge influence level approach. A performance comparison has been applied between the three approaches: heuristic algorithm for 3CCED, fixed-parameter algorithm for 2CCED, and Cluster Editing. Results show that the 3CCED is 4 times faster and 17 times lower cost than the Cluster Editing approach, and it is twice faster and half costly as the fixed parameter 2CCED approach. It shows that correlation clustering via 3-CLUB CLUSTER EDGE DELETION can be more relevant to practical work and show information as significant as the rigid Cluster Editing model.

For future work, it would be interesting to see the experimental implementation of our new fixed-parameter algorithm for 3-CLUB CLUSTER EDGE DELETION. Another direction would be exploring the vertex deletion approach that results in 3-clubs which can be useful when dealing with outliers in some practical areas.

BIBLIOGRAPHY

- [1] F. N. Abu-Khzam. On the complexity of multi-parameterized cluster editing. *Journal of Discrete Algorithms*, 45:26–34, 2017.
- [2] F. N. Abu-Khzam, N. E. Baldwin, M. A. Langston, and N. F. Samatova. On the relative efficiency of maximal clique enumeration algorithms, with applications to high-throughput computational biology. In *International Conference on Research Trends in Science and Technology*, pages 1–10, 2005.
- [3] F. N. Abu-Khzam, C. Bazgan, K. Casel, and H. Fernau. Clustering with lower-bounded sizes - A general graph-theoretic framework. *Algorithmica*, 80(9):2517–2550, 2018.
- [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, S. Gaspers, A. Shaw, and P. Shaw. Cluster editing with vertex splitting. In *International Symposium on Combinatorial Optimization*, pages 1–13. Springer, 2018.
- [6] F. N. Abu-Khzam, N. Makarem, and M. Shehab. An improved fixed-parameter algorithm for 2-club cluster edge deletion. *Theor. Comput. Sci.*, 958:113864, 2023.

- [7] R. D. Alba. A graph-theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, 3(1):113–126, 1973.
- [8] M. Almeida and F. Carvalho. The k-club problem: new results for k= 3. *Centro de*, 2008.
- [9] M. T. Almeida and F. D. Carvalho. Integer models and upper bounds for the 3-club problem. *Networks*, 60(3):155–166, 2012.
- [10] B. Balasundaram, S. Butenko, and S. Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1):23–39, 2005.
- [11] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.
- [12] J. R. Barr, P. Shaw, F. N. Abu-Khzam, and J. Chen. Combinatorial text classification: the effect of multi-parameterized correlation clustering. In *2019 First International Conference on Graph Computing (GC)*, pages 29–36. IEEE, 2019.
- [13] J. R. Barr, P. Shaw, F. N. Abu-Khzam, T. Thatcher, and S. Yu. Vulnerability rating of source code with token embedding and combinatorial algorithms. *International Journal of Semantic Computing*, 14(04):501–516, 2020.
- [14] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999.
- [15] S. Böcker and J. Baumbach. Cluster editing. In *Conference on Computability in Europe*, pages 33–44. Springer, 2013.
- [16] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.
- [17] A. Boral, M. Cygan, T. Kociumaka, and M. Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory of Computing Systems*, 58(2):357–376, 2016.

- [18] J.-M. Bourjolly, G. Laporte, and G. Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European journal of operational research*, 138(1):21–28, 2002.
- [19] R. Cook. Wcs data archives.
- [20] C. Crespelle, P. G. Drange, F. V. Fomin, and P. A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv preprint arXiv:2001.06867*, 2020.
- [21] M. Doucha and J. Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *MFCs*, volume 2012, pages 348–359. Springer, 2012.
- [22] M. M. A. El-Wahab, F. N. Abu-Khzam, K. Wang, and P. Shaw. Semi-exact exponential-time algorithms: an experimental study. In *2020 Second International Conference on Transdisciplinary AI (TransAI)*, pages 96–99, 2020.
- [23] 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.
- [24] A. Figiel, A.-S. Himmel, A. Nichterlein, and R. Niedermeier. On 2-clubs in graph-based data clustering: Theory and algorithm engineering. In *CIAC*, pages 216–230, 2021.
- [25] Gottesbüren. Parameterized algorithms and computational experiments, 2021.
- [26] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. In *Italian Conference on Algorithms and Complexity*, pages 108–119. Springer, 2003.
- [27] 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.

- [28] J. Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009.
- [29] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010.
- [30] P. Heggernes, D. Lokshtanov, J. Nederlof, C. Paul, and J. A. Telle. Generalized graph clustering: recognizing (p, q) -cluster graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 171–183. Springer, 2010.
- [31] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010.
- [32] L. Kellerhals, T. Koana, A. Nichterlein, and P. Zschoche. The pace 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- [33] kittobi1992. Kittobi1992/clusterediting: Kapoce is a framework for solving the cluster editing problem.
- [34] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.
- [35] J. Leskovec and A. Krevl. Snap datasets: Stanford large network dataset collection, 2014.
- [36] H. Liu, P. Zhang, and D. Zhu. On editing graphs into 2-club clusters. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 235–246. Springer, 2012.
- [37] D. Lokshtanov and D. Marx. Clustering with local restrictions. *Information and Computation*, 222:278–292, 2013.

- [38] N. Misra, F. Panolan, and S. Saurabh. Subexponential algorithm for d-cluster edge deletion: Exception or rule? In *International Symposium on Mathematical Foundations of Computer Science*, pages 679–690. Springer, 2013.
- [39] R. J. Mokken et al. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.
- [40] S. Pasupuleti. Detection of protein complexes in protein interaction networks using n-clubs. In *European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 153–164. Springer, 2008.
- [41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [42] 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.
- [43] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.
- [44] R. Sharan. *Graph modification problems and their applications to genomic research*. PhD thesis, Tel-Aviv University, 2002.
- [45] D. Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory of Computing Systems*, 65(2):323–343, 2021.
- [46] E. Ulitzsch, Q. He, V. Ulitzsch, H. Molter, A. Nichterlein, R. Niedermeier, and S. Pohl. Combining clickstream analyses and graph-modeled data clustering for identifying common response processes. *psychometrika*, 86(1):190–214, 2021.
- [47] A. Veremyev, V. Boginski, E. L. Pasiliao, and O. A. Prokopyev. On integer programming models for the maximum 2-club problem and its robust generalizations

in sparse graphs. *European Journal of Operational Research*, 297(1):86–101, 2022.

- [48] D. B. West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.
- [49] T. Wittkop, D. Emig, S. Lange, S. Rahmann, M. Albrecht, J. H. Morris, S. Böcker, J. Stoye, and J. Baumbach. Partitioning biological data with transitivity clustering. *Nature methods*, 7(6):419–420, 2010.