On Clique and Cluster Problems and their Applications

Zhuoli Xiao

June 30, 2018

Abstract

Clique-based problems are getting more and more attentions in the field of theoretical computer science. They can be used to model many real-world problems and provide solutions for them. Before this project, there are lots of papers on clique-base problems, but many of them are only discussing one specific one. Therefore, we do a project where we integrate those clique-based problems from those papers, research on their algorithms and running time, and discuss their applications.

The main method in this project is literature research. We research on existing FPT algorithms on Edge Clique Partition problem and Edge Clique Cover problem as well as rewriting some proofs for some lemmas and reduction rules in existing papers, since some proofs in those paper are missing or lacking important details. We also calculate the running times of FPT algorithms for these two problems, which are $O(kn^3 + (2^{k^2-2})^k 2^{k^2-2} k^4)$ for Edge Clique Partition and $O(n^4 + 2^{k^{k+2}})$ for Edge Clique Cover. Moreover, we investigate three new reduction rules and two new properties of Edge Clique Partition, which are not discussed in existing papers.

Then we introduce some existing applications for Edge Clique Cover on the fields of Hardware Designing and Experimental Data Analysis. We also propose two new applications for Edge Clique Partition on the fields of Graph Data Compression and Distributed Network. We not only show the efficiency of these applications but also prove their soundness in this project.

Contents

Chapter 1

Introduction

Nowadays, computer science is playing a more and more important role in many fields, such as natural sciences, engineering and social science. With the advances of technology, there are many ways how computer science impacts a vast variety of areas.

While a number of computational problems are solvable efficiently, in the general case[1]Why don't we put the footnode right after "efficiently", many such problems have the property to be intractable, that is they are not solvable efficiently in the general case. An important complexity class of intractable computational decision problems are the ones that are NP-hard. For any NP-hard problem there is no algorithm known that solves the problem for any given instanceI think it should be for all given instances. in polynomial time. Finding a polynomial time algorithm that solved an NP-hard problem would answer the famous "P=NP?"-question affirmative. Typical running times of algorithms for NP-hard problems are exponential in the size of the input. Much progress has been achieved to tackle NP-hard problems despite its intractability property. Techniques to deal with these problems include fixed-parameter algorithms [8, 9, 23], approximation algorithms [27], and heuristics [21]. Approximation algorithms[2] and heuristics[3] can be fast but do not necessarily deliver optimal solutions.

Parameterized complexity deals with NP-hardness (or classical intractability in general) and includes the design of fixed-parameter algorithms [19, 8, 9, 23]. In parameterized complexity, the problem complexity is studied using a second dimension, the *problem parameter*. Intuitively, problems are distinguished to be either tractable for the specific parameterization considered (such prob-

[1]A computational problem is understood to be solvable efficiently if it can be solved by an algorithm in polynomial time, for any input.

[2]An approximation algorithm is an algorithm that returns a solution with a guaranteed quality: the solution obtained is within a multiplicative factor of the optimal one [27].

[3]A heuristic algorithm is an algorithm that reaches a possible solution to the given problem with no optimization guarantee [21]

lems are said to be *fixed-parameter tractable* or members of FPT, the class of fixed-parameter-tractable parameterized decision problems) or intractable (that is they are shown to be *hard* for a parameterized class such as complexity class W[1], a superclass of FPT).

To understand membership in FPT, we consider a decision problem that is associated with one (or more) parameter(s). For simplicity, assume the problem parameter is called k, and an instance to our parameterized problem is described as $I =< X, k >$. Then our parameterized problem is in FPT if it can be solved in time $O(n^{O(1)} + f(k))$, where $n = |X|$ is the size of problem instance and $f(k)$ is a function that depends only on the parameter k, that is it is independent of n.

Note that, while function $f(k)$ can be exponential, when the value of k is small and f does not grow too fast, such a running time can indeed describe a practical algorithm for a large number of instances of substantial size. Problems that are hard or complete for W[1] typically have running times of $\binom{n}{k}$. A parameterized problem that is hard for W[1] is assumed to be not in FPT unless FPT=W[1], which is assumed to be false for most computer scientists (just like P=NP is assumed to be false). For further reading on parameterized intractability, see e.g. [19, 8, 7].

A rich toolkit of algorithm-design techniques has been developed for fixed-parameter algorithms of problems that are in FPTCan we say for sovling FPT problems. This toolkit includes the use of polynomial time *reduction rules*, *kernelization* and the technique of *bounded search trees*.

In this report, we concentrate on computational problems discussed in the literature that benefit from determining large complete or dense subgraphs. We will investigate their solution approaches and their applications. Complete subgraphs are also called *cliques*.

The goal of the clique problem is to determine, for a given graph, whether there exists a clique of size at least k. While this classic problem when parameterized by k, is complete for class W[1][4], a number of its variants are proven to be in FPT [19].

Our main focus in this report is to use approaches to deal with clique-based graph problems as a tool to assist solving problems in other fields. We exhibit some existing types of clique-based or cluster-based graph problems, especially those in the class of FPT. We discuss the complexity of these problems as well as some of their algorithmic solutions. Then we will survey some existing applications of cliques on fields of hardware design, bioinformatics, social networks and data mining. We model some *FPT* problems from those fields as clique-based

[4]A parameterized decision problem is complete for class W[1] if it is (1) hard for class W[1] and (2) a member of W[1].

FPT problems, shrink the graph with $r$eduction rules and search for the exact solutions with $b$ounded search tree algorithm. For each modelled problem, we also analyze how much we can improve its running time by reducing them into clique-based problems. this last sentences need to be looked at once the report is pretty much done.

Chapter 2

Background and Terminology

In this chapter, we introduce terminology required in this document, including basics in graph theory. For further resources on terminology used in the report see [4, 12, 8].

Graph: All *graphs* in this report, typically denoted as $G = (V, E)$ with vertex set V and edge set E, are *undirected simple graphs*, unless otherwise stated.

Open Neighborhood: Given a graph $G = (V, E)$ and a vertex $v \in V$, the *open neighborhood* of v in G is defined as $N(v) = \{w : v \neq w, vw \in E\}$.

Closed Neighborhood: Given a graph $G = (V, E)$ and a vertex $v \in V$, the *closed neighborhood* of v in G is defined as $N[v] = N(v) \cup \{v\}$.

Induced Subgraph: For a graph $G = (V, E)$ with $V' = \{x_1, x_2, ..., x_m\} \subseteq V$, $G(x_1, x_2, ..., x_m) = (V', E')$ denotes the graph that is the by V' *induced subgraph* of G. That is, for $G(x_1, x_2, ..., x_m)$, for every pair of vertices $x, y \in V'$: $xy \in E'$ if and only if $xy \in E$. In this project report, $E(V')$ denotes the set of edges in the by V' *induced subgraph*.

Complete Graph/Clique: A graph $G = (V, E)$ is *complete* if and only if for every pair of vertices $u, v \in V$ there is an edge $uv \in E$. The vertex set of a complete (sub)graph is also called *clique*. A singleton is also considered to be a *clique*.

Note that graphs containing no vertices or graphs that contain exactly one vertex are also complete graphs.

Maximal Clique: A clique C is maximal if and only if it cannot be extended by including any vertex v such that $v \in V$ but $v \notin C$. In other words, a maximal clique C is not a proper subset of any other clique in G.[2].

K_n **Graph**: The complete graph with n vertices is denoted K_n.

K_n**-free Graph**: A K_n*-free* graph is a graph that does not contain a complete subgraph consisting of n vertices.

Bipartite Clique: a *bipartite clique* B is a subset of vertices from graph $G = (V, E)$ such that the vertices in B can be divided into two disjoint sets V_1, V_2, with:

1. $V_1 \cup V_2 = B$.
2. $V_1 \cap V_2 = \emptyset$.
3. Every vertex in V_1 is connected to every vertex in V_2 in graph G.
4. For every pair of vertices $u, v \in B$, if $u, v \in V_1$ or $u, v \in V_2$, then $uv \notin E$.

Edge Clique Partition: Given a graph $G = (V, E)$, an *edge clique partition* of G is a set $CP = \{C_1, C_2, ..., C_l\}$, where l is a positive integer, such that:

1. Each $C_i \in CP$ is a clique in G.
2. For every edge $uv \in E$: $u, v \in C_i$ for exactly one $C_i \in P$.

Non-trivial Clique Partition: A *non-trivial clique partition* of a complete graph G is a clique partition of size at least two, that is a clique partition that contains two or more cliques.

Bipartite Clique Partition: Given graph $G = (V, E)$, a *bipartite clique partition* of graph G is a set $BP = \{B_1, B_2, ..., B_k\}$ such that:

1. $B_1 \cup B_2 \cup ... \cup B_k = V$.
2. Each $B_1, B_2, ..., B_k$ is a bipartite clique of G.
3. For every edge $uv \in E$, there is a unique $B_i \in BP$ which contains both of its endpoints u and v.

Dominating Set: Given a graph $G = (V, E)$, a *dominating set* is a subset $V' \subseteq V$ such that for every $v \in V$, there exists a vertex $v' \in N[v]$ with $v' \in V'$.

Polynomial-time Reduction: Given decision problems X and Y, a *polynomial-time reduction* from X to Y is a polynomial-time algorithm A that converts any instance I of X into an instance $I' = A(I)$ of Y, such that I' is a yes-instance for Y if and only if I is a yes-instance for X.

Note that $I =< X, k >$ is a yes-instance if and only if there exists a solution S to the given problem X, where the size of S is at most k.Sometimes larger than k, e.g. k-clique

Class NP (class of *non-deterministic polynomial time* problems): NP is the complexity class of all decision problems whose *yes-answers* can be *verified* in polynomial time.

NP-hardness: *NP-hard* problems are the decision problems that are at least as hard as the hardest problems in NP. That is, every problem in NP can be polynomial-time reduced to every NP-hard problem.

NP-Completeness: NP-complete problems are the hardest problems in NP. That is a decision problem is called *NP-complete* if it is NP-hard and a member of NP.

Parameterized decision problem: A *parameterized* decision problem is denoted as $< X, k >$, where X denotes the decision problem and k denotes its parameter.

Parameterized Complexity Class FPT (Fixed-parameter tractability): A parameterized decision problem $< X, k >$ is *fixed-parameter tractable*, or a member of FPT, if there exists an algorithm A that solves every instance I of $< X, k >$ in running time $O(|X|^{O(1)} + f(k))$ or $O(|X|^{O(1)} \cdot f(k))$. A is called a *fixed-parameter algorithm.*

Reduction Rules: A *reduction rule* is a polynomial-time mapping from a given parameterized instance I to a new parameterized instance I', where I' is a yes-instance if and only if I is a yes-instance.

Kernelization [22]: A *kernelization* is a polynomial time reduction that maps a given instance $I =< G, k >$ to a new instance $I' =< G', k' >$, such that:

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

I' is called *(problem) kernel.* Note that if $g(k)$ is polynomial in k then we also call I' a *polynomial (problem) kernel.*

Bounded Search Trees [6]: The *technique of bounded search trees* is an algorithm-design technique that solves a parameterized decision problem $< X, k >$ by using a systematic enumeration of candidate solutions in fixed-parameter-tractable time. The resulting search tree is bounded in size in a function $g(k)$ that depends on parameter k only. The root of the (bounded) search tree corresponds to the original problem $< X, k >$.

Typically, when applying the technique of bounded search trees, the problem is solved recursively by branching according to a constant number constant number of cases? of different cases. When using a recursive bounded-search-tree algorithm, the size of the search tree $g(k)$ is a function that depends on k and its largest *branching number*, defined below.

Branching Vector: When applying the technique of bounded search trees, whenever branching is applied to an instance $I =< X, k >$, the instances resulting from the branching are represented as the children of I in the search tree. If a branching rule applied to I creates h children that reduce their respective parameter to $k - d_1, k - d_2, ...$ and $k - d_h$, respectively, then this branching rule has a *branching vector* of $(d_1, d_2, ..., d_h)$.

Note that branching vector $(d_1, d_2, ..., d_h)$ corresponds to the recursion $T_k = T_{k-d_1} + T_{k-d_2} + \cdots T_{k-d_h}$ and its corresponding characteristic polynomial is $z^d = z^{k-d_1} + z^{k-d_2} + ... + z^{k-d_h}$, where $d = max\{d_1, d_2, ..., d_h\}$. If $\alpha \in \mathcal{R}$ is a root of maximum absolute value solving this characteristic polynomial, we call α the *branching number* of branching vector $(d_1, d_2, ..., d_h)$. Furthermore, $t_k = O(\alpha^k)$ is the size of this search tree.

Below, we use the k-VERTEX COVER problem as an example to demonstrate the technique of bounded search trees. We also highlight the corresponding branching vector, branching number, and search-tree size.

k-VERTEX COVER
Instance: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does there exist a set $V' \subseteq V$ with $|V'| \leq k$ such that every edge is *covered* by V' (i.e. for each edge uv, at least one of u and v is in V')?

Consider the following branching rule from the bounded-search-tree algorithm for k-Vertex described in [24]:

Branching Rule 1. *: Let G be a graph where all vertices have degree at least two and no vertex has degree larger than six. Assume $x \in V$ is a vertex of degree 2 in G with $N(x) = \{a, b\}$No need to state degree 2, where $\deg(a) \geq 3$, $ab \notin E$, and there is no vertex $c \in V$, $\deg(c) = 2$ that builds a cycle x, a, c, b in G x is the only common neighbour. If $|N(a) \cup N(b)| \geq 4$, then branch according to $\{a, b\}$ and $N(a) \cup N(b)$.*

Observation [24]: The branching vector resulting from Branching Rule 1 is $(2, 4)$.

Note that branching vector $(2,4)$ describes how much the value of parameter k is reduced in each of the two branches. To be more precise, when given instance of vertex cover problem $I =< G, k >$, if Branching Rule 1 is applied, we obtain two branches of instances as follows:

1. $I_1 =< G_1, k-2 >$ with $G_1 = G - \{a,b\}$
2. $I_2 =< G_2, k - |N(a) \cup N(b)| >$ with $G_2 = G - (N(a) \cup N(b))$

Note that the value of k_1 in Branch One is $k_1 = k-2$ since we include a, b into vertex cover V', and the value of k_2 in Branch 2 is at most $k_2 = k-4$ since we include $N(a) \cup N(b)$ into V', where $|N(a) \cup N(b)| \geq 4$. Therefore, the branching vector of this branching rule is $(2,4)$.

Now we prove the soundness of Branching Rule 1 for k-Vertex Cover.

Proof. To prove soundness for this branching rule we must show that I is a yes-instance for k-Vertex Cover if and only if I_1 or I_2 is a yes-instance for k-Vertex Cover.

We first prove that if I is a yes-instance then at least one of I_1 and I_2 is a yes-instance by contradiction.

Assume I is a yes-instance and V' is a vertex cover of size at most k for I. Further assume both I_1 and I_2 are no-instances. Let $E' \subseteq E$ be the set of edges incident to vertices a and b.

For an edge $e \in E'$, one of its endpoints is in $\{a,b\}$ and the other is in $N(a) \cup N(b)$. If neither I_1 or I_2 is a yes-instance, then including any one e's endpoint in V' leads to
Below is old. Let us consider Case 1. Then $V_1' = V' - \{a,b\}$ is a vertex cover of size $k-2$ for graph G_1. Therefore, I_1 is a yes-instance since

In Case 2, $(N(a) \cup N(b)) \subseteq V'$ $(a, b \notin V')$. Then $V_2' = V' - (N(a) \cup N(b))$ is a vertex cover of $G_2 = G - (N(a) \cup N(b))$ with $V_2' = k - |N(a) \cup N(b)|$. Therefore, I_2 is a yes-instance.

In Case 3, $a \in V'$ and $N(b) \subset V'$ $(b \notin V')$. In this case, a and $x \in N(b)$ both cover edge ax, x covers edge bx. The remaining edges in E' are covered by a and the vertices in $N(b) - b$. If we substitute in the vertex cover x by b, all edges in G are still covered with the same number of vertices. Thus, we can assume that whenever an instance satisfies Case 3, then it also satisfies Case 1 with $V" = (V' - \{x\}) \cup \{b\}$ and $|V"| = |V'|$. Therefore, I_1 is a yes-instance.

Case 4 can be argues similar to Case 3: Also here, I_1 is a yes-instance since.

Therefore, if I is a yes-instance, at least one of I_1 and I_2 is a yes-instance.

We now prove that if I_1 or I_2 is a yes-instance, then I is a yes-instance.

First, assume I_1 is a yes-instance. We show that then also I is a yes-instance. Note that, since I_1 is a yes-instance, it has a vertex cover of size at most $k-2$. Considering G, we now argue that all edges incident to set $\{a, b\}$ can be covered by no more than 2 additional vertices than the vertices in vertex cover V_1' for graph G_1, since $G_1 = G - \{a, b\}$. $V' = V_1' \cup \{a, b\}$ satisfies this condition.Therefore, I is a yes-instance.

Finally, we prove that if I_2 is a yes-instance, then also I is a yes-instance. Note that, since I_2 is a yes-instance, it has a vertex cover of size at most $k - |(N(a) \cup N(b))|$. Considering G, we argue that all edges incident to set $\{a, b\} \cup (N(a) \cup N(b)$ can be covered by including $N(a) \cup N(b)$ into the vertex cover. Thus, I is a yes-instance.

Therefore, if I_1 or I_2 is a yes-instance, there exists a vertex cover of size at most k for instance G, this, I is a yes-instance.

□

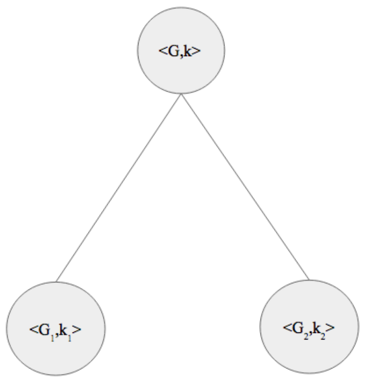


Figure 2.1: Branches on Rule 1, where $G_1 = G - \{a, b\}$, $k_1 = k - 2$ and $G_2 = G - (N(a) \cup N(b))$, $k_2 = k - |N(a) \cup N(b)|$.

The corresponding branching number of Branching Rule 1 is 1.272020, which is decided by its branching vector $(2, 4)$ and computed using its corresponding characteristic polynomial (described above).

Chapter 3

Problem Definitions

In this chapter, we list the definitions of graph-based parameterized decision problems that we discuss in this report. Many of their classic versions can be found in [12].

k-Clique [8]

Input:	A graph $G = (V, E)$, a positive integer k
Parameter:	k
Question:	Does there exist a clique $C \subseteq V$ of size at least k, that is $\lvert C\rvert \geq k$?

k-Edge Clique Partition [8]

Input:	A graph $G = (V, E)$, a positive integer k
Parameter:	k
Question:	Does there exist an *edge clique partition* $CP = \{C_1, C_2, \ldots, C_l\}$ of size $l \leq k$ where each $C_i \in CP$ is a clique in G, such that for each edge $uv \in E$, there is exactly one $C_i \in CP$ with $u, v \in C_i$?

k-Vertex Clique Partition

Input:	A graph $G = (V, E)$, a positive integer k
Parameter:	k
Question:	Does there exist a *vertex clique partition* $VP = \{C_1, C_2, \ldots, C_l\}$ of size $l \leq k$ where each $C_i \in VP$ is a clique in G, such that for each vertex $v \in V$, v is in exactly one $C_i \in VP$?

k-Edge Clique Cover [8]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does there exist an *edge clique cover* $CC = \{C_1, C_2, \ldots, C_l\}$ of size $l \leq k$ where each $C_i \in CC$ is a clique in G, such that for each edge $uv \in E$, $u, v \in C_i$ for some $C_i \in CC$?

k-Annotated Edge Clique Cover [15]
Input: A graph $G = (V, E)$, a set $A \subseteq E$, a positive integer k
Parameter: k
Question: Does there exist an *annotated edge clique cover* $ACC = \{C_1, C_2, \ldots, C_l\}$ of size $l \leq k$ where each $C_i \in ACC$ is a clique in G, such that for each edge $uv \in E - A$, $u, v \in C_i$ for some $C_i \in ACC$?

k-Vertex Clique Cover [8]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does there exist a *vertex clique cover* $VCC = \{C_1, C_2, \ldots, C_l\}$ of size $l \leq k$ where each $C_i \in VCC$ is a clique in G such that for each vertex $v \in V$, $v \in C_i$ for some $C_i \in VCC$?

k-Compression Clique Cover [8]
Input: A graph $G = (V, E)$, an edge clique cover CC for G of size $k + 1$ for G and a positive integer k
Parameter: k
Question: Does there exist an edge clique cover CC' for G of size at most k?

k-Bipartite Clique Partition [25]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does there exist a *bipartite clique partition* $BP = \{B_1, B_2, \ldots, B_l\}$ of size $l \leq k$ such that for each edge $uv \in E$, there is exactly one bipartite clique $B_i \in BP$ with $u, v \in B_i$?

k-Bipartite Clique Cover [25]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does there exist a *bipartite clique cover* $BC = \{B_1, B_2, \ldots, B_l\}$ of size $l \leq k$ such that for each edge $uv \in E$, $u, v \in B_i$ for some bipartite clique $B_i \in BC$?

k-Clique Editing [8]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Is it possible to use at most k operations, consisting of edge additions or edge deletions, to edit G into a union of vertex-disjoint cliques?

k-Clique Deletion [8]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Is it possible to use at most k edge deletions to edit G into a union of vertex-disjoint cliques?

k-Dominating Clique [8]
Input: A graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does there exist a *dominating clique* $DC \subseteq V$ of size at least k such that DC is both a dominating set and a clique in G?

k-Multicolored Clique [8]
Input: A graph $G = (V, E)$ with a vertex coloring $\chi : V(G) \to [k]$, a positive integer k
Parameter: k
Question: Does there exist a *multicolored clique* $MC = \{v_1, v_2, \dots, v_l\}$ such that MC is a clique and for each $v_i, v_j \in MC$, $\chi(v_i) \neq \chi(v_j)$ if and only if $i \neq j$?

Chapter 4

Algorithms for Clique-based Problems

In our literature research, we have found that many clique-based problems are NP-complete, and there are no known polynomial-time algorithms to solve them. While k-Clique, the classic Clique problem parameterized by the clique size k is W[1]-complete, some interesting parameterizations of these NP-hard clique-based problems are members of the parameterized complexity class FPT. Techniques from the FPT toolkit such as kernelization or bounded-search-tree algorithms can be used to solve such problems to achieve a running time in the form of $O(f(k) * n^c)$ or $O(f(k) + n^c)$ for parameter k, input size n, and constant c. Examples of three clique-based problems that are in FPT, as well as their running times, are listed below. Here, for input graph G the number of vertices and edges are denoted by n and m, respectively.

1. k-Edge Clique Partition: $O(kn^3 + 2^{k^2-2})^k 2^{k^2-2} k^4)$ for general graphs [22].
2. k-Edge Clique Cover: $O(n^4 + 2^{k^{k+2}})$ [15].
3. k-Clique Editing: $O(1.62^k + m + n)$ [5].

In below sections we describe fixed-parameter algorithms for k-Edge Clique Partition and k-Edge Clique Cover problems: reduction rules followed by search-tree algorithms.

When introducing clique-based problems and algorithms in this chapter, we assume the vertices and the edges are stored in adjacency lists. Therefore, it takes time $O(1)$ to search for a vertex and time $O(n)$ to search for an edge.

Note that we assume the indices of the array correspond to the names of the vertices in G (see Figure 4.1,). Furthermore, in a list with array index u, there exists a node with value v if and only if $uv \in E$.

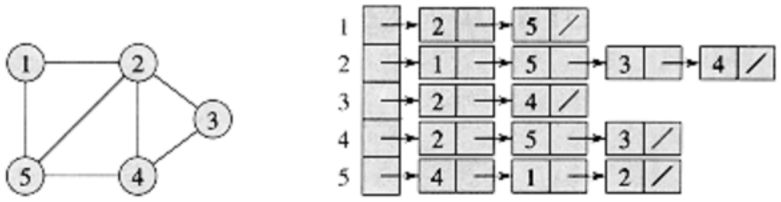


Figure 4.1: A graph $G = (V, E)$ and its corresponding adjacency list.

We use hash tables to determine the intersection of the neighbourhoods of two vertices u and v as below, which can be done in time $O(n)$.

1. Initialize a hash table ht, an empty set R of vertices.
2. Push all vertices $N[u]$ on ht.
3. For every $w \in N[v]$, if w in ht, add w in R.
4. Return R, which is the intersection of $N[u]$ and $N[v]$.

4.1 Algorithms for k-Edge Clique Partition

As defined in Chapter 3, when given an input to k-Edge Clique Partition, a solution CP of size k for G requires that for each edge in input graph G both of its endpoints are included in the same clique in CP. Furthermore, each edge is part of exactly one clique in clique partition CP.

In this section, we introduce a fixed-parameter algorithm by Mujuni and Rosamond [22] that solves k-Edge Clique Partition. This algorithm combines polynomial time reduction rules that reduce a given an instance $I =< G, k >$ with $|V| = n$ to a kernelized instance that contains at most k^2 vertices, with a bounded search-tree algorithm. We also give an upper bound of the algorithm's running time.

4.1.1 Reduction Rules and a k^2 Kernal for Edge Clique Partition

Following are four reduction rules selected from Section 2 in [22]. We also prove their correctness, which is not provided in [22].

Reduction Rule 1. *Let $I =< G, k >$ be an instance for k-*Edge Clique Partition*. If there exists a vertex $v \in V$ with $deg(v) = 0$, then remove v from G.*

Proof. Given $I =< G, k >$, and let v be a vertex in G with $deg(v) = 0$. Furthermore, let $I' =< G - v, k >$ be the instance obtained from instance $I =< G, k >$ and singleton v for k-Edge Clique Partition by applying Rule 1. We prove that I is a yes-instance if and only if I' is a yes-instance.

First, we show that if I is a yes-instance then I' is a yes-instance. Consider $I =< G, k >$, $G = (V, E)$, and vertex $v \in V$ with $deg(v) = 0$. Let CP be an edge clique partition for I of size k. Note that since v is a singleton in G, $v \notin C_i$ for any $C_i \in CP$. Therefore, CP is also an edge clique partition of size k for instance I'.

We next prove that if $I' =< G', k >$ with $G' = (V - \{v\}, E')$ and $E' = E$ is a yes-instance then I is a yes-instance. Assume I' is a yes-instance and CP' is a solution of size k for I'. Since $E' = E$, CP' is also a solution for I. □

For the following reduction rule we assume that Rule 1 is not applicable to instance I.

Reduction Rule 2. *Let $I =< G, k >$ be an instance for k-*Edge Clique Partition*. If there is a vertex $v \in V$ with $deg(v) = 1$, then remove v from G and decrease k by one.*

Proof. Given $I =< G, k >$, let v be a vertex in G with $deg(v) = 1$. Furthermore, let $I' =< G - v, k - 1 >$ be the instance obtained from $I =< G, k >$ and pendant vertex v by applying Rule 2. We prove that I is a yes-instance if and only if I' is a yes-instance.

First we show that if I is a yes-instance then I' is a yes-instance. Given $I =< G, k >$, let $N(v) = \{u\}$. Let CP be an edge clique partition for I, where CP is of size at most k. Note that there exists a $C_i \in CP$ with $C_i = \{u, v\}$, since edge u and v cannot both be part of a larger clique in G's edge clique partition since u and v have no common neighbours, but every edge must be included in some clique in CP. Therefore $CP' = CP - \{C_i\}$ is an edge clique partition for I' of size at most $k - 1$, since G' does not contain edge uv.

We next prove that if I' is a yes-instance then I is a yes-instance. Let CP' be a solution of size at most $k - 1$ for I'. Then $CP = CP' \cup \{\{u, v\}\}$ is a solution for I, since uv is the only edge that is not included in partition CP'. □

For the following reduction rule we assume that neither Rule 1 nor Rule 2 can be applied to instance I.

Reduction Rule 3. *Let $I =< G, k >$ be an instance of k-*Edge Clique Partition*. If there exists an edge $uv \in E$ with $N(u) \cap N(v) = \emptyset$, then remove edge uv from G and decrease k by one.*

Proof. Given $I =< G, k >$, $G = (V, E)$, with $uv \in E$ with $N(u) \cap N(v) = \emptyset$, let $I' =< G - uv, k - 1 >$ be the instance obtained from I and uv by applying Rule 3. We prove that I is a yes-instance if and only if I' is a yes-instance.

First we show that if I is a yes-instance then I' is a yes-instance. Assume CP is a solution for instance I of size at most k. Because $N(u) \cap N(v) = \emptyset$, there is a $C_i \in CP$ with $C_i = \{u, v\}$. Then $CP' = CP - \{C_i\}$ is a solution of size at most $k - 1$ for I'.

We next prove if I' is a yes-instance then I is a yes-instance. Let CP' be a solution of size at most $k - 1$ for I'. Then $CP = CP' \cup \{\{u, v\}\}$ is a solution of size at most k for I since uv is the only edge in G that is not covered by CP'. □

In order to prove the correctness of the next reduction rule, Rule 4 described below, we consider the following lemma. For this, recall that a non-trivial edge clique partition of a complete graph is of size at least two (Section 2).

Lemma 1. *Let $G = (V, E)$ be a complete graph with $n > 2$ vertices. Then the smallest non-trivial edge clique partition of G is of size at least n.*

Proof. We prove this lemma by induction on n, the number of vertices in the graph.

Hypothesis: let G be a complete graph with $n > 2$ vertices. Any non-trivial edge clique partition for G is of size at least n.

Base Case: Let $n = 3$, assume vertices in G are v_1, v_2, v_3. The smallest non-trivial edge clique partition for G is $\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\}$.

Induction Step: We prove, by contradiction, that the smallest non-trivial edge clique partition of a complete graph $G' = (V', E')$ with $n + 1$ vertices is of size at least $n + 1$.

Let CP' be a smallest non-trivial edge clique partition of size h for $G' = (V', E')$, and let $v \in V'$. Note that each clique in CP' is of size at least 2.

Furthermore let $G^* = G' - v$. Then G^* is a complete graph of size n and therefore for G^* the hypothesis holds. That is G^* has a non-trivial edge clique partition of size at least n.

Now we prove that if there is an edge clique partition CP' of size h for G', then there exists an edge clique partition CP^* of size at most h for G^* obtained by removing v from every clique in CP'.

For any clique C, removing any vertex v and v's incident edges again a clique, namely $C - v$. Furthermore, removing v from G' and from all cliques in partition CP' does not create any edges that are not partitioned by CP^*. Note that if $v \in C$, with $|C| = 2$ and $C \in CP'$, then when creating CP^* from CP', we must remove C instead of including a size-one clique in CP^*. Therefore $|CP^*| \leq |CP'|$.

Let us assume that $h < n + 1$. We show a contradiction for each of the two cases: $h < n$ and $h = n$.

Case 1: $h < n$. By removing v from every clique in CP' we obtain a clique partition CP^* for G of size smaller than n, which contradicts the hypothesis that any non-trivial edge clique partition for G^* is of size at least n.

Case 2: $h = n$. Assume there is a CP' of size n for G', and let $S' = \{C'_1, C'_2, \ldots C'_l\}$, $l \leq n$, be the subset of CP' which is composed of all cliques from CP' that contain vertex v.

Define $S^* = \{C^*_1, C^*_2, \ldots, C^*_l\}$ as the set by removing v from cliques in S'. To be more precise, $C^*_1 = C'_1 \backslash \{v\}$, $C^*_2 = C'_2 \backslash \{v\}$,. . .,$C^*_l = C'_l \backslash \{v\}$. We have proved above that there exists a solution CP^* for G^* such that $|CP^*| \leq |CP'|$ by removing v from every clique in CP'. Then S^* is a subset of CP^*. S^* is of size at least two. Otherwise every vertex $v \in V$ is in a clique C and $V \backslash C = \emptyset$, which leads to a trivial edge clique partition for G.

Note that if removing v from every $C' \in CP'$ leads to a trivial clique partition $CP^* = \{C\}$ for G, then $C \in CP'$ and each edge $u^*v \in E'$ such that u^* is also in V is partitioned in a different clique in CP'. There are n such edges. Thus CP' is of size at least $1 + n$.

Now we prove CP' is of size at least $n + 1$ if removing v leads to a clique partition CP^* such that $|CP^*| \geq 2$ for G^*. Based on our definition of edge clique partition, each clique in CP^* and CP' is of size at least two. Let C^*_i be such a clique that $C^*_i \in S^*$ and $|C^*_i| \geq 2$. Since S^* is of size at least two, if we can prove that to partition out-coming edges from C^*_i, we need at least $n - 1$ additional cliques, then the size of CP' is at least $2 + n - 1 = n + 1$.

Assume $|C^*_i| = m$ that $m \geq 2$. There are $n - m$ other vertices in G^* but not in C^*_i. Assume $v^*_1, v^*_2 \in C^*_i$. There are $n - m$ edges connecting v^*_1 and other $n - m$ vertices not in C^*_i. Denote them with a set E_1. And there are also $n - m$ edges connecting v^*_2 and other $n - m$ vertices not in C^*_i. Denote them with a set E_2. Note that $E_1 \cap E_2 = \emptyset$.

For every pair of edges $e_1 \in E_1$ and $e_2 \in E_2$, e_1 and e_2 is not partitioned by the same clique, otherwise pair of vertices v^*_1 and v^*_2 are in two different cliques from CP'.

Consider following edges:

1. e_{1f} connects vertices v_1 and f that $f \notin C_i^*$.
2. e_{1g} connects vertices v_1 and g that $g \notin C_i^*$.
3. e_{2f} connects vertices v_2 and f that $f \notin C_i^*$.
4. e_{2g} connects vertices v_2 and g that $g \notin C_i^*$.

It takes one clique to partition edges e_{1f} and e_{1g}, since we can partition both edges with a clique $C_0' \in CP'$ that $v_1, f, g \in C_0'$. However, it takes two cliques to partition edges e_{2f} and e_{2g}, since f, g is in C_0' and cannot appear in any clique which is not C_0'.

For this reason, in an minimum solution CP' it takes exactly one clique to partition out-coming edges from vertex v_1 and each of the rest out-coming edge takes one cliques to be put in the partition.

Therefore to partition all the out-coming edges from C_i^*, we need at least $f(j) = 1 + (j-1)(n-j) = 1 + \frac{n^2-1}{4} - (j - \frac{(n+1)^2)}{2})$ cliques. $f(j)$ takes its minimum value $n-1$ when $j = 2$. Therefore CP' is of size at least $2 + n - 1 = n + 1$, which is a contradiction.

Now we have shown that both Case 1 and Case 2 yield a contradiction. Therefore the smallest non-trivial edge clique partition of a complete graph $G' = (V', E')$ with $n + 1$ vertices is of size at least $n + 1$.

□

For the following reduction rule we assume that none of Rules 1–3 can be applied.

Reduction Rule 4. *Let $I =< G, k >$ be an instance of k-*Edge Clique Partition*. If there exists a vertex $v \in V$ with $|N[v]| > k$ where $N[v]$ induces a clique, then remove $E(N[v])$ from G, and reduce k by one.*

Proof. Assume v is a vertex in G such that $|N[v]| > k$ and vertices in $N[v]$ induces a clique. Furthermore, let $I' =< G - E(N[v]), k-1 >$ be the instance obtained from applying Rule 4 to I and v. We prove that I is a yes-instance if and only if I' is a yes-instance.

First we show that if I is a yes-instance then I' is a yes-instance. Assume CP is a solution of size at most k for I. Further assume C is a clique in G where $|C| > k$. Then the subgraph induced by C, denoted as $G(C)$, is a complete graph. According to Lemma 1, if we want to do a non-trivial partition on $G(C)$, we need at least $k + 1$ cliques. Therefore the only way to put all the edges in $E(C)$ in partition is to include C as a member into CP.

Therefore, if there is a vertex $v \in V$ with $|N[v]| > k$ where vertices in $N[v]$ induces a clique, then $C_i = N[v]$ is a member of CP. Then $CP - \{C_i\}$ partitions all edges in $E - E(N[v])$, and $CP' = CP - \{C_i\}$ is a solution of size at most $k-1$ for I'.

We next prove that if I' is a yes-instance then I is a yes-instance. Given instance $I' =< G - E(N[v]), k-1 >$, let CP' be a solution of size at most $k-1$ for I'. Then $CP' \cup \{N[v]\}$ is a solution of size at most k for $I =< G, k >$ since clique $N[v]$ partitions all the edges in $E(N[v])$. $\square$

We now prove the running time of above reduction rules, and then prove the size of problem kernel when none of these rules is applicable.

Lemma 2. *In time $O(kn^3)$, one can use Rules 1–4 to generate a reduced instance where none of these rules applies.*

Proof. First, we only consider the application of Rule 1. Given a singleton in G, it takes constant time to remove it from G. The removal of singletons does not decrease the degree of any vertex in G. Therefore to determine all singletons in a given graph G (and their removal) can be done in time $O(n)$.

Then we consider the running time of applying Rule 1 and Rule 2 together using below algorithm:

1. Initialize an empty queue Q.
2. Inspect $deg(v)$ for all $v \in V$. If $deg(v) = 0$, remove v from G. If $deg(v) = 1$, add v to Q.
3. Keep dequeuing first vertex v' from Q, remove v' from G and reduce k by one. Assume u' is the only neighbor of v'. Update $deg(u')$ after the removal of v'. If $deg(u') = 0$, remove u' from Q and G. If $deg(u') = 1$, add u' to the rear of Q. Stop when Q is empty or k decreases to zero.

In above algorithm, we look up the degree of a vertex at most $2n$ times in total. The first n times happen when we look up the degree of every vertex and enqueue the degree-one vertices into Q. Then when removing a vertex v' from a vertex in Q, we look up the degree of its neighbor u', where u' is not in queue Q. Note that there are at most n vertices in Q. Therefore, we need to inspect at most n vertices in step 3. Overall, all look ups of degrees of vertices can be done in linear time.

After applying Rule 1 and Rule 2 in time $O(n)$, assuming that none of the Rules 1 and 2 can be applied any longer, there are neither singletons nor degree-one vertices in I.

We now consider the application of Rule 3.

To apply Rule 3, we need to determine an edge uv with $N(u) \cap N(v) = \emptyset$. For each edge $uv \in E$, we obtain $N(u)$ and $N(v)$ directly from the adjacency list of the graph, which takes time $O(n)$. Determining the intersection of $N(u)$ and $N(v)$, which can be done in linear time, e.g. by using a hash table. Thus, it takes time $O(n)$ to determine whether a given edge uv can be removed using Rule 3. The removal of edge uv itself takes time $O(1)$. Inspecting all m edges in E for applicability of Rule 3, and applying the rule, takes time $O(m) * O(n) = O(mn)$.

Note that we only need to inspect all $uv \in E$ once to determine and remove all uv with $N(u) \cap N(v) = \emptyset$ from G, because the application of Rule 3 does neither remove any vertex from G nor decreases $|N(u') \cap N(v')|$ for any edge $u'v' \in E$ such that $u'v' \neq uv$. Therefore, each application of Rule 3 takes time $O(mn)$.

Note that nor singleton or degree-one vertex can be a common neighbour of vertices u and v when $uv \in E$. Therefore, after applying Rule 3, and when Rule 3 is not applicable, we require an extra $O(n)$ time to re-apply Rule 1 and Rule 2. Only now we are guaranteed that none of Rules 1,2 and 3 applies to the reduced graph.

Therefore the overall running time to apply Rules 1–3 is $O(n^3)$.

Now let us assume that none of Rules 1–3 applies. We consider Rule 4.

It takes time $O(n)$ to find a vertex v with $|N[v]| > k$ from G by inspecting the degree of each $v \in V$. To determine if the vertices in $N[v]$ induce a clique takes time $O(n^2)$ (by checking whether there is an edge between every pair of vertices in $N[v]$). Since there are n vertices in G, it takes time $O(n) * O(n^2) = O(n^3)$ to determine a vertex that satisfies the preconditions of Rule 4 in I. There are at most n^2 edges in $E(N[v])$. Therefore the removal of the edges in $E(N[v])$ takes time $O(n^2)$.

Figure 4.2 illustrates that Rule 4 might be applicable many times: If we assume that we inspect vertex v_2 before v_4 when applying Rule 4, the first time we inspect vertex v_2, we note that $|N[v_2]| = k$, that is Rule 4 is not applicable at this time. Then we inspect v_4, which meets all the requirements to apply Rule 4. When applying Rule 4, we remove $E(N[v_4])$ and reduce k by one. As it is shown in Figure 4.3, now $E(N[v_4])$ is removed and k is reduced to two. Therefore at this time, Rule 4 can be applied on v_2 since $|N[v_2]| > k$ and vertices in $N[v_2]$ induce a clique.

Thus, we need to inspect all $v \in V$ again whenever Rule 4 is applied. Thus, the overall running time to detect and apply all Rule 4 cases is $O(n^3k)$.

Rule 4 has no influence on the applicability of Rule 3 because no vertex is removed in Rule 4. Rule 1 and 2 have no influence on the applicability of Rule 3 neither, which is proved before. Therefore when Rule 4 is not applicable, we

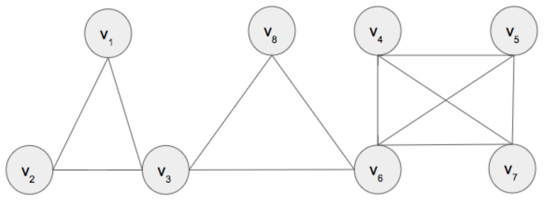


Figure 4.2: Given an instance $I =< G, k >$ with $k = 3$, Rule 4 is applicable on v_4 but not v_2.

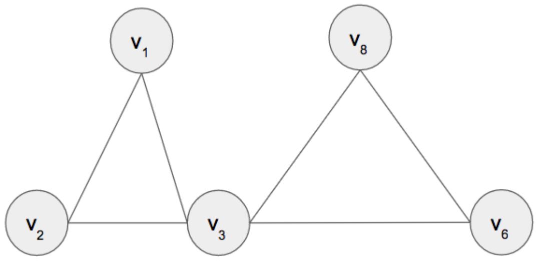


Figure 4.3: After the removal of v_4 and $E(N[v_4])$ in I, k is reduced by one. Rule 4 now can be applied on v_2.

keep applying Rule 1,2,4 until none of them is applicable.

Every time we apply Rule 2 and 4, k is reduced by one. Therefore we can apply them at most k times in total. Among all four reduction rules, the time complexity of Rule 4 is the highest. In the worst case, we apply Rule 4 k times, which takes time $O(kn^3)$.

□

We call an instance $I =< G, k >$ for k-Edge Clique Partition *reduced* if none of Rules 1–Rule 4 is applicable.

In general, we call an algorithm that transforms any parameterized instance I into a parameterized instance I' a *kernelization* if the size of instance I' is bounded by a function that only relies on parameter k. We also call I' a *kernelized* instance.

Lemma 3. *Let $I =< G, k >$ be a reduced instance of k-Edge Clique Partition. Then I is of size at most k^2 [22].*

Proof. Assume there are more than k^2 vertices in instance $I =< G, k >$ where Rule 1 to Rule 4 do not apply. Because Rule 1 is not applicable, there is no singleton in G. That is, every vertex is incident to some edge $e \in E$. Because Rule 4 is not applicable, all the cliques in G are of size at most k. By including k cliques into CP, we can only put at most k^2 vertices in the partition. Furthermore, if there are more than k^2 vertices of degree ≥ 2, there is at least one vertex v that is not in any clique $C \in CP$. Therefore v's incident edge e_0 is not induced by a clique in partition CP. Thus, I is a no-instance.

□

4.1.2 A brute force algorithm and a bounded search tree algorithm for Edge Clique Partition

When none of the above four reduction rules can be applied, using a brute force algorithm to search for the solution of the reduced instance, could looks as follows: Try all combinations for possible edge clique partitions of size k and test their validity. For each such partition, we arrange at most n^2 pairs of vertices (which correspond to edges in the graph) into a set $H = C_1^h, C_2^h, \ldots, C_k^h$. If each subset $C_i^h \in H$ is a clique and for every edge xy $x, y \in C_i^h$ for some i, $1 \leq i \leq k$, then H is an edge clique partition for the reduced instance I.

There are at most $S_{(n^2,k)}$ ways to construct such set H, where $S_{(n^2,k)}$ is the number of ways to put "n^2 balls into k identical bins". To determine whether each C_i is a clique can be done in time $O(n^2)$. To verify whether each H is an edge clique partition, we verify that each $C_i \in H$ is a clique. This takes time $O(kn^2)$. Therefore it takes time $O(S_{(n^2,k)}kn^2)$ to obtain the solution for the reduced instance. Since in the reduced instance $n < k^2$, the total running time is $S_{(k^4,k)}O(k^5)$.

To improve the running time, we provide a bounded search-tree algorithm below to determine our final answer for the obtained kernelized instance of size at most k^2.

Lemma 4. *Algorithm 1 solves the sized k^2 kernelized instance of k-Edge Clique Partition in the running time of $O((2^{k^2-2})^k 2^{k^2-2} k^4)$.*

Proof. Assume $I =< G, k >$ is the kernelized instance of k-Edge Clique Partition. According to Lemma 3 there are at most k^2 vertices in graph G. We apply Algorithm 1 to search for the solution to $I =< G, k >$, where $G =< V, E >$ and $|V|$ is bounded by k^2 since none of Rules 1 4 applies.

We first calculate the maximum height of tree. In Line 13 of Algorithm 1, we create one branch for including each possible clique that partitions edge $uv \in E$ where u,v has minimum number of common neighbours over other edges. The

Algorithm 1 Bounded Search-Tree Algorithm for a kernelized instance of k-Edge Clique Partition

Input: A graph $G = (V, E)$, an initially empty set CP to store cliques, an integer k.
Question: Is there an edge clique partition of size at most k for G?

```
1: function $C_{Partition}(G, CP, k)$
2:     if $E = \emptyset$ then return TRUE
3:     else if $k = 0$ then return FALSE
4:     else
5:         choose $uv \in E$ such that $|N(u) \cap N(v)|$ is minimum
6:         find a set $S$ of all cliques in $N(u) \cap N(v)$
7:         for each $K \in S$ do
8:             $K' = K \cup \{u, v\}$
9:             $CP' = CP \cup \{K'\}$
10:            $k' = k - 1$
11:            $G' := G - E(K')$
12:            if $C_{Partition}(G', CP', k')$ then return TRUE
13:     return FALSE
```

maximum height of the tree T is k, because whenever we branch on any subinstance of I, k will be reduced by one for including a clique in solution CP.

Now we calculate the number of nodes in the tree. We first branch on the given instance $I =< G, k >$, and then branch recursively on I's subinstances until we find a solution or no more branches can be created. For each subinstance, we search G and find the edge uv such that $|N(u) \cap N(v)|$ is minimum. The size of $|N(u) \cap N(v)|$ is at least one since none of Rule 1 to Rule 3 applies. There are at most $k^2 - 2$ common neighbours for u and v. Therefore there are at most 2^{k^2-2} different cliques in $N(u) \cap N(v)$. We branch on this subinstance by including each possible $K \cup \{u, v\}$ in CP (see Lines 8 and 9), where K is a member of S. Therefore, there are at most 2^{k^2-2} branches for each subinstance in T.

Now we calculate the running time of processing each node(subinstance) in the tree. When processing a subinstance of I, we first check all the edges in E to determine $uv \in E$ such that $|N(u) \cap N(v)|$ is minimum (see Line 5). The running time for this operation is $O(k^6)$. Then we generate set S which stores all the cliques in $N(u) \cap N(v)$ (see Line 6). Since there are at most $k^2 - 2$ vertices in $N(u) \cap N(v)$, we obtain 2^{k^2-2} sets of vertices that have the potential to be cliques. To verify whether or not each of those sets is a clique, it takes time $O(k^4)$. Therefore, the time complexity of processing a subinstance to generate S is $2^{k^2-2}k^4$.

The height of T is k and each instance in T has at most 2^{k^2-2} branches. There are at most $(2^{k^2-2})^k$ nodes in the bounded search tree T. The time of processing a node (subinstance) in T is $2^{k^2-2}k^4$. Therefore, the upper bound for Algorithm 1 for a kernelized graph is $O((2^{k^2-2})^k 2^{k^2-2}k^4)$.

□

4.1.3 New Lemma and Reduction Rules for edge clique partition

Except for the reduction rules from Paper [22], we have developed a list of new lemmas and reduction rules which help pre-process the given instance I before using bounded tree algorithms to reach the final solution.

Below are the new lemmas and reduction rules we develop, which are followed by the proofs for their correctness.

Reduction Rule 5. *Let $I =< G, k >$ be an instance of k-*Edge Clique Partition*. Suppose there are 3 vertices $u, v, w \in V$, such that $uv, uw, vw \in E$ and $deg(u) = 2, deg(v) = 2, deg(w) \geq 2$. Then $I =< G, k >$ is yes if and only if $I' =< G - u - v, k - 1 >$ is yes.*

Proof. We want to prove that I is a yes-instance if and only if I' is a yes-instance.

We first prove if I is a yes-instance then I' is a yes-instance. Let CP be an minimum solution of size k for I. In graph G, there is no clique C of size larger than three such that $u, v, w \in C$. Therefore, to cover edges uv, uw and vw, we need either include clique $C_{(u,v,w)}$ or all cliques $C_{(u,w)}, C_{(v,w)}, C(u, v)$ in CP. However, including $C_{(u,w)}, C_{(v,w)}, C(u, v)$ in CP provides a non-minimum edge clique partition. Therefore, $C_{(u,v,w)} \in CP$ and $CP' = CP - \{C_{(u,v,w)}\}$ covers all edges in graph $G' = G - u - v$. Therefore I' is a yes-instance of size at most $k - 1$.

Then we prove if I' is a yes-instance, then I is a yes-instance. Assume CP' is a solution of size at most $k - 1$ for I'. We know clique $C_{(u,v,w)}$ covers all edges incident to u and v, which are uv, uw, vw. Since CP' covers all edges in graph $G - u - v$, then $CP = CP' \cup \{C_{(u,v,w)}\}$ covers all edges in G. Therefore I is a yes-instance.

□

Reduction Rule 6. *Let $< G, k >$ be an instance of k-*Edge Clique Partition*. Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two subgraphs of G, where G_1 is a clique. If $V_1 \cap V_2 = v$, $V_1 \cup V_2 = V$, $E_1 \cap E_2 = \emptyset$ and $E_1 \cup E_2 = E$. $I =< G, k >$ is a yes-instance if and only if $I' =< G_2, k - 1 >$ is a yes-instance.*

Proof. We want to prove that I is a yes-instance if and only if I' is a yes-instance.

We first prove if I is a yes-instance then I' is a yes-instance. Let CP be an minimum solution of size k for I. If we don't include $C_1 = V_1$ in CP, according to Lemma 1, we need at least $|V_1|$ cliques to partition all edges in G_1. And none of those $|V_1|$ cliques contains any vertex from $V_2 - \{v\}$. This contradicts that CP is minimum. Then $CP' = CP - \{C_1\}$ covers all edges in graph $G - G_1$, which is our G_2. Therefore CP' is a solution of size at most $k-1$ for $I' =< G_2, k-1 >$.

Then we prove if I' is a yes-instance then I is a yes-instance. Clique $C_1 = V_1$ covers all edges in G_1 since G_1 is a complete graph. Therefore $CP = CP' \cup \{C_1\}$ is a solution of size at most k for I.

□

Reduction Rule 7. *Let $I =< G, k >$ be an instance of* EDGE CLIQUE PARTITION. *Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two subgraphs of G. Assume $V_1 \cap V_2 = v$, $V_1 \cup V_2 = V$, $E_1 \cap E_2 = \emptyset$ and $E_1 \cup E_2 = E$. Further assume the size of minimum* EDGE CLIQUE PARTITION *for subgraph G_1 is k_1. $I = (G, k)$ is a yes-instance if and only if $I' =< G_2, k - k_1 >$ is a yes-instance.*

Proof. We want to prove that I is a yes-instance if and only if I' is a yes-instance.

We first prove if I is a yes-instance then I' is a yes-instance. Let CP be a minimum edge clique partition of size k for instance I and asuume CP_1 is a minimum EDGE CLIQUE PARTITION for G_1. Since the only intersection of G_1 and G_2 is vertex v and CP_1 covers all edges in G_1, then $CP_2 = CP - CP_1$ covers all edges in $G - G_1$, which is our G_2. Therefore $CP_2 = CP - CP_1$ is an edge clique partition of size $k - k_1$ for G_2.

Then we prove that if I' is a yes-instance then I is a yes-instance. Let CP_2 be an edge clique partition of size at $k - k_1$ for I'. Since CP_2 covers all edges of G_2 and CP_1 covers all edges of G_1, $CP_1 \cup CP_2$ covers all edges in $G_1 \cup G_2$, which is our G. Therefore $CP = CP_1 \cup CP_2$ is an edge clique partition of size k for I.

□

Lemma 5. *Let $I =< G, k >$ be an instance of* EDGE CLIQUE PARTITION. *Suppose that $S = \{G_1, G_2, ..., G_m\}$ a set of subgraphs of G. Assume $V_1 \cup V_2 \cup ... \cup V_m = V, E_1 \cup E_2 \cup ... \cup E_m = E$ and $\{v\}$ is the intersection of any pair of graphs $G_i, G_j \in S$. If $m = k$, I is yes if and only if all $V(G_1), V(G_2), \ldots V(G_m)$ are cliques.*

Proof. If all $V(G_1), V(G_2), \ldots V(G_m)$ are cliques, $CP = \{V(G_1), V(G_2), \ldots V(G_m)\}$ is an edge clique partition of size k for I.

Assume $V(G_1)$ is not a clique. Since pairwise intersections of $G_1, G_2, ..., G_m$ are vertex v, we need at least three cliques to cover edges in $E(G_1)$ and need at least one clique to cover edges from each of $E(G_2), E(G_3), \ldots, E(G_m)$. Then we need at least $k+2$ cliques to covers edges in G, so I is a no-instance. □

Lemma 6. *Let $I =< G, k >$ be an instance of* Edge Clique Partition. *Suppose that $S = \{G_1, G_2, ..., G_m\}$ a set of subgraphs of G. Assume $V_1 \cup V_2 \cup ... \cup V_m = V, E_1 \cup E_2 \cup ... \cup E_m = E$ and $\{v\}$ is the intersection of any pair of graphs $G_i, G_j \in S$. If $m > k$, I is a no-instance.*

Proof. Since the pairwise intersection of $G_1, G_2, ..., G_m$ are vertex v, we need at least one clique to partition edges from each of $G_1, G_2, \ldots, G_m$. Therefore we need at least $m > k$ cliques to partition edges in G and I is a no-instance. □

Note that if $m < k$, we can apply Rule 6 to shrink the graph G in instance I.

4.2 Algorithms for k-Edge Clique Cover

In a solution for k-Edge Clique Cover, when given an instance $I =< G, k >$, for every edge $uv \in E$, u and v appear together in some clique $C_i \in CC$, where $|CC| \leq k$. In general, we call an edge uv *covered* by a clique C in G if $u, v \in C$.

Compared to k-Edge Clique Partition, a solution for an instance to k-Edge Clique Cover allows an edge $uv \in E$ to be covered by multiple cliques in the solution. Note that for a given graph $G = (V, E)$, a clique partition of size k for G is also a clique cover for G. However, a clique cover of size k for G is not guaranteed to be a clique partition for G (see Figure 4.4).

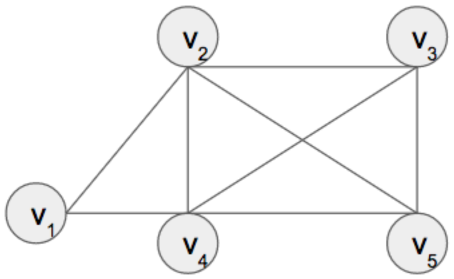


Figure 4.4: A graph $G = (V, E)$ with a minimum clique partition of size 3 and a minimum clique cover of size 2: A minimum clique partition of size 3 for G is $CP = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_3\, v_4, v_5\}\}$. CP is also a clique cover of size 3 for G since for each edge of G both endpoints are members of at least one clique in CP. However, a (minimum) clique cover for G of size 2 exists, i.e. $CC = \{\{v_1, v_2, v_4\}, \{v_2, v_3, v_4, v_5\}\}$. CC is not a clique partition for G since vertices v_2 and v_4 appear both in both cliques of CC.

To be able to allow edges to be covered by several cliques k-Edge Clique Cover, we allow the possibility to mark an edge e as *covered*. To be able to solve k-Edge Clique Cover in the case where the graph contains edges covered edges in a current instance that are marked as covered, we add an auxiliary set A to our instance I and consider solving k-Annoted Edge Clique Cover, the annotated version of k-Edge Clique Cover (defined above). Note that k-Edge Clique Cover is a special case of k-Annoted Edge Clique Cover where $A = \emptyset$.

We now describe an FPT algorithm for k-Annoted Edge Clique Cover that was introduced in [15]. The preprocessing phase of this algorithm consists of four reduction rules that reduce a given instance $I =< G, A, k >$ of k-Annotated Edge Clique Cover to a kernelized instance I' with a graph of size at most k^2. The preprocessing is followed by a bounded search-tree algorithm that solves I'. We prove that this algorithm solves k-Annotated Edge Clique Cover in time $O(2^{k^{k+2}})$.

Before introducing reduction rules to be applied on instance $I =< G, A, k >$, we run an initialization phase that computes some useful parameters of graph G.

Initialization. For every edge $uv \in E$, we compute a set $N_{\{u,v\}}$ that stores all common neighbours of vertices u and v. We also record the size of $N_{\{u,v\}}$ for every edge $uv \in E$. Furthermore, compute $c_{\{u,v\}} = |E(N_{\{u,v\}})|$.

After the initialization phase, we apply the following reduction rules to the instance.

Reduction Rule 8. *Let $I =< G, A, k >$ be an instance of k-Annotated Edge Clique Cover, and let $v \in V$ be a vertex such that $deg(v) = 0$. Then remove v from G.*

Proof. Assume $I' =< G - v, A, k >$ is an instance obtained by applying Rule 8 on I. We prove that I is a yes-instance if and only if I' is a yes-instance.

First, we show if I is a yes-instance then I' is a yes-instance. Given $I =< G, A, k >$, consider vertex v in G with $deg(v) = 0$. Assume CC is a solution for for I with $|CC| = k$. Since v has no incident edges, there is no $C_i \in CC$ with $|C_i| > 1$ such that $v \in C_i$. Therefore, $CC - C_j||C_j| = 1$ is a solution of size at most k for I'.

We now show that if I' is a yes-instance then I is a yes-instance. Given instance $I' =< G - v, A, k >$, where CC' is a solution of size k for I'. Adding any new vertex v with $deg(v) = 0$ to G' does not create any additional edges. Therefore $CC = CC'$ is a solution of size k for I.

□

From now on we assume that Rule 8 is not applicable to instance I.

Reduction Rule 9. *If $I =< G, A, k >$ is an instance of k-*Annotated Edge Clique Cover *with $A \neq \emptyset$, and where $v \in V$ is a vertex in G with all incident edges of v are elements in A, then I' is obtained by creating G' from G via removing v and updating the set of covered edges to $A' = A - \{e \in E | e \text{ is incident to } vinG\}$.*

Proof. Let $I' =< G - v, A', k >$ with $A' = A - \{e \in E | e \text{ is incident to } vinG\}$ is an instance obtained by applying Rule 9 on I for a vertex $v \in V$ in G with all incident edges of v are elements in A.

We prove that I is a yes-instance if and only if I' is a yes-instance.

First, we show if I is a yes-instance then I' is a yes-instance. Let CC be a solution of size k for $I =< G, A, k >$, and let v be a vertex in G such that all of v's incident edges are members of the set of covered edges A. We observe that for every clique $C_i \in CC$ such that $v \in C_i$, $C_i - \{v\}$ is also a clique in G. Furthermore, every edge $u'v' \in E$ with $u', v' \neq v$ is covered by a at least one clique in $CC' = \{C_i - \{v\} | C_i \in CC\}$. Therefore CC' is a solution of size at most k for I'.

We next prove that if I' is a yes-instance then I is a yes-instance. Let $I' =< G - v, A', k >$ with $A' = A - \{e \in E | e \text{ is incident to } vinG\}$ be a yes-instance. Let CC' be a solution of size k for I'. Since $E' = E - \{e \in E | e \text{ is incident to } vinG\}$ and $A' = A - \{e \in E | e \text{ is incident to } v\}$, $E - A = E' - A'$. Therefore CC' is also a solution for I.

□

From now on we assume that neither Rule 8 nor Rule 9 are applicable to instance I. Before introducing and proving correctness of rules 10 and 11, we consider the following lemma.

Lemma 7. *Let $I =< G, A, k >$ be an instance of k-*Annotated Edge Clique Cover*, and let CC be a solution of size k for I. Then there is a solution CC' for I, $|CC'| \leq |CC|$, where every member of CC' is a maximal clique.*

Proof. Assume CC is a solution for I. Consider a clique $C_i \in CC$ that is not a maximal clique—if such a member exists (if not, we are done since then $CC' = CC$ is a solution).

We build a maximal clique C_j incrementally by including v in C_i such that $v \in N(C_i)$ and $vv' \in E$ for every $v' \in C_i$. When there is no such v, C_j is maximal since it cannot be further enlarged. At each step we only add a vertex that is adjacent to every vertex of C_i, so C_j is indeed a clique.

Since $C_i \subset C_j$, C_j covers all edges in $E(C_i)$. Therefore replacing every such C_i by a maximal clique C_j leads to a new solution CC' for instance I. □

Reduction Rule 10. *Let $I =< G, A, k >$ be an instance of k-*Annotated Edge Clique Cover. *If there exists an edge uv in G such that $N_{\{u,v\}}$ induces a clique, then include all edges induced by $N_{\{u,v\}} \cup \{u, v\}$ into A and reduce k by one.*

Proof. Let $I =< G, A, k >$ be an instance of k-Annotated Edge Clique Cover with $uv \in E$ and $N_{\{u,v\}}$ induce a clique. Furthermore, let $I' =< G, A', k - 1 >$ with $A' = A \cup E(N_{\{u,v\}} \cup \{u, v\})$ be the instance obtained by applying Rule 10 on I for $uv \in E$. We prove that I is a yes-instance if and only if I' is a yes-instance.

First, we show that if I is a yes-instance then I' is a yes-instance. Given instance $I =< G, A, k >$, assume CC is a solution of size k for I with all its members are maximal cliques (Lemma 7).

Note that if $N_{\{u,v\}}$ induces a clique and $uv \in E$, then $N_{\{u,v\}} \cup \{u, v\}$ is a clique as well.

Furthermore note that $C_i = N_{\{u,v\}} \cup \{u, v\}$ is the unique maximal clique that covers edge uv, since there is no other vertex in G that is adjacent to both u and v.

Since every clique in CC is maximal and C_i is the unique maximal clique that covers edge uv, $C_i \in CC$. Since CC covers $E - A$, $CC - \{C_i\}$ covers $E - A - E(N_{\{u,v\}} \cup \{u, v\}) = E - A'$. Therefore $CC' = CC - \{C_i\}$ is a solution of size at most $k - 1$ for I'.

We an now prove that if I' is a yes-instance then I is a yes-instance. Let $I' =< G, A', k - 1 >$ be an instance of k-Annotated Edge Clique Cover, and let CC' be a solution of size $k-1$ for I'. Since $A' = A \cup E(N_{\{u,v\}} \cup \{u, v\})$ and clique $C_i = N_{\{u,v\}} \cup \{u, v\}$ covers all edges in $E(N_{\{u,v\}} \cup \{u, v\})$, $CC = CC' \cup \{C_i\}$ is a solution of size at most k for I.

□

From now on we assume that none of the Rules 8–10 applies to instance I.

Reduction Rule 11. *Let $I =< G, A, k >$ be an instance of k-*Annotated Edge Clique Cover*, and let $uv \in E$ with $N[u] = N[v]$. Then include all of u's incident edges into A.*

Proof. Let $I =< G, A, k >$ be an instance of k-Annotated Edge Clique Cover, and let $uv \in E$ with $N[u] = N[v]$. Let $I' =< G, A', k >$ where

$A' = A \cup \{$ all edges incident to $u\}$ be an instance obtained by applying Rule 11 to I on edge uv. We prove that I is a yes-instance if and only if I' is a yes-instance.

We first prove that if I is a yes-instance then I' is a yes-instance. Given an instance $I =< G, A, k >$, let CC be a solution of size k for I. Since $E - A'$ is a subset of $E - A$, if CC covers all edges in $E - A$ then CC covers all edges in $E - A'$. Therefore $CC' = CC$ is a solution of size k for I'.

Now we prove that if I' is a yes-instance then I is a yes-instance. Given an instance $I' =< G, A', k >$, where $A' = A \cup \{$ all edges incident to $u\}$, according to Lemma 7, there exists a solution CC' of size k for I' that is composed of maximal cliques only.

We show that for any member $C_0 \in CC'$ where every clique in CC' is maximal, if $v \in C_0$, then $u \in C_0$. We prove this statement by contradiction. Assume that there is a clique $C_0 \in CC'$ such that $v \in C_0$ but $u \notin C_0$. Because $v \in C_0$, every vertex in clique $C_0 - v$ is adjacent to v. Since $N[u] = N[v]$, every vertex in C_0 is adjacent to u as well. Therefore adding u to C_0 would result in a larger clique. This shows that C_0 is not maximal, a contradiction to Lemma 7.

Let $C_1, C_2, \ldots, C_h \in CC'$ such that $h < k$, $v \in C_i$ for all $1 \leq i \leq h$. From above we know that $u \in C_1$, $u \in C_2$,..., $u \in C_h$. Since $C_1, C_2, \ldots, C_h$ cover all edges incident to v and $N[u] = N[v]$, they cover all edges incident to u as well. Therefore, $CC = CC'$ is a solution to I of size at most k.

□

We call an instance of k-Annotated Edge Clique Cover *reduced* if none of the Rules 8–11 can be applied.

Lemma 8. *Given instance I of k-*Annotated Edge Clique Cover*, in time $O(n^4)$ a reduced instance can be obtained.*

Proof. Before applying the reduction rules introduced above, we pre-process the given instance in the initialization phase and compute, for every edge in E, $N_{\{u,v\}}$ and $c_{\{u,v\}}$. We now show that this initialization can be done in time $O(n^4)$.

For each edge $uv \in E$, we use an adjacency list and a hash table (page 15) to calculate the intersection of $N[u]$ and $N[v]$. This can be done in time $O(n)$. To obtain $c_{\{u,v\}}$, we inspect every edge $e \in E$ and sum up the number of edges with both endpoints in $N_{\{u,v\}}$. This can be done in time $O(n^2)$. Therefore it takes time $O(n^2)$ to compute $N_{\{u,v\}}$ and $c_{\{u,v\}}$ for one edge $uv \in E$. Since there are at most n^2 edges in E, this step takes time $O(n^4)$ in total for the initialization phase.

Next we consider applying Rule 8 on I. To determine and remove all singletons in G, in time $O(n)$ we can inspect all $v \in V$, followed by time $O(1)$ for each singleton found to apply the removal. Note that the removal of singletons does not affect the degree of other vertices. Therefore every vertex needs to be inspected only once. Thus, given graph G using Rule 8 to generate an instance such that Rule 8 is no longer applicable takes time $O(n)$.

Now we consider the application of Rule 9. Since Rule 8 is not applicable to I, we only determine and remove a vertex v, where v is incident to edges only in A. To achieve this goal, we use a hash table to store edges in A. For each vertex $v \in V$, we obtain all its incident edges and inspect for each of them whether or not it is in A. Each inspection takes time $O(1)$, and each v is incident to at most $n - 1$ edges. Therefore it takes time $O(n)$ to decide whether or not v is only incident to edges in A.

When applying Rule 9, we only remove vertex v and its incident edges. Assume a vertex $v' \in V$, where $v' \neq v$, is incident to an edge $e \notin A$. After the removal of v, v' is still incident to edge e. Therefore we only inspect every vertex once and perform the removal. This procedure takes time $O(n^2)$. Afterwards, Rule 9 is no more applicable to G.

When applying Rule 9, since some vertex $v \in V$ is removed, we need to update $N_{\{u,v\}}$ and $c_{\{u,v\}}$ for every v's neighbour u'. This can be one in time $O(n)$ since v has at most $n - 1$ neighbours.

Note that the removal of v where v was only incident to edges in A, might again create singletons. It takes an additional $O(n)$ time to execute Rule 8 once more to G after applying Rule 9.

Now assume neither Rule 8 nor Rule 9 applies.

To determine whether Rule 10 is applicable on instance I, we need to inspect every edge $uv \in E$ to determine whether or not set $N_{\{u,v\}}$ induces a clique.

For every edge uv, the size of $N_{\{u,v\}}$ and the value of $c_{\{u,v\}}$ are already computed during the initialization phase (and updated when necessary, as discussed above). Let $n' = |N_{\{u,v\}}|$. If $c_{\{u,v\}} = \frac{n'(n'-1)}{2}$, then $N_{\{u,v\}}$ is a clique. Therefore it takes time $O(1)$ to determine whether an edge satisfies the conditions of applying Rule 10.

Note that the application of Rule 10 does not change G (however, it does change A).

We inspect every edge in the graph once for applicability of Rule 10. Since there are at most n^2 edges in the graph, in time $O(n^2)$ we can apply Rule 10 until the rule is no more applicable.

When Rule 10 is not applicable, we in an additional time of $O(n^2)$ we can re-apply Rule 9 and then Rule 8, since the application of Rule 10 increases the number of edges in A.

Now assume that Rule 11 is the only applicable rule on I. Rule 11 considers whether or not $N[u] = N[v]$, for each $uv \in E$. This can be done in time $O(n)$ by using a hash table to inspect edge uv. The application of Rule 11 does not change G. Therefore each edge needs to be inspected only once. There are at most n^2 edges in the graph. Therefore in time $O(n^3)$ we can determine all edges for which Rule 11 is applicable.

When Rule 11 is not applicable, in an additional time of $O(n^2)$ we can first apply Rule 9 and then Rule 8. We do this since Rule 11 increases the number of edges in A and therefore might invoke the applicability of Rule 9.

Based on above analysis, the $O(n^4)$-time initialization phase is the most expensive operation. After the initialization, in $O(n^3)$ we apply Rules 8–11, which leads to a reduced instance. Thus, the overall running time for the preprocessing of given instance is $O(n^4)$.

□

When none of the Rules 8–11 is further applicable, G is reduced to an instance with at most 2^k vertices. To prove this kernelization result, we first consider the following lemma.

Lemma 9. *Assume u and v are two vertices in a given instance of k-*Annotated Edge Clique Cover*, denoted as $I =< G, k, A >$. Let CC be a solution for I. If for every $C_i \in CC$, C_i either contains both u and v, or neither u nor v, then $N[u] = N[v]$.*

Proof. We prove this lemma by contradiction. Assume we are given an instance of k-Annotated Edge Clique Cover, denoted as $I =< G, k, A >$ with solution CC of size k. Assume that in CC for every $C_i \in CC$, C_i either contains both u and v, or neither u nor v. For the sake of contradiction assume that $N[u] \neq N[v]$.

Since $N[u] \neq N[v]$, there exists a vertex v' such that $uv' \in E$ but $vv' \notin E$. Let clique C_i be the clique in CC that covers edge uv'. Then, since $u \in C_i$, according to our assumption also $v \in C_i$. However, $vv' \notin E$, a contradiction to the definition of clique. □

Next we prove that our instance is kernelized when reduced.

Lemma 10. *Assume that none of the Rules 8–11 is applicable to a reduced instance $I =< G, k, A >$ of k-*ANNOTATED EDGE CLIQUE COVER. *Then there are at most 2^k vertices in G, otherwise I is a no-instance.*

Proof. We prove Lemma 10 by contradiction. Assume $I =< G, k, A >$ is a reduced instance with more than 2^k vertices. Since there is no singleton in G (Rule 8) and there are no vertices that are incident only to edges in A (Rule 9), every vertex v in G has to be a member of some $C_i \in CC$, since otherwise v's incident edges are not covered.

Since Rule 11 does not apply, there is no pair of vertices u, v in G such that $N[u] = N[v]$. Now we show that if there are more than 2^k vertices, there is a pair of vertices u, v with $N[u] = N[v]$:

Recall that we assume that there are more than 2^k vertices in V. Let v_{2^k} be the $(2^k + 1)^{th}$ vertex in V. Since there are at most k cliques in CC and there are at most 2^k different ways to arrange a particular vertex into the cliques of CC, it is impossible for v_{2^k} to be part of cliques in CC while being different from all the configurations of vertices $v_0, \ldots, v_{2^k-1}$ in the arrangement.
According to Lemma 9, there is a $v_i \in V$ with $N[v_i] = N[v_{2^k}]$, which contradicts our assumption that $N[u] \neq N[v]$ (since Rule11 is not applicable).

Therefore, a clique cover for I can contain at most 2^k vertices in G if I is a reduced instance. □

Next we use a bounded search tree algorithm on the reduced instance I [15]:

Algorithm 2 Bounded Search Tree Algorithm for k-ANNOTATED CLIQUE COVER

Input: A reduced graph $G = (V, E)$, a set A of edges, an integer k.
Output: A clique cover of size at most k.

```
1:  $A \leftarrow \emptyset, CC \leftarrow \emptyset$.
2:  return $C_{cover}(G, k, A, CC)$;
3:
4:  function $C_{cover}(G, k, A, CC)$
5:      if $CC$ covers $G - A$ then: return $CC$
6:      if $k < 0$ then: return nil
7:      choose $e_{ij}$ such that $\binom{|N_{\{i,j\}}|}{2} - c_{\{i,j\}}$ is minimum
8:      for each maximal clique $C$ in $N[i] \cap N[j]$ do
9:          $CC' \leftarrow CC \cup \{C\}$
10:         $X' \leftarrow C_{cover}(G, k - 1, A, CC')$
11:         if $CC' \neq nil$ then: return $CC'$
12:     return nil
```

Now we explain above algorithm in details and discuss its running time for a reduced instance with $|V| \leq 2^k$ vertices.

As it is shown in Lemma 7, among all the minimum solutions for I, there is a solution CC where every member is a maximal clique. The above algorithm finds such a solution CC as below:

For an edge $e \in E - A$, we branch by including each possible maximal clique that contains e and then solve the obtained instance recursively (Line 10).

In each subinstance, k is reduced by one. When $k = 0$, I is a no-instance if $E' - A' \neq \emptyset$. Otherwise, it is a yes-instance. Therefore the height of the tree is a most k.

The the height of tree is bounded by k, so the size of the tree can be controlled by reducing its width.

At each node, after choosing $e \in E$ (Line 7), we branch by including each possible maximal clique that contains e.

Note that in Line 7, for any edge $v_i v_j \in E$, if $h = \binom{|N_{\{i,j\}}|}{2} - c_{\{i,j\}}$ is minimum, $|E_m|$ is minimum where E_m is the set of missing edges that prevent $N_{\{i,j\}}$ from being a clique. Then $v_i v_j$ is contained in fewest maximal cliques: the removal of one endpoint from each $e \in E_m$ leads a maximal clique that contains both vertices v_i and v_j.

Now we calculate the size of the search tree, that is its maximum number of nodes. In the worst case, the endpoints v_i, v_j of edge $e \in E$ have $n - 2$ common neighbours and no pair of their neighbours are adjacent to each other. Then every edge e is contained in $|V|$ maximal cliques. Therefore each node has at most $|V|$ branches. Since the height of the tree is at most k, there are at most $|V|^k$ nodes in the tree. Recall that $|V| \leq 2^k$.

Now we calculate the time to process any node in the tree, and the total running time to solve the reduced instance. When processing each instance (node) in the tree, we need to determine an edge with a minimum number of maximal cliques by calculating $\binom{|N_{\{i,j\}}|}{2} - c_{\{}i,j\}$ for each $e_ij \in E$. Since $N_{\{}i,j\}$ and $c_{\{}i,j\}$ are pre-calculated in initialization phase, it takes time $O(|V|^2)$ to process each node. There are $O(|V|^k)$ nodes in the tree. Therefore this bounded search algorithm takes time $O(|V|^{k+2})$. Since $|V|$ is bounded by 2^k, the overall running time for the bounded search tree algorithm is $O(2^{k^{k+2}})$.

Recall that preprocessing and kernelization are done in $O(n^4)$. Therefore $O(n^4 + 2^{k^{k+2}})$ is the total running time to solve k-ANNOTATED EDGE CLIQUE COVER with bounded search tree Algorithm 2.

Chapter 5

Applications of Clique-based Problems

Considering the variants of cluster problems we mentioned in this project, Edge Clique Cover and Edge Clique Partition might be the ones that are most popular in application settings. While reviewing the literature, we found a number of papers discussing applications of these two problems in different fields. In the following sections of this chapter, we discuss three representative papers, which are in the fields of Hardware Design, Computational Geometry and Bioinformatics.

5.1 Applications of Clique Cover

As it is defined in Chapter 3, pairs of vertices in a solution of an instance for Edge Clique Cover are allowed to appear together in more than one clique. It means that if we model an application with this problem, we do not constrain the number of individual objects shared by any pair of clusters when constructing a solution. This property leads to a wide range of applications in the fields of compiler optimization, computational geometry and applied statistics. In this section, we present two application papers in details: Handling Irregular ILP Within Conventional VLIW Schedulers Using Artificial Resource Constraints by Subramanian [26] and Can Visibility Graphs be Represented Compactly by Panakaj [1].

5.1.1 Handling Irregular ILP Within Conventional VLIW Schedulers Using Artificial Resource Constraints [26]

We discuss an application of Edge Clique Cover when implementing VLIW (Very Long Instruction Word) Processors. The goal of VLIW problems is to find an assignment of resources to operation types while minimizing the number of resources and supporting all operation types (where some of the operation types might be required to be run in parallel) in a given instruction set architecture (ISA). The authors of [26] discuss models for and solutions of VLIW problems by first using Edge Clique Cover (see Chapter 4), and then converting the obtained solution of Edge Clique Cover to a solution of the given VLIW instance.

The size of VLIW problems is the number of operation types in the given ISA, which is in practice typically very small.

To understand this application, we first introduce the following definitions.

Definition 1. *(ISA) An* instruction set architecture (ISA) *or* computer architecture *is a low level abstract model of a computer. The description of the model uses native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O.*

We can also view an ISA as a list of instructions that are all supported by a computer, and we need to consider how to program the instructions and assign resources to realize those instructions.

Famous examples of ISAs include: Hiperion DSP (digital signal processor), DEC VAX-11/780 and Intel 80x60.

Definition 2. *(*Operation Types*) are types of operations supported in a particular ISA.*

For example, a particular ISA may support the following four operations: loading, addition, subtraction and multiplication. The corresponding set of *operations types* is $O = \{LOAD, ADD, SUB, MUL\}$.

Definition 3. *(*Operation Set*) We define $Q \subset I \times O$ to be the set of operations, where I is the set of identifiers.*

Note that a certain operation type might appear many times to execute a given set of instruction. All operations in an ISA (even those of the same type) are considered different from each other. To distinguish different operations of the same type, in the definition above, there is an identifier in each $q = (i, t) \in Q$.

We consider the following example. Let $I = \{1, 2, 3, 4, 5\}$ and $O = \{LOAD, ADD, SUB, MUL\}$. Then $Q = \{(1, LOAD), (2, ADD), (3, SUB), (4, LOAD)\}$ is a valid operation set.

Definition 4. *(*Resource Set*) Finite resource set R is used to denote the set of resources that operations may potentially occupy.*

For example, $R = \{ADDER1, LOADER1, ADDER3, ISSUE2\}$, where $ADDER1, LOADER1, \ldots$ are resources of the computer. Below are examples of the functionality of some types of resources.

ADDER: performs addition of binary numbers.
LOADER: assigns values to parameters.
ISSUE: obtains values from parameters.
MULTIPLIER: multiplies two binary numbers.

To execute an operation type in a given ISA, we might need to use several resources in a certain order, where each usage of resource is defined as a cycle.

Definition 5. *(*Resources Usage*) A resource usage for executing an operation of type $o \in O$ is an element u of set $U = N \times R$, where N is the set of cycles, and R is the set of resources.*

For example, for $n \in N$ and $r \in R$ $u = (n, r)$ means that in the n^{th} cycle of our operation, resource $r \in R$ is occupied. Note that a resource can only be occupied at most once in each cycle.

To perform a particular operation type, several cycles might be required, where each cycle occupies exactly one resource.
In such a case we can use a *resource usage table* T, $T \in 2^U$ where 2^U is the power set of U. to show which resource is used in which cycle when executing an operation type. In other words a resource usage table shows which resources are occupied for which cycle of operations.

For example, assume in a given ISA, we require $ISSUE1$ and $ADDER1$ to perform operation ADD. Then $T = \{(0, ISSUE1), (1, ADDER1)\}$ means that ISSUE1 is occupied in Cycle 0 and ADDER1 is occupied in Cycle 1. Figure 5.1 is the corresponding resource usage table (assuming that $ISSUE1$, $ADDER1$ and $MULTIPLIER1$ are the only resources of the machine).

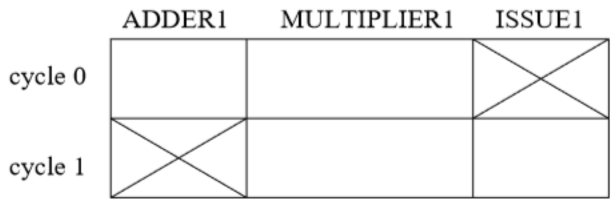

ADDER1
MULTIPLIER1
ISSUE1
cycle 0
cycle 1

Figure 5.1: A usage table for operation ADD.

In a given ISA, there might be several alternative usage tables for an operation type. For example, if there are resources $ADDER1$, $ADDER2$, $ISSUE1$ and $MULTIPLIER1$ in the machine, in Cycle 1 of our ADD operation, we can choose to use either $ADDER1$ or $ADDER2$. Then we have two alternative usage tables as Figure 5.1.1.

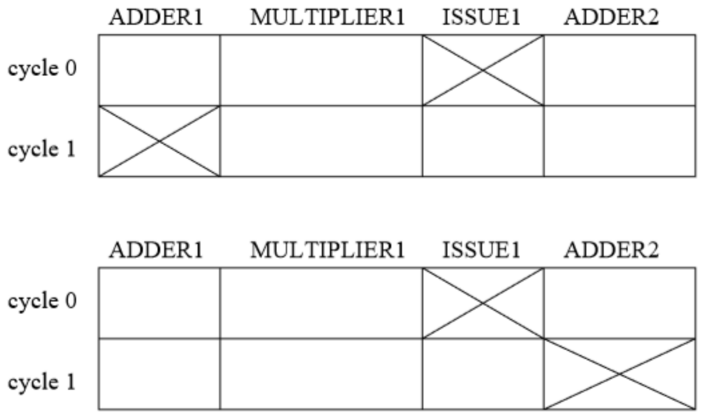


Figure 5.2: Two alternative resource usage tables for operation ADD.

Note that two operation types $o_i, o_j \in O$ can be executed in parallel if and only if they do not need to use the same resource in the same cycle.

Note that if we assign $r_1 \in R$ to $o_1 \in O$ and $r_1, r_2 \in R$ to $o_2 \in O$, then o_1 and o_2 can run in parallel since we can let o_2 use r_2 and o_1 use r_1 whenever these two operation types run at the same time. But if we would instead assign r_1 to o_2, then r_1 and r_2 cannot be executed in parallel.

In an ISA, if we do not require any pair of operation types to run in parallel, we need only one set of resources since we can run all operations one by one.

If we want to enable every pair of operation types to be executable in parallel, we need to assign a set of resources to each operation type, which is generally not practical, especially when resources are limited.

Therefore, computer engineers look for a trade-off between resource occupation and the possibility of parallel execution of operation types. When designing the ISA's, they specify which pairs of operation types should be executed in parallel and create a list accordingly. The list is notated as L in this section.

Based on this specification, our task is to assign resources to operation types to enable the parallel execution of every pair of operation types in L. The problem to be solved is called VLIW and defined below.

For the sake of reserving some resources for other operations and saving energy, we do not want to assign more resources of the machine to the ISA than necessary. Thus, the goal of VLIW is to minimize the number of resources to support all parallel operation types in L.

We can formally describe VLIW as follows. Given are a set O of operation types and a list L of pairs of operations types from O that are required to be executed in parallel. The goal is to determine a smallest possible set R of resources such that there exists an assignment A of resources from R to operation types in O while A satisfies these two constraints:

1. $A(o_i) \cap A(o_j) \neq \emptyset$ if $(o_i, o_j) \notin L$.
2. $A(o_i) \cap A(o_j) = \emptyset$ if $(o_i, o_j) \in L$.

Here $A(o)$ denotes the set of resources assigned to operation type o. For example, if resources r_1, r_3 are assigned to operation type o_i, then $A(o_i) = \{r_1, r_3\}$.

VLIW

Input:	A graph $G = (O, L)$ with operation set O as vertices of G and pairs in list L as edges of G.
Parameter:	*None*
Question:	What is a resource set R and an assignment A for R onto O such that the size of R is minimized?

We describe next how to solve an instance of VLIW via Edge Clique Cover.

ECC Algorithm for VLIW:

1. Compute the complement graph $G' = (O, E)$ for G where $E = (O \times O) - L$.
2. Compute a minimum edge clique cover CC for graph G'.
3. For each clique $C_i \in CC$, assign a unique resource r_i to every $o \in C_i$.

For example, given a set $O = \{o_1, o_2, o_3, o_4\}$ and a list $L = \{(o_1, o_4), (o_2, o_4)\}$ and we want to compute a assignment A of minimized R on O. First, we build a graph $G = (O, L)$ as stated above. Then we compute the complement graph $G' = (O, E)$ for G, where $E = \{o_1o_2, o_1o_3, o_2o_3, o_3o_4\}$. The minimum edge clique cover for G' is $CC = \{\{o_1, o_2, o_3\}, \{o_3, o_4\}\}$, the size of which is two. Therefore, the outputs are $R = \{r_1, r_2\}$ and $A(o_1) = r_1$, $A(o_2) = r_1$, $A(o_3) = r_1, r_2$ and $A(o_4) = r_2$. In our solution, operation types in pairs (o_1, o_4) share no resources, so they can be executed in parallel, so as pairs (o_2, o_4).

Before proving the soundness of above algorithm to search for the minimum assignment, we first show the definition of Edge Clique Cover problems.

Edge Clique Cover

Input:	A graph $G = (V, E)$
Parameter:	*None*
Question:	What is the minimum *edge clique cover* CC for graph G such that for every edge $uv \in E$, vertices u and v appear together in at least one clique $C \in CC$?

Since we compute the edge clique cover for $G' = (O, E)$ in our algorithm, each edge in E represents a pair of operation types which is not required to execute in parallel. In fact, we are searching for clusters where each cluster stores operation types do not execute in parallel. Therefore, each cluster only needs to be assigned one resource from R.

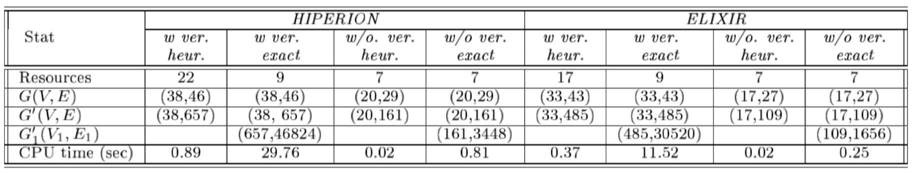
Now we prove the correctness of our algorithm by proving that the assignment generated satisfies the two constraints of VLIW we discussed above.

Proof. We first prove the first constraint. An edge $o_u o_v \in E$ exists if operation types o_u and o_v do not have to be executed in parallel. According to the definition of edge clique cover, edge $o_u o_v$ is covered by a clique $C_i \in CC$ and both of o_u and o_v are assigned with resource r_i. As a consequence, $r_i \in (A(o_u) \cap A(o_v))$, which means $A(o_u) \cap A(o_v)$ is not empty if o_u and o_v are not able to execute in parallel.

The we prove the second constraint. If operation types o_u and o_v can be executed in parallel, then $o_u o_v \notin E$ in our graph construction. Then u and v do not appear together in any $C_i \in CC$, otherwise definition of edge clique cover is violated. Therefore, there is no $r \in R$ assigned to both o_u and o_v, which means that $A(o_u) \cap A(o_v) = \emptyset$ if o_u and o_v can be executed in parallel.

Thus, if we use the algorithm described above to assign resources to operation types, both constraints are satisfies. Assume A is the assignment generated from CC. We cannot find a smaller assignment than A, which can be proved by contradiction and we do not discuss the details here. □

Though edge clique cover problem provide some sound strategies to solve the VLIW problem, as a well-known NP-Complete problem, it has no algorithms in polynomial running time. The paper [26] discusses a heuristic algorithms to solve edge clique cover problem, but it has no guarantee about the quality of solution. As it is shown in Figure 5.3, in some data sets, the results from heuristic algorithm are of size twice larger than the smallest solutions from exact algorithm.



Stat	*HIPERION*				*ELIXIR*			
	w ver. heur.	*w ver. exact*	*w/o. ver. heur.*	*w/o ver. exact*	*w ver. heur.*	*w ver. exact*	*w/o. ver. heur.*	*w/o ver. exact*
Resources	22	9	7	7	17	9	7	7
$G(V,E)$	(38,46)	(38,46)	(20,29)	(20,29)	(33,43)	(33,43)	(17,27)	(17,27)
$G'(V,E)$	(38,657)	(38, 657)	(20,161)	(20,161)	(33,485)	(33,485)	(17,109)	(17,109)
$G'_1(V_1,E_1)$		(657,46824)		(161,3448)		(485,30520)		(109,1656)
CPU time (sec)	0.89	29.76	0.02	0.81	0.37	11.52	0.02	0.25

Figure 5.3: Implementation Results of VLIW.

However, sometimes engineers do not ask for the smallest assignment of resources to support parallel executions in L. Instead, they reserve a certain amount of resources and want to know whether the resources are sufficient for the ISA. In this case, we can model the problem with decision version of Edge Clique Cover problem as below, and solve the k-Edge Clique Cover with FPT algorithm (see Section 4.2).

1. Compute the complement graph $G' = (O, E)$ for G, where $E = O \times O - L$.
2. Let k be the number of available resources.
3. Search solution of k-Edge Clique Cover for instance $I =< G', k >$. k resources are sufficient to support parallel executions of operation types in L if and only if I is a yes-instance.

5.1.2 Algorithms for compact letter displays: Comparison and evaluation [14]

In fields like Biology, Social Science and Statistic, scientists design experiments and collect data to examine their hypothesis. These experiments are conducted in different environmental conditions. Each environmental condition is also called a treatment. By analyzing the results of experiments running in various treatments, scientists determine which factors have impact on the outputs of the experiment.

While some treatments are significantly different from each other, others are not. We say that two treatments are significantly different from each other if the difference of experimental results in treatments is not attributed chance. To store the significant difference between treatments and to provide convenience for further data analysis, there is a need to find a data structure that stores the information that which pairs of treatment have significant difference, within a small amount of space.

In this section, we introduce a binary matrix as the data structure to store the significant difference between treatments, as well as discuss how to apply Edge

Clique Cover to generate a binary matrix for a given set H of significantly different treatment pairs. We also introduce a heuristic algorithm to find binary matrix for H and compare it with the bounded search tree algorithm for that determines binary matrix for H via Edge Clique Cover and discuss the evaluation of their running time on real-world datasets from [14].

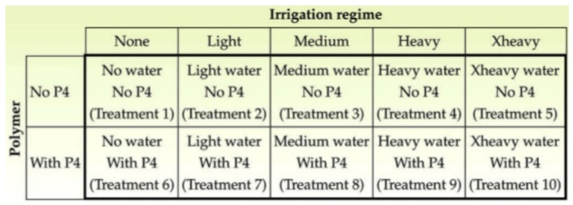
We first introduce some terminology.

Definition 6. *(Factor) In an experiment, a* factor *is an independent variable manipulated by the experimenter. [28]*

Definition 7. *(Levels)* Levels *are the possible values of a factor. In an experiment, each factor contains at least two* levels *[28].*

Definition 8. *(Treatments) A* treatment T *is an element in the Cartesian product of set of factors* F *and set of levels* L*, denoted as* $T \in F \times L$*. [28]*

We call experiments that contain multiple factors multifactor experiments. Since scientists want to determine how different levels of factors affect the experimental results, many experiments conducted are multifactor. Figure 5.4 is an example of a multifactor experiment.



		Irrigation regime				
		None	Light	Medium	Heavy	Xheavy
Polymer	No P4	No water No P4 (Treatment 1)	Light water No P4 (Treatment 2)	Medium water No P4 (Treatment 3)	Heavy water No P4 (Treatment 4)	Xheavy water No P4 (Treatment 5)
	With P4	No water With P4 (Treatment 6)	Light water With P4 (Treatment 7)	Medium water With P4 (Treatment 8)	Heavy water With P4 (Treatment 9)	Xheavy water With P4 (Treatment 10)

Figure 5.4: Experimental Design Table for Weight Gain of Golden Torch Cacti[28]

In Figure 5.4, there are two factors in set $F = \{$'No P4', 'With P4'$\}$ and five levels in set $L = \{$'None','Light', 'Medium','Heavy','Xheavy'$\}$. The size of Cartesian product $F \times L$ is 10. Therefore, there are 10 treatments in this experiment, one of which (Treatment 2) is 'Light water; No P4'.

In an experiment, it is important to determine whether two treatments are significantly different, since treatments have great impact on the outputs of the experiment.

Definition 9. *(Significant Difference) For treatments* T_1 *and* T_2*, if there are differences between experimental results under* T_1 *and under* T_2*, and the differences are not attributed to chance, then we define treatment* T_1 *and* T_2 *to be*

Treatment Number	Col 1	Col2	Col3
T_1		b	
T_2	a	b	
T_3	a	b	
T_4	a		
T_5	a		c

Table 5.1: Letter Display Table for $H = \{(T_1, T_4), (T_1, T_5)\}$

significantly different.

After attaining the results under different treatments, experimenters want to analyze the effects of different treatments on their hypothesis. Therefore, they use a set H of treatment pairs to store the significant differences between treatments: $(T_i, T_j) \in H$ if and only if T_i and T_j are significantly different from each other.

Letter display is a common method to display such a set H, in which a *letter display table* Ta is used to exhibit the significant differences between treatments:

1. The rows of Ta are the treatments in the experiment.
2. Each cell of Ta is empty or contains a letter.
3. Each letter can appear in at most one column.
4. If T_i and T_j are significant different then $Ta(i, c) \neq Ta(j, c)$ for every column c.

Table 5.1 is an example of a letter display table for a *significant difference set* $H = \{(T_1, T_4), (T_1, T_5)\}$. In Table 5.1, T_1 and T_2 are not significant different. Therefore $Ta(1, 2) = Ta(2, 2) = b$. T_1 and T_4 are significantly different: Row 1 and Row 4 do not share the same letter in any column.

For further simplification, we use a binary matrix M to represent set H of significant difference between treatments:

1. The rows of matrix M are the treatment indices.
2. $M[i][c] = M[j][c] = 1$ for some column c if $(T_i, T_j) \in H$. Otherwise, for every column c, at most one of $M[i][c]$ and $M[i][c]$ equals to 1.

In Figure 5.5, M is a binary matrix representation for set $H = \{(T_1, T_4), (T_1, T_5)\}$. Compared to the letter display table, binary matrix M takes less space to store the relation of significant difference since M has same number of rows and columns as Ta and the space to store a binary bit is only 1/8 of the space to store a character.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Figure 5.5: Binary Matrix M for $H = \{(T_1, T_4), (T_1, T_5)\}$

Given n treatments where n is a constant, the size of binary matrix M is defined by its number of columns. To minimize the size of M while preserving the relation of significant difference between treatments, we introduce CLD-C problem.

Compact Letter Display (CLD-C) [14]

Input:	A positive integer n and a set H (of size m) of integer pairs where $H = (i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)$ with $1 \leq i_r, j_r \leq n$ for $r = 1, \ldots, m$
Parameter:	None
Question:	Determine a smallest M to display H?

We will show below that CLD-C and Edge Clique Cover are equivalent problems. Due to this equivalence, we can find a solution to CLD-C by solving Edge Clique Cover when given a set H of significant differences between treatments as below:

1. Build a graph $G = (V, E)$, where each vertex $v_i \in V$ is a treatment T_i that appears in H and there is an edge $e_{ij} \in E$ if and only if $(T_i, T_j) \notin H$.
2. Find a minimum edge clique cover for graph G using Algorithm 2.
3. Enumerate each clique in the obtained minimum edge clique cover.
4. Let the size of binary matrix M be defined by the size of a minimum edge clique cover for graph G.
5. We next build M as follows: if v_i is in clique j then $M[i][j] = 1$. Otherwise, $M[i][j] = 0$. Do this until every cell in M is filled.

Given M for H as instructed by above algorithm, assume that M has r rows and c columns.
We show that above algorithm is correct. For this we show that M satisfies the following two constraints:

1. If $(T_i, T_j) \notin H$, $M[i][l] = 1$ and $M[j][l] = 1$ for some $0 \leq l < c$.
2. If $(T_i, T_j) \in H$, $M[i][l] \neq M[j][l]$ or $M[i][l] = M[j][l] = 0$ for all $0 \leq l < c$.

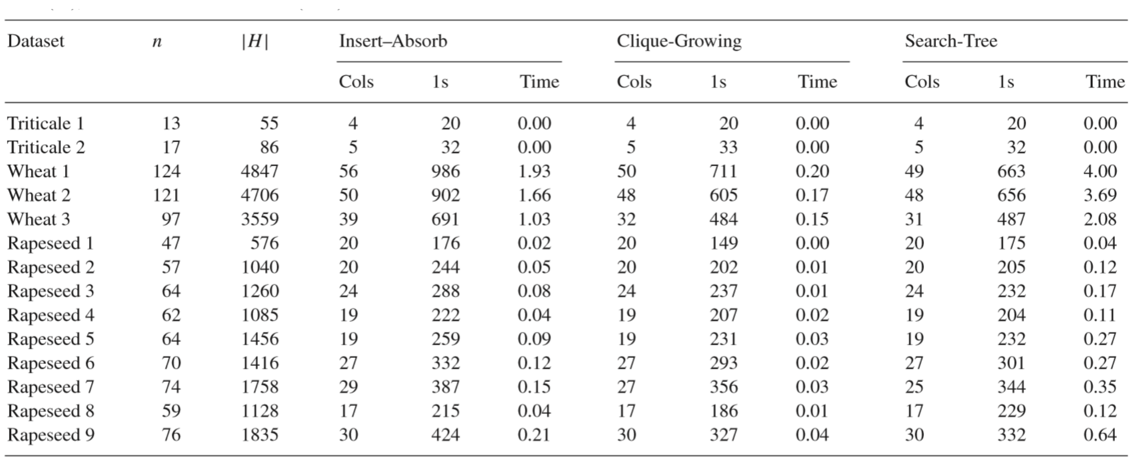
Proof. Assume M is the binary matrix constructed from edge clique cover CC for graph G, where M has r rows and c columns.

We first prove the M satisfies the first constraint. If $(T_i, T_j) \notin H$, there is is an edge $e_{ij} \in E$. When building a edge clique cover CC for G, vertex i and j appear together in some clique $C_l \in CC$, otherwise e_{ij} is not covered. Since $i, j \in C_l$, $M[i][l] = 1$ and $M[j][l] = 1$.

We now prove that the M satisfies the second constraint. If $(T_i, T_j) \in H$, edge $e_{ij} \notin E$. Therefore, for every clique $C_l \in CC$, either neither $i, j \notin C_l$, or only one of i and j is in C_l. Then $M[i][l] = M[j][l] = 0$ or $M[i][l] \neq M[j][l]$ for every $0 \leq l < c$.

Since both constraints are satisfied, M is a correct binary matrix to represent H. □

To solve the corresponding Edge Clique Cover Problem when given an CLD-C, we can apply a search tree algorithm. In paper [14], the authors compare the search tree algorithm to two heuristic algorithms: Insert-Absorb Heuristic and Clique-Growing Heuristic, where Clique-Growing Heuristic applies the concepts of Edge Clique Cover while Insert-Absorb Heuristic does not.



Dataset	n	$\|H\|$	Insert–Absorb			Clique-Growing			Search-Tree		
			Cols	1s	Time	Cols	1s	Time	Cols	1s	Time
Triticale 1	13	55	4	20	0.00	4	20	0.00	4	20	0.00
Triticale 2	17	86	5	32	0.00	5	33	0.00	5	32	0.00
Wheat 1	124	4847	56	986	1.93	50	711	0.20	49	663	4.00
Wheat 2	121	4706	50	902	1.66	48	605	0.17	48	656	3.69
Wheat 3	97	3559	39	691	1.03	32	484	0.15	31	487	2.08
Rapeseed 1	47	576	20	176	0.02	20	149	0.00	20	175	0.04
Rapeseed 2	57	1040	20	244	0.05	20	202	0.01	20	205	0.12
Rapeseed 3	64	1260	24	288	0.08	24	237	0.01	24	232	0.17
Rapeseed 4	62	1085	19	222	0.04	19	207	0.02	19	204	0.11
Rapeseed 5	64	1456	19	259	0.09	19	231	0.03	19	232	0.27
Rapeseed 6	70	1416	27	332	0.12	27	293	0.02	27	301	0.27
Rapeseed 7	74	1758	29	387	0.15	27	356	0.03	25	344	0.35
Rapeseed 8	59	1128	17	215	0.04	17	186	0.01	17	229	0.12
Rapeseed 9	76	1835	30	424	0.21	30	327	0.04	30	332	0.64

Figure 5.6: Comparison of Performances of Insert-Absorb Heuristic, Clique-Growing Heuristic and Search Tree Algorithm on datasets from real-world statistical analyses[14]

Figure 5.6 is the execution results of all three algorithms on different datasets. We can see that Insert-Absorb heuristic has highest running time and leads to a results of poor quality (high number of columns).

Only the search tree algorithm provides the exact solution for CLD-C, but its running time is exponential, which is only slightly smaller than Insert-Absorb heuristic.

Though Clique-Growing heuristic does not guarantee the minimum number of columns in its result, the quality of its output is very similar to the one from search tree algorithm. Moreover, its running time is much smaller than the Insert-Absorb heuristic and the search tree algorithm, which is $O(n^5)$ and can be improved to $O(n^3)$ with an additional sweeping phase [13]. Therefore, Clique-Growing heuristic might be our best choice in this application.

However, if we only need to search for a solution to CLD-C problem of size at most k, we can reduce CLD-C to k- EDGE CLIQUE COVER, then use reduction rules in Section 4.2 on the corresponding k-Edge Clique Cover instance then solve the reduced instance with Algorithm 2.

5.2 k-Edge Clique Partition on graph data compression.

When reviewing the literature, we found a number of application papers for EDGE CLIQUE COVER (e.g., [1, 14]) and VERTEX CLIQUE PARITION (e.g. [3, 11]), but not any paper on EDGE CLIQUE PARTITION.

However, in [1], the authors discuss the application of a variant of EDGE CLIQUE COVER for determining a compact representation of visibility graphs, where a given visibility graph G is represented by a set of cliques and bipartite cliques to reduce its storing space. This paper inspires us to apply EDGE CLIQUE PARTITION to the problem of compact representation of general graphs: when given a graph G, we want to store only its edge clique partition CP, instead of V and E.

When we use an edge clique partition CP accompanied by a set S, which store singleton in G, to compactly represent a given graph G, we store the graph as sets of vertices where repetition of vertices between sets is limited. In our future work, we will investigate whether this representation have an average size of $O(|V|)$ for general graphs $G = (V, E)$.

In this section, we prove the correctness of our approach for a compact representation of general graphs in EDGE CLIQUE PARTITION, and show its efficiency.

When given a graph $G = (V, E)$, we generate a compact representation CP with S for G as follows:

1. Include all singletons from G into a set S.
2. Compute an edge clique partition CP for G.
3. CP accompanied by S is the compact representation for G.

In above algorithm, we need a set S to store singletons before calculating an edge cliqe partition CP for G since we want to preserve those singletons in G, which will be discarded during the computation of CP.

To recover a compressed graph G from CP and S, we include every vertex that appears in CP and S as vertex of G and then connect every pair of vertices $u, v \in C$ for every clique $C \in CP$. The running time for the recovery is linear to the number of edges in E.

Given a graph $G = (V, E)$. We apply Edge Clique Partition to compress G into CP and S then do the recovery as stated above. Assume $G' = (V', E')$ is a graph recovered from CP. If $V = V'$ and $E = E'$, then CP is a correct compact representation for G.

Proof. We first prove $V = V'$. All the singletons in G are included in S in our algorithm and will be recovered in V'. Every vertex v with $deg(v) > 1$ is incident to an edge $e \in E$, which is covered by some $C \in CP$, so v is in C and will be recovered in V' as well. Therefore, in the recovery, no vertex is missing and $V = V'$.

Then we prove $E \subseteq E'$. For every edge $uv \in E$, u, v appear together in some $C \in CP$. In the recovery, every pair of vertices in C is connected by an edge, so $uv \in E'$. Therefore, $E \subset E'$

We now prove $E' \subseteq E$. For every pair of vertices u, v in cliques of CP, there is an edge $uv \in E$, since otherwise the definition of Edge Clique Partition is violated. In the recovery, we only connect pairs of vertices appear in every $C \in CP$ and include the corresponding edges in E'. Therefore, $E' \subseteq E$.

Finally, $E \subseteq E'$ and $E' \subseteq E$ together lead to $E = E'$ and CP is a correct compact representation for graph G.

□

Though there is no known polynomial time algorithm to generate a minimum edge clique partition for G, to store each G in graph database we only need to compute its minimum edge clique partition CP with a search tree algorithm (e.g. Algorithm 1) once and can use CP to recover G many times. Also, the time to recover G from CP is linear to the number of edges in G. Therefore, edge clique partition CP is a sound compact representation for G.

Moreover, there are many existing polynomial time heuristic algorithms to find an edge clique partition for G. Even a possibly non-minimum edge clique partition might save significant amount space for storing G.

We next prove that it takes no more space to store a graph G with edge clique partition CP than with V and E in an adjacency list, since adjacency list is a more space-efficient method to store G than the adjacency matrix.

Proof. Assume there are m edges in G. Further assume it takes 1 unit of space to store a vertex.

In an adjacency list, it takes 2 units of space to store an edge uv, since u appears in the linked-list rooted at vertex v and v appears in the linked-list rooted at vertex u. The total space to store G is $2m$ units.

Now we consider the worst case for our compact representation. If there are no K_3 subgraphs in G, then every clique in CP is of size two, since we cannot find a larger clique in G. It takes $2m$ units to store G with CP. In this case, to store G with CP takes the same amount of space as with an adjacency list.

If there is a K_ℓ subgraph in G such that $\ell > 2$, it takes $\ell^2 - \ell$ units to store the K_ℓ subgraph since there are $\ell^2 - \ell/2$ edges in K_ℓ. However, if we include K_ℓ with a $C \in CP$, it takes only ℓ units of space, which is smaller than storing all edges of K_ℓ in E.

Therefore, if there is a clique of size at least three in CP (even CP is not minimum clique partition), it takes less space to store G with CP than with V and E.

□

Our compact representation in Edge Clique Partition not only uses less space than adjacency list, but also provides convenience for some graph-based problems such as dominating set and graph connectivity.

First we discuss the upper bound that our compact representation provides to dominating set. Assume CP is a size-k edge clique partition for a graph $G = (V, E)$. For each $C \in CP$, ever pair of vertices u, v are connected by an edge uv, and CP includes all $v \in V$. Therefore, picking a v from each $C \in CP$ leads to a dominating set for graph G. And k is an upper bound for the dominating set problem on G.

Graph Connectivity

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

Parameter: None

Question: Whether G is a connected graph or not? That is, whether there is a path between every pair of vertices $v_i, v_j \in G$.

Then we discuss how to apply our compact representation on graph connectivity problem. Assume we determine whether G is a connected graph and given a size-k edge clique partition CP for G. We can make the decision for G using following algorithm:

1. Build a clique graph $G' = (V', E')$ for CP: each $C_i \in CP$ is an vertex $v_i \in V'$ and $v_0 v_1 \in E'$ if and only if $C_0 \cap C_1 \neq \emptyset$.
2. Use Breadth-first-search (BFS) to decide whether G' is a connected graph.
3. G is a connected graph if and only if G' is a connected graph.

In this application, we do not need to recover G and the size of graph connectivity problem is reduced from $|V|$ to k.

Chapter 6

Future Work

We propose two main tasks to be done in the future: first, investigating whether new properties and reduction rules for the clique-based problems discussed can yield faster fixed-parameter or practical algorithms to solve the problems. Second, researching new applications of Edge Clique Partition.

6.1 Developing and Applying lemmas and new reduction rules on Edge clique partition

When doing algorithm research on Edge Clique Partition problems, we have found a few reduction rules and properties from papers (see Section 4.1.1), as well as developed some new rules and properties (see Section4.1.3). However, we have not yet investigated if these rules and properties can speed up known algorithms or lead to new ones that are faster than the best algorithms known solving these problems. In particular, we are interested if the new rules we discussed can further reduce the running time complexity of Edge Clique Partition.

Also, there might be more reduction rules and properties to be discovered for Edge Clique Partition. New reduction rules and properties for Edge Clique Partition are meaningful since Edge Clique Partition has an increasing impact on applications.

6.2 New applications of Edge Clique Partition.

We have encountered a number of applications for Edge Clique Cover, k-Clique, and Vetex Clique Partition. Though there are quite a few articles describing FPT algorithms and their running time complexity of Edge Clique Partition, we have not found any literature that discusses applications of

EDGE CLIQUE PARTITION.

However, we believe that EDGE CLIQUE PARTITION or variants thereof has a great potential for application, since the problem provides some sound ways of partitioning vertices into cliques while satisfying some constraints on the repetition of vertices. We would be surprised if this this property does not help to model a number of real-world problems.

We suggest two possible applications of EDGE CLIQUE PARTITION, one of them is the area of distributed networks and the other one is the graph data compression problem that we described in .... We plan to continue investigating these two applications in addition to looking out for more.

6.2.1 Application of EDGE CLIQUE PARTITION on the Administration of Clustered Communication of a Distributed Network.

For some tasks we require a large amount of computation that is difficult to be executed on an individual computer. In such a situation, we need to consider the application of distributed networks, where a large number of computers work in parallel to execute a given task.

Frequently, there is a need for computers in a distributed network to communicate with each other, and those communications need to be administrated and monitored by master servers: if a communication between two computers is not administrated by a master server, there might appear consensus and inconsistency problems; if the same communication task is administrated by more than two different master servers, besides the possibility of wasting resources, the communication might receive conflicting commands from different master servers. Therefore, there it might be beneficial to consider how to arrange computers into clusters and how to assign one master server to each them.

EDGE CLIQUE PARTITION provides a possible solution to computers clustering problem as follows.

First, we create a graph $G = (V, E)$ where each computer in the distributed network is a vertex $v \in V$ and there is an edge $uv \in E$ if and only if communication between u, v is required to be administrated by a master server. Then we compute a minimum edge clique partition CP for G.

Next we assign a master server to a clique C to administrate all the communications between vertices of C. This ensures that every communication is administrated by exactly one master server while the number of total master servers are minimum.

For future work we suggest to use both real-world distributed network and simulated distributed network in Omnet++ to examine whether Edge Clique Partition provides a good clustering of computers and an efficient assignment of master servers on communications for the clusters. We can evaluate the efficiency by the occupation rate of master servers and the average waiting time of communications to be administrated in the network: a high occupation rate with low average waiting time means a good resource assignment.

6.2.2 Application of Edge Clique Partition on compact representation of graphs.

As discussed in Section 5.2, we can represent a graph G compactly by an edge clique partition CP instead of an adjacency list. We have shown the correctness of our compact representation and demonstrated that the space to store CP is no more than to store the adjacency list.

We have shown the best-case and worst-case space complexity of our compact representation. However, to determine whether our compact representations provide benefits on reducing the space of real-world datasets, we think the average space complexity is also important to us. Therefore, we suggest to investigate the average space complexity of our compact representation in the future, as well as testing our compression approach on real-world datasets.

Also, we have stated a few possible applications that can be computed from our edge clique partition CP without recovering the original graph. To determine whether our compact representation can really benefit those applications is another work to be done.

Chapter 7

Conclusion

Our project investigated clique-based problems and their applications. We first introduced a number of clique-based problems. Our main focus was on Edge Clique Partition and Edge Clique Cover. We not only studied and presented work from the literature on existing FPT algorithms, as well as their running time analysis, but also discussed existing applications found in the literature. Furthermore, we described some additional reduction rules an properties that we have bot seen presented in the literature, and proposed additional applications of the problems.

We believe that in the future more research will be conducted on clique-based problems, on both their algorithms and their applications. Hopefully, this project contributes ideas and motivation to the future research on such clique-based problems.

Bibliography

[1] Agarwal K.P. and Noga A. "Can visibility graphs be represented compactly?" Discrete & Computational Geometry 12.3 (1994): 347-365.

[2] Akkoyunlu, E. A. "The Enumeration of Maximal Cliques of Large Graphs." SIAM J. Comput. 2, 1-6, 1973.

[3] Boginski V., Butenko S., Pardalos P. Mining market data: A network approach. Comput. Oper. Res. (2006) 33(11):3171–3184 Crossref

[4] Bondy A.J. and Murty R. Graph theory with applications. Vol. 290. London: Macmillan, 1976.

[5] Böcker Sebastian: A Golden Ratio Parameterized Algorithm for Cluster Editing. IWOCA 2011.

[6] Branch and bound. (2017, February 7). In Wikipedia, The Free Encyclopedia. Retrieved 06:38, March 20, 2017, from `https://en.wikipedia.org/w/index.php?title=Branch_and_bound&oldid=764241953`

[7] Downey G.R, Fellows R.M. and Stege U. "Parameterized complexity: A framework for systematically confronting computational intractability." Contemporary trends in discrete mathematics: From DIMACS and DIMATIA to the future. Vol. 49. 1999.

[8] Downey G.R and Fellows R.M. Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer, 2013.

[9] Downey G.R, Fellows R.M. and Stege U. "Computational tractability: The view from mars." Bulletin of the EATCS 69 (1999): 73-97.

[10] Feder T. and Motwani R. "Clique partitions, graph compression and speeding-up algorithms." Proceedings of the twenty-third annual ACM symposium on Theory of Computing. ACM, 1991.

[11] Figueroa, A., Bornemann, J., Jiang, T.: Clustering binary fingerprint vectors with missing values for DNA array data analysis. Journal of Computational Biology 11, 887–901 (2004)

[12] Gary M.R. Gary and Johnson S.D. "Computers and Intractability: A Guide to the Theory of NP-completeness." W. H. Freeman, 1979.

[13] Gramm J., Guo J. and Hüffner N., 2006.Data reduction,exact,and heuristic algorithms for clique cover.In:Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX'06), SIAM, Philadelphia, PA, pp. 86–94.

[14] Gramm J. and Guo J. "Algorithms for compact letter displays: Comparison and evaluation." Computational Statistics & Data Analysis 52.2 (2007): 725-736.

[15] Gramm J. and Guo J. "Data reduction and exact algorithms for clique cover." Journal of Experimental Algorithmics (JEA) 13 (2009): 2.

[16] Guo Jiong. "A more effective linear kernelization for cluster editing." Theoretical Computer Science 410.8 (2009): 718-726.

[17] Holyer Ian. "The NP-completeness of some edge-partition problems." SIAM Journal on Computing 10.4 (1981): 713-717.

[18] Intersection graph. $(2016, March22)$. In Wikipedia, The Free Encyclopedia. Retrieved 23:11, August 18, 2016, from `https://en.wikipedia.org/w/index.php?title=Intersection_graph&oldid=711452212`

[19] Jörg F. and Grohe M. Parameterized Complexity Theory (Texts in Theoretical Computer Science. an EATCS Series). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[20] Lin Bingkai. "The parameterized complexity of k-biclique." Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2015.

[21] Michalewicz Z. and Fogel B.D. How to Solve It: Modern Heuristics. Springer, Berlin, Germany, second edition, 2004. ISBN 3-540-22494-7

[22] Mujuni E. and Rosamond F. *Parameterized Complexity of the Clique Partition Problem.* Proceedings of the fourteenth symposium on Computing: the Australasian theory, Volume 77. Australian Computer Society, Inc., 2008.

[23] Niedermeier Rolf. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications).* Oxford University Press, USA, March 2006.

[24] Niedermeier R. and Rossmanith P. "Upper bounds for vertex cover further improved." Annual Symposium on Theoretical Aspects of Computer Science. Springer Berlin Heidelberg, 1999.

[25] Orlin James. "Contentment in graph theory: covering graphs with cliques." Indagationes Mathematicae (Proceedings). Vol. 80. No. 5. North-Holland, 1977.

[26] Rajagopalan S., Vachharajani M. and Malik S. ”Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints.”Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems. ACM, 2000.

[27] Vazirani V.V. Approximation algorithms. Springer-Verlag, 2013.

[28] Weiss A. Neil, Introductory Statistics, Addison Wesley, 2007