GRAPHLET MINING IN BIG DATA

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by
Büşra ÇALMAZ

December 2024
İZMİR

We approve the thesis of **Büşra ÇALMAZ**

Examining Committee Members:

Associate Professor Belgin ERGENÇ BOSTANOĞLU
Department of Computer Engineering, İzmir Institute of Technology

Professor Dr. Cüneyt Fehmi BAZLAMAÇCI
Department of Computer Engineering, İzmir Institute of Technology

Associate Professor Rıza Cenk ERDUR
Department of Computer Engineering, Ege University

Associate Professor Fatih SOYGAZİ
Department of Computer Engineering, Aydın Adnan Menderes University

Assistant Professor Emrah İNAN
Department of Computer Engineering, İzmir Institute of Technology

19 December 2024

Associate Professor Belgin ERGENÇ BOSTANOĞLU
Department of Computer Engineering
İzmir Institute of Technology

Professor Dr. Onur DEMİRÖRS
Head of the Department of
Computer Engineering

Professor Dr. Mehtap EANES
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

First and foremost, I want to express my heartfelt thanks to my thesis advisor, Assoc. Prof. Belgin ERGENÇ BOSTANOĞLU, who has been my greatest guide in academic life. Her constant support, warmth, kindness, understanding, and encouragement throughout my graduate studies have been invaluable. I deeply admire her vision and perspective and am truly grateful for her guidance.

I would like to extend my sincere thanks to my PhD thesis jury members, Prof. Cüneyt Fehmi BAZLAMAÇCI, Assoc. Prof. Rıza Cenk ERDUR, Assoc. Prof. Fatih SOYGAZİ, and Asist. Prof. Emrah İNAN for valuable comments and feedback. I would also like to express my gratitude to Asist. Prof. Işıl ÖZ for kindly allowing me to use the resources of her lab.

I want to express my deepest gratitude to my family for their unwavering love and support. I thank my beloved husband, Fatih ÇALMAZ, for his unconditional support, understanding, and dedication throughout my doctoral journey. His immense patience during my countless hours working and his care for our daughter were invaluable. I genuinely appreciate his encouragement and patience throughout this process.

ABSTRACT

GRAPHLET MINING IN BIG DATA

This thesis explores graphlet counting algorithms, which are crucial for understanding the structural principles of complex networks such as bioinformatics, social networks, and network model evaluation. Counting graphlets in large networks is computationally challenging due to the combinatorial explosion of possibilities, particularly for larger graphlet sizes. To address this, we focus on clique graphlets, fully connected subgraphs, which reveal critical patterns in areas like protein structure analysis, social network modeling, community detection, and spam detection. Counting k-cliques (subgraphs with k nodes) becomes infeasible for large datasets and high k values. Existing exact and approximate algorithms struggle with large k, often failing when k exceeds 10. To tackle these limitations, we propose BDAC (Boundary-Driven Approximations of K-Cliques), a novel algorithm that efficiently approximates k-clique counts using classical extremal graph theorems. BDAC uniquely provides lower and upper bounds for k-clique counts at both local (per vertex) and global levels, making it particularly suited for large, dense graphs with high k values. Unlike existing methods, the algorithm's complexity remains unaffected by the value of k. We validate BDAC's efficiency and scalability through extensive comparisons with leading algorithms on diverse datasets, spanning k values from minor (e.g., 8) to large (e.g., 50). Parallelization techniques enhance its performance, making it highly scalable for analyzing large and dense networks. BDAC offers a significant advancement in k-clique counting, enabling the analysis of previously considered computationally intractable networks.

ÖZET

BÜYÜK VERİDE ALT ÇİZGE MADENCİLİĞİ

Bu tez, biyoenformatik, sosyal ağlar ve ağ modeli değerlendirmesi gibi karmaşık ağların yapısal prensiplerini anlamak için kritik öneme sahip olan alt çizge sayma algoritmalarını incelemektedir. Büyük ağlarda alt çizgelerin sayılması, özellikle daha büyük alt çizge boyutları için olasılıkların kombinatoryel patlaması nedeniyle hesaplama açısından zorludur. Bu zorlukları ele almak için, protein yapısı analizi, sosyal ağ modelleme, topluluk tespiti ve spam tespiti gibi alanlarda kritik desenleri ortaya çıkaran tam bağlı alt çizgeler olan k-klik alt çizgelerine odaklanıyoruz. K-klik'lerin (k düğümlü alt çizgeler) sayılması, büyük veri kümeleri ve yüksek k değerleri için uygulanamaz hale gelmektedir. Mevcut kesin ve yaklaşık algoritmalar, k 10'u aştığında genellikle başarısız olur. Bu sınırlamaların üstesinden gelmek için, klasik ekstremal çizge teoremlerini kullanarak k-klik sayılarını verimli bir şekilde yaklaşık olarak hesaplayan yenilikçi bir algoritma olan BDAC'ı (K-kliklerin Sınır Tabanlı Yaklaşımı) öneriyoruz. BDAC, k-klik sayımları için hem yerel (düğüm bazında) hem de küresel seviyelerde benzersiz bir şekilde alt ve üst sınırlar sağlayarak, özellikle yüksek k değerlerine sahip büyük ve yoğun çizgeler için son derece uygundur. Mevcut yöntemlerin aksine, algoritmanın karmaşıklığı k değerinden etkilenmez. BDAC'ın verimliliğini ve ölçeklenebilirliğini, küçük (ör. 8) ile büyük (ör. 50) arasında değişen k değerlerini kapsayan çeşitli veri kümeleri üzerinde önde gelen algoritmalarla yapılan kapsamlı karşılaştırmalarla doğruluyoruz. Paralelleştirme teknikleri, performansını daha da artırarak, büyük ve yoğun çizgelerin analizinde oldukça ölçeklenebilir hale getirmektedir. BDAC, k-klik sayımı konusunda önemli bir ilerleme sunarak, daha önce hesaplama açısından ulaşılamaz kabul edilen çizgelerin analizine olanak tanımaktadır.

To my lovely daughter, Azra.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The process of extracting usable data from a larger set of raw data is commonly referred to as data mining. Finding important patterns and trends requires examining and analyzing large datasets. The significance of using data analysis to obtain actionable knowledge has grown as datasets get larger for many application areas. As a result, data mining and knowledge discovery have become important research topics in recent years. Data can take many forms, including vectors, tables, graphs, text, images, and more, and can be represented in various ways.

Graph data consists of a collection of nodes connected by edges, which can be used to naturally represent structured and semi-structured data. The data across numerous domains, such as social media (Nocaj, Ortmann, and Brandes 2015; Chakraborty et al. 2018; X. Yang et al. 2021), molecular biology (Huber et al. 2007; Rosselló and Valiente 2005b), chemical reactions (Rosselló and Valiente 2005a; Ralaivola et al. 2005), protein interactions (Jha, Saha, and Singh 2022; F. Yang et al. 2020), and computer networks (McQuillan 1977; Jahnke, Thul, and Martini 2007), can be represented via graphs.

Graphlets, also known as motifs, are small, induced subgraphs representing large data's underlying structure. Analyzing the number of graphlets in a dataset can provide valuable insights into the complex network's properties, functions, and dynamics. For example, this analysis can help elucidate community structures in the context of social media data (Dourisboure, Geraci, and Pellegrini 2009). Similarly, it can be used to study interactions between proteins in bioinformatics and to analyze interactions between molecules in chemistry. Common graphlets such as triangles, cycles, and cliques are widely used in various domains such as spam detection (Leon-Suematsu et al. 2011; Becchetti et al. 2006), link prediction (X. Li et al. 2019), uncovering patterns in biological networks (Betzler et al. 2011; Saha et al. 2010), anomaly detection (Yan et al. 2021; Gibson, Kumar, and Tomkins 2005), social network analysis (Foucault Welles, Van Devender, and Contractor 2010; Son et al. 2012; Han, Pei, and Kamber 2006), conducting clustering (Seeland et al. 2010) and classification (Acosta-Mendoza, Gago-Alonso, and Medina-Pagola 2012) tasks. However, graphlet counting is computationally challenging,

as the number of possible graphlets increases exponentially with their size, leading to a combinatorial explosion. For instance, there are two induced graphlets with three nodes, six induced graphlets with four nodes, 21 induced graphlets with five nodes, and 112 induced graphlets with six nodes. This makes counting large graphlets (with larger nodes) infeasible, particularly for large datasets. Furthermore, large datasets contain millions or billions of nodes and edges, making exhaustive enumeration impractical.

There are lots of algorithms in the literature that can count the occurrences of graphlets up to 4 nodes exactly; there are only a few that can count up to 5 nodes (Ahmed et al. 2015; Jha, Seshadhri, and Pinar 2015; Pinar, Seshadhri, and Vishal 2017; Ribeiro et al. 2021). Due to the complexity of the problem, many algorithms resort to approximation schemes that trade some accuracy for faster execution times. Additionally, some algorithms count global graphlet counting for the entire dataset, while others focus on local graphlet counting in an edge or vertex context. It has been observed that parallel computing can significantly improve execution time and enable scalable graphlet counting computation.

Due to the combinatorial explosion, exact graphlet counting in large datasets is not tractable. Therefore, it is essential to sparsify large datasets while preserving the essential structural properties of a graph and staying computationally manageable. Typically, sampling strategies and statistical methods are employed to estimate graphlet counts close to their exact values. In approximate methods, a suitable sampling size must be selected. The sampling size affects both the computational efficiency and accuracy of the algorithm. The sampling strategies represent the large datasets by a smaller subset and enable the analysis of large datasets impractical to handle using exact algorithms. A well-chosen sample size accelerates analysis, conserves computational resources, and provides meaningful insights.

Each type of graphlet has distinct characteristics, such as its node and edge counts, degree distribution, and overall density. The sampling strategies are built upon these characteristics, so providing a sampling strategy that preserves each graphlet type and size is another challenge. Finding a base structure that encompasses each graphlet type and size is challenging. Consequently, the research shifted focus towards counting cliques, among the most widely used graphlets across various application areas. Fully connected k vertices represent a dense subgraph known as a k-clique.

The cliques in a social or communication network can help to detect communities, spams, or groups of individuals with strong connections (Holland and Leinhardt 1977; Jackson, Rodriguez-Barraquer, and Tan 2012; Cleemput 2012; Yildiz and Kruegel 2012; Lu, Wahlström, and Nehorai 2018; Nedioui et al. 2020; Shi et al. 2021; Gregori, Lenzini, and Mainardi 2012; Jayanthi 2012; Faust 2010; Han, Pei, and Kamber 2006; Pan et al. 2023; Foucault Welles, Van Devender, and Contractor 2010; Son et al. 2012). Similarly, cliques can be used to analyze functional modules, identify protein complexes, and uncover key interactions critical for understanding biological processes in protein networks (X.-L. Li et al. 2005; H. Yu et al. 2006; Chen et al. 2013; Milik, Szalma, and Olszewski 2003; Betzler et al. 2011; Pržulj, Corneil, and Jurisica 2004; Saha et al. 2010). This sheds light on biological pathways and disease mechanisms such as epilepsy prediction (Prokopyev et al. 2007; Iasemidis et al. 2003). Cliques can enhance the accuracy of suggestions in recommendation systems, where they can help identify groups of closely connected users or items with similar preferences (Vilakone, Xinchang, and Park 2019; Vilakone et al. 2018; Manoharan et al. 2020; Gong et al. 2019; Tsourakakis et al. 2011). In fraud detection applications, the collusive groups of actors who engage in fraudulent activities (J. Yu et al. 2023) can be detected with the help of clique counts. Moreover, clique counting finds applications in diverse fields such as graph compression (Buehrer and Chellapilla 2008) and clustering (Duan et al. 2012). However, dealing with the number of k-cliques for larger values of k poses significant algorithmic challenges, mainly due to the exponential growth of the search space associated with large cliques.

There are two algorithms serve as the foundation of clique counting algorithms: the Bron-Kerbosch algorithm (Bron and Kerbosch 1973), the foundation for maximal clique listing, and the ARBO algorithm (Chiba and Nishizeki 1985), designed explicitly for k-clique listing. Many subsequent algorithms either build upon these base algorithms or incorporate enhancements inspired by them (Finocchi, Finocchi, and Fusco 2015; Danisch, Balalau, and Sozio 2018; Jain and C. Seshadhri 2017, 2020; Jain and Tong 2022; Ye et al. 2023). Finocchi et al. (Finocchi, Finocchi, and Fusco 2015) present a MapReduce-based adaptation of the Chiba and Nishizeki's approach (ARBO) (Chiba and Nishizeki 1985) -a k-clique listing algorithm- with degree orientation. Similarly, the kClist algorithm (Danisch, Balalau, and Sozio 2018) enhances the ARBO (Chiba and Nishizeki 1985) algorithm by incorporating parallelization techniques and degeneracy orientation

to eliminate duplicate clique enumeration. Turán-shadow (Jain and C. Seshadhri 2017) introduces an innovative method for approximate clique counting, using Turán's theorem (Turán 1941) to enumerate cliques up to size 10. Meanwhile, Pivoter (Jain and Seshadhri 2020) has revolutionized clique counting by eliminating the need for enumeration, allowing the exact computation of k-clique counts for all k values across different graphs. However, the authors state that it is unsuitable for large datasets such as "$com - lj$" (Leskovec and Krevl 2014) beyond $k > 10$. The Yacc algorithm (Jain and Tong 2022) approximates k-cliques up to 40 by revisiting the Turán-shadow algorithm and incorporating various insights to achieve faster clique counting. The DPColorPath (Ye et al. 2023) algorithm combines the strengths of both exact and approximate solutions. It divides the graph into sparse and dense regions. For the sparse region, it employs exact solutions, utilizing the Pivoter algorithm (Jain and Seshadhri 2020). The algorithm introduces three distinct dynamic programming-based sampling approaches for the dense region.

1.1. Contribution of the Thesis

This section clarifies the contribution of the thesis.

- The graphlet counting algorithms in the literature are analyzed, highlighting the challenges of the problem, as well as the advantages and disadvantages of the proposed algorithm. Detailed examination of these algorithms is given in the scope of the thesis. Additionally, potential research directions are identified.

- As a potential research directions, clique-a special graphlet type- counting algorithms are analyzed, and a comprehensive survey is presented for researchers (Çalmaz and Bostanoğlu 2024b). This survey provides a detailed review of the algorithms.

- We propose an algorithm, BDAC (Boundary-driven approximations of k-cliques), which uses the Turán theorem (Turán 1941) and additional extremal graph theorems (Erdős 1969; Zykov 1949; Kruskal 1963; Reiher 2016) from previous literature to approximate the number of k-cliques (Çalmaz and Bostanoğlu 2024a). The proposed method differs from existing methods in eliminating the need for sampling procedures and repetitive recursive calls. Instead, it provides lower and upper

bounds for local (per-vertex) and global on the k-clique count over the entire range of k-values up to 50. Unlike existing algorithms, our algorithm's complexity is unaffected by the k-values. We evaluate our algorithm by comparing it with state-of-the-art clique counting algorithms across a wide range of k values, from 8 to 50.

- To enhance the BDAC algorithm's performance and scalability, we also provide the parallel version of BDAC through CPU-based parallelization techniques. The BDAC algorithm is compared with the state-of-the-art algorithms in the literature and has shown competitive performance. The parallel version enhances the performance and scalability, and the algorithm becomes suitable for large and dense datasets. We are preparing a conference paper that focuses on the parallel implementation of the BDAC algorithm.

1.2. Outline of the Thesis

The thesis is structured as follows: Chapter 2 introduces essential terminology. Chapter 3 reviews the graphlet mining problem, covering challenges, applications, preprocessing steps, and algorithmic variants. It also surveys existing graphlet counting algorithms. Chapter 4 discusses the problem, presents the applications in different domains, reviews and discusses the relevant literature on clique counting algorithms, and finally, presents the motivation behind this thesis. Chapter 5 details the proposed k-clique approximation algorithms. The experimental result of the proposed algorithm is presented in Chapter 6. Finally, Chapter 7 concludes the paper and suggests future research directions.

CHAPTER 2

PRELIMINARIES

This section presents concepts and terminology associated with graphlet counting that will be employed throughout this study. Our research focuses only on simple, connected, non-isomorphic subgraphs on a single graph.

2.1. Graph

An undirected simple graph $G(V, E)$, comprises a collection of vertices (or nodes) denoted by $|V|$, with cardinality $\mathbf{n} = |V|$, and a set of edges represented by E, with cardinality $\mathbf{m} = |E|$. A k-graph refers to a graph with k vertices.

The **degree** (denoted as $d(u)$) of a node u is the number of its neighboring vertices, in other terms, the count of adjacent edges connected to that node. The **degeneracy** (d) of a graph G is the smallest number d, so the vertices can be arranged in a sequence where each vertex is adjacent to at most d of the vertices that come before it in the sequence.

2.1.1. Directed and Undirected Graph

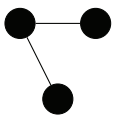
A directed graph comprises nodes and edges, with the latter having a direction that indicates a one-way flow of information or a relationship between the connected nodes. In contrast, an undirected graph lacks such direction. Figure 2.1a represents the undirected graph example, and Figure 2.1b is an example of a directed graph.

(a) Undirected graph. (b) Directed graph.

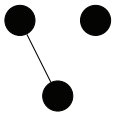
Figure 2.1. Undirected and directed graph example.

2.1.2. Connected and Unconnected Graph

Suppose every pair of vertices is connected via a path; such a graph is called connected. If any pair has no path between them, such a graph is called unconnected or disconnected. Figure 2.2a and Figure 2.2b illustrate the connected and unconnected graphs, respectively. Suppose a connected graph, if there are no multiple edges between any pair of vertices or a node to itself, is called a **simple graph**. A **path** is a series of connected vertices, each connected to the next by an edge. Suppose a graph, a path starts from a vertex and ends with the same vertex; this graph contains a **cycle**. This graph is called **acyclic** if no such path exists.



(a) Connected graph.



(b) Unconnected graph.

Figure 2.2. Connected and unconnected graph example.

2.1.3. Tree, Forest, and Arboriticy

A connected graph with no cycles is called a ***tree***. A disconnected acyclic graph is called ***forest***. Any of the two vertices of a forest are connected by at most one path. Figure 2.3 is an example of a forest consisting of three trees. The minimum number of forests required to cover the graph is known as the **arboricity** of that graph.

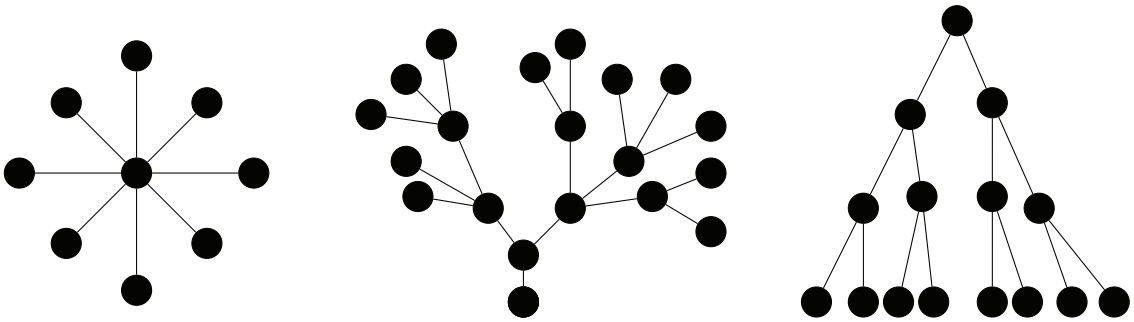


Figure 2.3. An example of a forest with three trees.

2.2. Subgraph

Given a graph $G = (V, E)$, a graph G_s=(V_s,E_s) is a subgraph of G if and only if $V_s \subseteq V$ and $E_s \subseteq E$. Figure 2.4a, and Figure 2.4b are the examples of a graph and subgraph.

(a) G. (b) G_s.

Figure 2.4. A graph G and a subgraph G_s example.

2.2.1. Induced vs Non-induced Subgraph

Suppose a subgraph that consists of a subset of vertices of another graph. If this subgraph contains all edges between pairs of vertices within the subset, this subgraph is called **induced subgraph**. Otherwise, it is called as **non-induced subgraph**.

2.2.2. Subgraph Isomorphism

Suppose G and H are two graphs. There is an isomorphism between G and H if there is a bijection f between the vertices ($f : V(G) \rightarrow V(H)$), that is, any two adjacent vertices (u and v) in graph G, if and only if $f(u)$ and $f(v)$ are adjacent in H. These

two graphs are called **isomorphic** since they are topologically identical. An isomorphism from a graph to itself is called **automorphism**. **Embeddings** in a graph dataset refer to isomorphic graphs of a subgraph within that dataset.

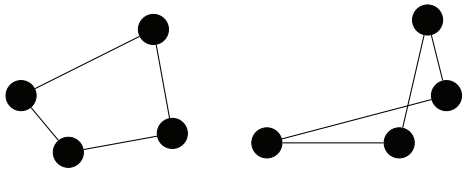


Figure 2.5. Isomorphic graphs.

2.2.3. Orbit

In a graph, G, the equivalence classes of vertices under automorphism actions are referred to as **vertex orbits**, while those of edges are termed **edge orbits**. If two edges belong to the same orbit, their degrees must be identical. Orbits represent unique configurations of subgraphs, where nodes sharing the same label in Figure 2.6 share the same orbit and possess topological equivalence.

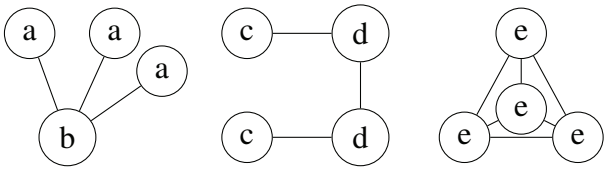
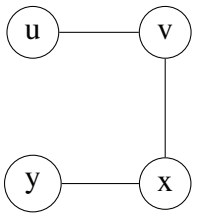


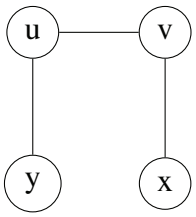
Figure 2.6. The examples of vertex orbits.

In Figure 2.7, if we disregard the vertex labels u and v, the two graphlets in Figure 2.7a and Figure 2.7b are topologically identical. However, if we consider the vertex labels, Figure 2.7b represents a non-local edge orbit of the 3-path shown in Figure 2.7a. When two edges are part of the same orbit, they must have the same degree pair. In calculating the

local count of graphlets, certain types of graphlets with distinct edge orbits are excluded, even if they are isomorphic to each other.



(a) 3-path.

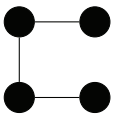


(b) Non-local edge orbit of 3-path.

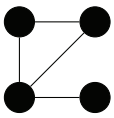
Figure 2.7. The examples of edge orbits.

2.3. Graphlets

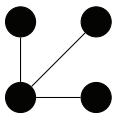
Graphlets (or motifs) are induced subgraphs that are connected and consist of a small number of nodes. A k-graphlet refers to the k nodes graphlet. The Figures between 2.8a and 2.8f are examples of 4-graphlets.



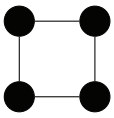
(a) 3-path.



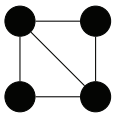
(b) Tailed triangle.



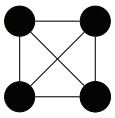
(c) 3-star



(d) 4-cycle.



(e) Diamond.



(f) 4-clique.

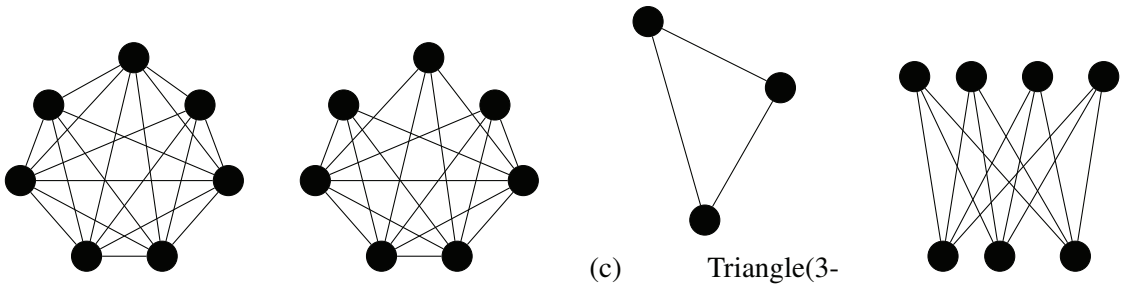
Figure 2.8. Example of 4-graphlets.

2.3.1. Clique

A clique is a maximal subgraph where each vertex pair is connected via an edge, defining a complete subgraph. A k-clique indicates a clique with k vertices. There are different types of cliques.

- **Maximal clique:** A maximal clique is a clique that cannot be expanded by adding another adjacent vertex, i.e., it is not a subset of any larger clique in the graph.
- **Bi-clique:** Suppose a bipartite graph whose vertices can be divided into two distinct, non-overlapping groups if every vertex of the first group is connected to every vertex of the second group. Such a graph is called a **complete bipartite graph** or **bi-clique**.
- **Quasi-clique:** A quasi-clique is a group of vertices in a graph where each vertex is connected to many vertices; unlike the cliques, it is not necessarily to all.

Figure 2.9 illustrates the clique and its different types.



(a) Clique. (b) Quasi-clique. (c) Triangle(3-clique). (d) Bi-clique.

Figure 2.9. Clique and its types.

2.3.2. GFD Vector (Graphlet frequency distribution)

The Graphlet Frequency Distribution (GFD) is a vector of a graph that characterizes the relative frequencies of the various graphlets in the graph. Constructing GFD requires the frequencies of all graphlets of sizes 3, 4, and 5. There are 29 connected and undirected

k-graphlets for $k = 3, 4, 5$ (see Figure 2.10). Assume f(i) be the frequency of graphlet g_i where i $\in \{1, 2, ..., 29\}$. Each entry in the GFD is calculated with this formula:

$$log(\frac{f(i)+1}{\sum_{i=1}^{29} f(i)+29}) \tag{2.1}$$

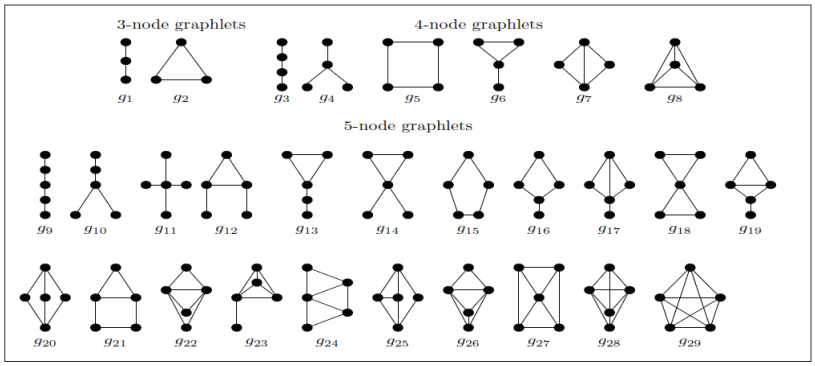


Figure 2.10. All 3,4,5 connected and undirected graphlets (Rahman, Bhuiyan, and Al Hasan 2014).

In this formula 2.1, 1 is added to the frequency of each graphlet to normalize the frequency of each graphlet and avoid the undefined problem $\log 0$. The resulting value is then divided by the total frequency of all graphlets and adjusted by adding 29 (the number of graphlets considered) to account for the smoothing factor applied to each graphlet.

2.3.3. Edge Density, Triangle Density, and Induced Density

The **edge density** of a graph is the ratio of the number of edges to the number of possible edges based on the number of vertices in the graph. It is calculated by dividing the number of actual edges (m) by the total number of possible edges (n), mathematically defined as $\frac{m}{\binom{n}{2}}$.

Let's assume that t_v denotes the number of triangles that form by node v, and d_v refers to the degree of node v. The **triangle density** can be computed using this formula: $\frac{t_v}{\binom{d_v}{2}}$.

Suppose H is a graph with k vertices and G is a graph with n vertices. The induced

copies of H in G are denoted as $Ind(H;G)$. The **induced density** of H is calculated as:

$$d(H;G) = \frac{|Ind(H;G)|}{\binom{n}{2}}$$ (Huang et al. 2016).

In this paper, the k-clique density of node v is denoted as $d(K_k)_v$, k-clique density of graph G is denoted as $d(K_k;G)$.

2.4. Extremal Graph Theorems on Clique Existence

This section presents the theorems that provide insights into the conditions under which cliques are present. These methods are instrumental for approximation techniques. Several algorithms leverage these theorems for approximation purposes. In our work, we have applied these theorems to enhance our approaches. To access the formal proofs of the referenced theorems, refer to the provided references, which offer thorough and rigorous demonstrations of their validity.

2.4.1. Turán's Theorem (Turán 1941)

In extremal graph theory, Turán posed a question regarding a positive number n and a graph F. He asked for the maximum number of edges a graph with n vertices can have without containing graph F as a subgraph. Turán provided a complete solution for the case where F is a clique. In other words, Turán's theorem focuses on determining whether a graph contains a complete subgraph (clique) of a given size k using edge density information.

Theorem 1 *For any integer k,* $d(K_k;G) = 0$ *if* $d(K_2;G) \leq \frac{k-2}{k-1}$.

The expression $\frac{k-2}{k-1}$ is denoted as Turán threshold. This threshold indicates the maximum number of edges a graph can have without k-clique. For clarity, if the edge density $d(K_2;G)$ satisfies the Turán threshold, the graph contains at least one k-clique (K_k).

2.4.2. Erdős's Theorem (Erdős 1969)

Erdős' theorem states that if the edge density of a graph G exceeds a certain threshold relative to k, then it indicates the minimum number of k-cliques present in the

graph.

Theorem 2 *A graph with n vertices, if* $d(K_2; G) > \frac{k-2}{k-1}$*, this graph contains at least* $(\frac{n}{k-1})^{k-2}$ *k-cliques.*

This theorem provides a lower bound for the number of k-cliques in a graph that meets the Turán threshold.

2.4.3. Zykov's Theorem (Zykov 1949)

Zykov generalized Turán's theorem. Let's refer this expression $\frac{(k-1)(k-2)...(k-r)}{(k-1)^r}$ as Zykov threshold. Another point of view is that if K_r density of a graph $d(K_r; G)$ satisfies the Zykov threshold, this graph contains at least one k-clique.

Theorem 3 *For integers* $2 \leq r \leq k$*, if* $d(K_k; G) = 0$ *then* $d(K_r; G) \leq \frac{(k-1)(k-2)...(k-r)}{(k-1)^r}$*(Huang et al. 2016).*

If we know the density for $r > 2$, the Zykov theorem provides a much lower threshold than the Turán threshold. For $r = 2$, it corresponds to the Turán threshold, as observed.

2.4.4. Kruskal-Katona Theorem (Katona 1987; Kruskal 1963)

Based on the known density of r-cliques, the theorem provides an upper bound on the density required for the existence of k-cliques.

Theorem 4 *if* $d(K_r; G) = \alpha$ *then* $d(K_k; G) \leq \alpha^{k/r}$

This theorem establishes an upper bound on the number of k-cliques within a given graph. We can derive the upper bound by applying the following formula, where $|K_k|$ denotes the count of k-cliques, n denotes the number of vertices.

$$|K_k| \leq (\alpha^{k/r})\binom{n}{k}.$$

2.4.5. Reiher's Theorem (Reiher 2016)

Reiher addresses the question by determining the minimum number of k-cliques that must exist in graphs with n vertices and more edges than $\frac{k-2}{2(k-1)}n^2$. Consider a graph

G with n vertices and m edges. Let $\gamma = m/n^2$. Thus, if $\gamma > \frac{k-2}{2(k-1)}$, How many k-cliques does this graph contain at a minimum, guaranteed?

Theorem 5 *If $k \geq 3$ and $\gamma \in \left[0, \frac{1}{2}\right)]$, then every graph on n vertices with at least γn^2 edges contains at least*

$$\frac{1}{(s+1)^k}\binom{s+1}{k}(1+\beta)^{k-1}(1 - (k - 1)\beta)n^k$$

k-cliques, where $s \geqslant 1$ is an integer with $\gamma \in \left[\frac{s-1}{2s}, \frac{s}{2(s+1)}\right]$ and $\beta \in \left[0, \frac{1}{s}\right]$ is implicitly defined by $\gamma = \frac{s}{2(s+1)}(1 - \beta^2)$ (Reiher 2016).

According to this theorem, in a graph, if we know the number of vertices (n) and edges (m), we calculate γ as $\gamma = m/n^2$, so by rearranging the interval $\gamma \in \left[\frac{s-1}{2s}, \frac{s}{2(s+1)}\right]$ in terms of s, we obtain an interval for s as follows:

$$\frac{2\gamma}{1-2\gamma} \leq s \leq \frac{1}{1-2\gamma}$$

Then, we compute β by substituting γ and s into the equation. Finally, we The theorem indicates that, it can predict the existence of any k-cliques if $\gamma > \frac{k-2}{2(k-1)}$. The $\frac{k-2}{k-1}$ represents the Turán threshold. If $s < k - 2$, the binomial coefficient $\binom{s+1}{k}$ becomes zero. According to Turán's theorem, this scenario does not predict the existence of k-cliques.

2.5. Parallel and Distributed Computing Paradigms

It is inevitable to harness the power of parallel architectures to provide scalable approaches that can reduce the computing time needed. Parallelization strategies comprise methodologies designed to break down intricate computational tasks into smaller, manageable components, executed concurrently to bolster efficiency and tackle scalability challenges. Leveraging shared memory systems, MapReduce frameworks (Dean and Ghemawat 2004), and distributed platforms like Hadoop harness computational resources. These methods decompose computations into parallel sub-tasks executed simultaneously across distributed systems. Significant performance enhancements are achieved by distributing workload and executing tasks concurrently, circumventing the constraints of sequential processing and effectively addressing computationally intensive problems. Multiple parallelization strategies exist, but this work focuses on the methods used in clique counting. This chapter introduces types of parallel and distributed computing paradigms.

Different parallel platforms offer different advantages and are better suited for specific strategies.

2.5.1. Shared Memory

Shared memory can run on single processor systems, parallel multiprocessors, or clustered microprocessors. The workers use the same memory resources for higher performance. The communication is accomplished via Inter-Process Communication (IPC) technology. Shared memory is not highly scalable, and data consistency can be an issue. In addition, the number of cores is usually deficient compared to distributed memory environments.

2.5.2. Distributed Memory

In this system, multiple (heterogeneous) machines speed up the computation in a parallel cluster. Work-sharing can be done in two ways: One is to have a master node mediating work-sharing, and the other is to have workers who steal work directly from each other. Communication is accomplished by sending messages over the network (message passing interface(MPI)), so network bandwidth is a bottleneck in these systems.

Map-reduce paradigm (Dean and Ghemawat 2004) simplifies parallel processing by the map and reduces semantics. The map phase deals with the splitting and mapping of data, while the Reduce phase shuffles and reduces the data. It partitions data randomly across machines and achieves data-parallel computing. It is not generally well suited for dependent computation due to the random data partition. It has a high I/O cost of data shuffling at every iteration, so it is unsuitable for iterative computation algorithms. Hadoop (Borthakur 2007) is an open-source framework that implements the MapReduce model, consisting of the Hadoop Distributed File System (HDFS) for distributed storage and the Hadoop MapReduce framework for distributed processing, handling task scheduling, data partitioning, and fault tolerance.

2.5.3. Graphics Processing Units (GPUs)

GPUs are specialized hardware devices optimized for parallel processing. They have many parallel threads and many cores, so while they seem well suited for parallel tasks, GPUs are known to be challenging to program. With thousands of processing cores, GPUs are particularly adept at executing numerous computations simultaneously. By offloading computations from the CPU to the GPU, parallel tasks can be processed more efficiently. This parallel processing capability enables significant performance improvements, especially for tasks suitable for parallel execution. GPUs distribute workloads across multiple cores, allowing for concurrent execution of independent sub-tasks, which leads to enhanced efficiency and faster processing times. Irregular data accesses are one of the significant challenges affecting performance in GPUs.

In conclusion, the preliminary section has laid out the basic graph terminology, including graphs, subgraphs, graphlets, and extremal graph theorems, necessary to clarify the concept of this study. It lays the foundation for understanding the development of k-clique counting in the literature, its challenges, and our proposed algorithm. This understanding is critical as we continue with more extensive analysis and comparisons in subsequent chapters.

CHAPTER 3

GRAPHLET MINING

This section formally defines the graphlet mining problem, discusses the challenges, explores various applications, and details a common preprocessing step. It then examines the variants of this preprocessing step based on the different characteristics of algorithms.

3.1. Problem Definition, Challenges and Applications

Graphlet mining algorithms aim to count the frequencies of small, induced connected subgraphs H, also known as motifs or graphlets, within a dataset G. The counting can be achieved by listing and checking all induced graphlets in the dataset or using combinatorial methods without examining each one individually. Finding common subgraphs in graph datasets is a well-researched problem in data mining. Having a scalable and exact algorithm is crucial, especially for large-scale datasets. However, due to the problem's complexity, counting graphlets with five or more nodes in large datasets is very challenging. As the size of the graph or the desired clique size increases, the problem becomes exponentially more difficult. The complexity of graphlet counting is O($\mid V \mid \triangle^k$), where $\triangle$ is the max degree of a graph, k represents the possible subgraphs count, $\mid$ V $\mid$ denotes vertex count.

Graphlet mining has extremely high computational complexity because it involves solving two intractable problems:

- **Computational expense**: Counting the frequency of a given graphlet requires finding all instances of that graphlet in the dataset. This process involves subgraph isomorphism, which is an NP-complete problem (Garey, Johnson, and Stockmeyer 1974).

- **Combinatorial explosion:** The algorithm's complexity is influenced by the number of nodes and edges in the input. The number of possible graphlets increases super-exponentially with their size (Table 3.1).

Table 3.1. Distinct graphlet counts for different numbers of vertices.

Vertex Count	Graphlet Count
2	1
3	2
4	6
5	21
6	112
7	853

A dataset's graphlet frequencies (or counts) provide insight into complex networks. They reveal details about the properties, structure, and roles of individual nodes and their relationships in areas as diverse as social, biological, and computer networks, including tasks such as community or anomaly detection. In social networks, they help dissect crucial structural aspects such as social balance, the strength, and stability of ties, and trust (Granovetter 1983) or detect event precursors (Abou Jamra, Savonnet, and Leclercq 2021). In biological networks, they aid in uncovering the functions of proteins (Milenković and Pržulj 2008). Moreover, graphlets can function as features to help tasks like graph clustering (Schaeffer 2007) or classification (Vishwanathan et al. 2010), as well as improving community (Lu, Wahlström, and Nehorai 2018; Tsourakakis 2015), and anomaly detection (Abou Jamra, Savonnet, and Leclercq 2021). In large and complex datasets, examining the number of small subgraphs provides a powerful method for revealing hidden patterns and structures in these complex networks.

3.2. Preprocessing: Graph Orientation

In graphlet counting algorithms, a common approach to reducing the search space involves converting an undirected graph into a **directed cyclic graph (DAG)** based on an ordering technique. The objective is to attain an acyclic orientation of an undirected graph, where each edge is assigned a direction, ensuring the absence of any directed cycles. This orientation is performed using a specific ordering of the vertices. The edges between any two vertices are directed from the lower vertex to the higher vertex based on their position in the ordering, ensuring no cycles are formed. This orientation technique aims to reduce

the number of outgoing or incoming edges for each vertex, thus reducing search space and eliminating duplicate graphlet discoveries. Let's explain it with some sample codes to be more descriptive. The Algorithm 1 presents the brute-force triangle counting algorithm on a undirected graph.

Algorithm 1 Triangle counting without orientation (Seshadhri and Tirthapura 2019)

```
1: for all vertices u do
2:     for all pairs of neighbors v, w do                ▷ this is a wedge
3:         if (v, w) is an edge then
4:             Check if (v, w) is an edge
5:         end if
6:     end for
7: end for
```

Performing an edge lookup for every wedge incurs a high cost, especially for vertices with high degrees. The total number of edge lookups is given by $\Sigma_v \binom{d_v}{2}$, where d_v denotes the degree of vertex v.

Algorithm 2 Triangle Counting with Out-Neighbors (Seshadhri and Tirthapura 2019)

```
1: for all vertices u do
2:     for all pairs of out-neighbors v, w do
3:         Check if (v, w) is an edge
4:     end for
5: end for
```

Algorithm 2 presents the triangle counting algorithm on a directed acyclic graph (DAG). With the introduction of orientation, the algorithm significantly reduces the degrees of nodes by considering only outgoing edges (outdegree < degree), thus reducing the number of edge lookups. This provides a significant improvement in graphlet counting. For example, a triangle consists of three edges, each triangle rediscovered three times, once for each of its edges. With orientation, only checking outgoing edges and the neighbors with higher order eliminates duplicate enumeration.

There are three methods commonly employed to perform orientation, which based on ordering techniques:

- **Degree orientation:** In this technique, nodes are ordered according to priority. This priority depends on the degree of the nodes; that is, nodes with higher degrees have higher priority by resolving ties based on the vertex ID, which ensures uniqueness. So, all the directed edges point from vertices lower in the ordering to higher ones.

- **Degeneracy orientation:** This technique iteratively removes min-degree vertices and updates all remaining vertice degrees. Nodes are prioritized based on removal time. Ties are resolved based on the vertex ID to ensure uniqueness. The goal is to order the vertices so each vertex has fewer neighbors than the preceding one in the sequence. Table 3.2 presents the wedge counts of different datasets after degree and degeneracy orientations (Seshadhri and Tirthapura 2019). An out-wedge refers to a graphlet consisting of a node with its outgoing neighbors and their connecting edges. This table demonstrates a significant reduction in the number of wedges after orientation, particularly for large datasets.

Table 3.2. Wedge counts after degree and degeneracy orientations.

Graph	Wedges	Out-Wedge (degree)	Out-Wedge (degeneracy)
ca-AstroPh	1.3E7	2.0E6	1.9E6
cit-Patents	3.4E8	5.1E7	4.4E7
web-Google	7.2E8	1.7E7	1.6E7
LiveJournal	7.3E9	6.7E8	6.4E8
tech-as-skitter	1.6E10	9.5E7	8.7E7
com-Orkut	4.6E10	4.0E9	3.8E9

- **Color-based orientation:** This technique employs a greedy graph coloring algorithm (Hasenplaugh et al. 2014; L. Yuan et al. 2017) to assign a color to each vertex between $1, 2, ..., m$ using m colors. The aim is to assign different colors to two adjacent nodes. The graph orientation is performed based on this ordering (R. Li et al. 2020).

3.3. Graphlet Mining Variants

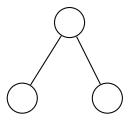
In the literature, graphlet counting algorithms provide various solutions to reduce problem complexity, develop more efficient algorithms, or meet specific objectives. These variations can be categorized based on the algorithmic output:

- Induced or non-induced graphlet counting,

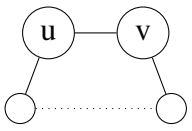
- Global or local graphlet counting,
- Exact or approximate graphlet counting,
- Parallel graphlet counting.

3.3.1. Induced or Non-Induced Graphlet Counting

The literature features numerous algorithms that count induced or non-induced graphlets or both. Typically, non-induced subgraphs are more straightforward to count, but induced subgraphs are more informative and valuable. For example, in Figure 3.1b, the presence of the dotted edge signifies a 4-cycle, whereas its absence indicates a 3-path. The total number of wedges (Figure 3.1a) induced on all edges represents the combined frequencies of 4-cycles and 3-paths. Without distinguishing between induced and non-induced graphlets, it becomes challenging to separate the frequencies of graphlets like the 3-path and the 4-cycle, as illustrated in Figure 3.1b. However, these graphlets are individually significant and often require individual counts. For instance, using graphlet counts as a cost function in an optimization algorithm can yield network topologies that resemble real brain networks (Sporns and Kötter 2004). Notably, for any size k, an invertible linear transformation exists between the vectors of induced and non-induced counts (Figure 3.2). Consequently, one can also derive induced subgraph counts by obtaining non-induced subgraph counts.



(a) Wedge.



(b) 3-path and 4-cycle(with dotted edge) graphlet.

Figure 3.1. An example of induced and non-induced graphlet.

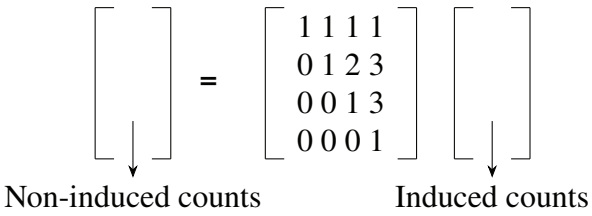


Figure 3.2. Linear transformation between vectors of induced and non-induced counts.

3.3.2. Local or Global Graphlet Counting

Local subgraph counting focuses on the frequencies of a graphlet within the neighborhoods of an edge or vertex. Tasks such as machine learning, link prediction, classification, and clustering require a comprehensive set of vertex features. To generate such a feature set, local counts are necessary. This means that for every vertex v or edge e in a dataset, the count of the graphlets involving v or e (also called vertex orbit counts or edge orbit counts) must be determined. For example, how many 4-cycle patterns are formed by the edge e or vertex v, and how many triangles do they participate in? The local counts are extremely difficult to obtain and require algorithms that obtain much finer-grained information, and the output size makes it difficult to design scalable algorithms.

Global graphlet counting focuses on the total frequency of graphlets within a large input dataset. While local counting focuses on neighborhood-specific occurrences of graphlets and provides insight into the local structure and connectivity around individual vertices or edges, global counting provides a broader perspective by considering the overall frequency of graphlets.

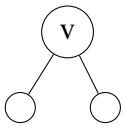
3.3.3. Exact or Approximate Graphlet Counting

In the literature there are plenty of graphlet counting algorithm either provides exact counts or approximate counts. Due to the complexity of the problem, exact counting methods are preferred when accuracy is critical and for smaller datasets. In contrast, approximate methods are generally used for large datasets where computational resources are limited and exact methods are infeasible or ineficcient.

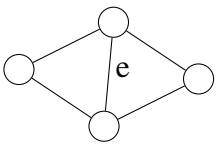
3.3.3.1 Exact Graphlet Counting

Exact techniques meticulously identify all induced graphlets or ascertain their presence within a graph. There are two distinct approaches to exact counting:

- **Enumeration approach:** This approach aims to identify and list all isomorphic instances of the graphlet. However, solving the subgraph isomorphism problem is NP-complete (Garey, Johnson, and Stockmeyer 1974), making this method impractical for large datasets.
- **Analytic approach:** Instead of explicitly enumerating every connected k-size graphlet, this method decomposes each graphlet into smaller components, such as common neighbors or triangles connecting the two vertices. It then counts only these smaller graphlets and uses combinatorial arguments to derive the count of all others. For example, the non-induced count of wedges formed by node v can be computed using the provided formula given in Figure 3.3a in linear time. The formula in Figure 3.3b allows us to find the number of diamond patterns formed by an edge e. The binary combination of triangle counts formed by an edge gives the diamond count.



(a) The number of wedges formed by v : $\binom{d_v}{2}$



(b) The number of diamonds formed by e : $\binom{t_e}{2}$

Figure 3.3. Examples of counting graphlets without enumeration.

3.3.3.2 Approximate Graphlet Counting

The approximate counting methods offer valuable estimations for analyzing large, intricate graphs in contrast to the exact methods, which obtain the actual value—the research objectives and the graph's scale guide the methodology choice. Exact techniques

are well-suited for smaller or medium-sized graphs, while approximate counting is advantageous for exploring larger network structures. Current state-of-the-art algorithms for determining exact subgraph frequencies can take hours or even days to process extensive networks. Given the increasing volume of data, counting all possible subgraphs in datasets like Facebook and Twitter is impractical. In such scenarios, approximate results are also valuable. To address this challenge, subgraph counting research has shifted towards approximating these frequencies, striking a balance between accuracy and time efficiency.

This study examines various graph sparsification and sampling methods as approximation techniques. These methods are outlined below:

- **Spectral Sparsification**: For an undirected graph $G = (V, E)$ with the set of nodes $V = 1, 2, ..., n$ and the set of edges $E \subseteq VxV$, let A be the adjacency matrix with $A_{i,j} = 1$ if $(i, j) \in E$ and $A_{i,j} = 0$ otherwise. Let D be the diagonal matrix with node degrees $di = \sum_{j \in V} A_{i,j}$ on the diagonal. Laplacian of original graph G can be defined as $L_G = D - A$. Laplacian L_H of a sparsified graph H can be defined similarly.

  In this method, two graphs are considered close if their Laplacian matrices are close as linear operators. The purpose of spectral graph sparsity is to preserve the Laplacian quadratic form. More formally, a spectral sparser H is a subgraph of the original G whose Laplacian quadratic form is approximately the same as that of the G on all real vector x inputs. Laplacian quadratic form, which on $x \in \mathbb{R}^n$ as follows:

  $$x^T L_G x$$

  Then H satisfies the following inequality for every $x \in \mathbb{R}^n$, known as the spectral similarity (Spielman and Teng 2011). The concept of effective graph resistance is derived from electrical circuit analysis, which is defined as the accumulated effective resistance between all pairs of vertices. The effective resistance distances between all pairs of vertices are similar in spectrally similar graphs (Batson et al. 2013). The goal is to minimize the total effective resistance, the sum of the resistances between all pairs of nodes.

$$(1-\varepsilon)x^T L_G x \leq x^T L_H x \leq (1+\varepsilon)x^T L_G x \tag{3.1}$$

This method has a robust theoretical guarantee but has a severe drawback. When applied to extensive networks, access to the entire graph is required to calculate the effective resistances of all edges. Also, the computation requires a complex linear system solver by Spielman and Teng (Spielman and Teng 2011), which is not easy to implement in practice. Due to this problem, various methods have been developed that offer more straightforward and faster solutions.

- **Monte-Carlo Markov Chain Sampling**: Monte Carlo is a technique that randomly samples a probability distribution and approximates a desired quantity. On the other hand, the Markov chain is a systematic method for constructing a sequence of random variables in which the current value is probabilistic dependent on the value of the previous variable. By combining these two methods, Markov Chain Monte Carlo (MCMC) sampling provides a class of algorithms to approximate the posterior distribution of a parameter of interest by systematic random sampling in a high-dimensional probabilistic space. A Markov Chain is a mathematical process that transitions from one state to another. $Pr(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = Pr(X_{n+1} = x|X_n = x_n)$

  X is a random variable with a defined range of values (states) in a state space S, and X_t specifies the value (state) of X at t(discrete) time. There are transition probabilities between a pair of states in S, depending only on the current value(state) of X. These probabilities can be expressed in a Transition Probability Matrix (T) matrix. The entry $T(i, j)$ denotes the transition probability from state i to state j. For all $i, j \in S, 0 \leq T(i, j) \leq 1, and \sum_j T(i, j) = 1$.

  The key features of a Markov process are that it is random, and at each step in the process, the future state depends only on the current state of the process, not on the past. This method allows it to narrow in on the quantity from the distribution, even if there are many random variables. Random walk is used to construct the Markov sample chain. With this random walk, distribution is sampled repeatedly in small steps, independent of the previous movement and therefore memoryless.

This process then defines a distribution or estimates fixed parameters such as means, variances, and expected values.

A Markov chain converges to a stationary distribution if its probability of being in a given state is independent of its initial state ($\pi = \pi T$) (Goodfellow, Bengio, and Courville 2016).

- **Random Sampling**: In the random sampling method, vertices or edges are uniformly selected regardless of their attributes or relevance within the graph. This method provides an unbiased estimation and graph representation but risks the critical structural components or nodes crucial to graph dynamics that need attention.

- **Edge Sampling**: In the edge sampling method, some edges are eliminated, preserving as much as possible the key characteristics and properties of the original graph, such as sparsity and connectivity. The aims to select only important edges while preserving the original nodes, thus reducing the search space. Edges are sampled based on specific edge scores, for example, the Jaccard similarity score (Satuluri, Parthasarathy, and Ruan 2011), the number of triangles (Hamann et al. 2016), wedges (Comandur Seshadhri, Pinar, and Kolda 2013), or paths (Jha, Seshadhri, and Pinar 2015). Research has demonstrated that sampling and storing $O(nlogn)$ weighted edges is sufficient to preserve the original graph's important topological structure (Spielman and Srivastava 2008). Most generally, in the edge sampling method,

  - A probability $p_{i,j}$ is assigned to each edge $(i, j) \in G$,
  - Each edge is selected according to its probability $p_{i,j}$
  - When edge (i, j) is chosen to be in the graph, multiply its weight by $1/p_{i,j}$.

  This procedure guarantees that:

$$E[L_H] = L_G$$

$$p_{i,j} = min(1, \frac{\gamma}{min(d_i, d_j)}), \text{ for all edges } (i, j)$$

  γ : Controls the number of edges to find in the graph. In this approach, the key step is to determine the sampling probability for each edge. There is a tension between

- choosing small $p_{i,j}$ generates a sparser H,
- choosing larger $p_{i,j}$ approximates G more accurately.

Edge sampling methods have three main problems in real applications.

- They have to modify edge weights to approximate the total weight of the sparsified graph to the original graph.
- The number of edges removed before sparsification is unknown, as the number of edges removed can vary even under the same parameter(s).
- When considering a random selection method, the importance of the selected edges is not taken into account.

- **Color-Coding Method and Color-Based Sampling**: Let H be a graphlet with k vertices, and $G = (V, E)$ be a graph. In this method, a random color c ($c \geq k$) is assigned independently and uniformly to each vertex of G from the color set $[c] = \{1, , c\}$. An embedding of H in G is said to be ***colorful*** if a distinct color colors each vertex. The probability that all its vertices are assigned unique colors is $p = c!/c^c$. If the endpoints of an edge have the same color, that edge is called a ***monochromatic*** edge. This method is known as the color-coding technique. A subgraph is constructed from these monochromatic edges. Initially, this method is proposed by (Pagh and Tsourakakis 2012) and counts the number of triangles (3-cliques) in this subgraph using either an exact or approximate triangle counting method. The resulting estimation value is scaled by multiplying p^{-2} for the total number of triangles in the original. The focus is to create a subgraph that preserves the original graph's structural attribute based on the connectivity of vertices according to their color. Thus, large graphs can be analyzed computationally more efficiently using smaller samples representing them. This can also highlight the more meaningful structural patterns or cliques by eliminating less significant edges in the graph.

- **Rejection Sampling**: This method is a Monte Carlo-based (Mackay 1998) technique that generates samples from a target probability distribution function when direct sampling is impractical or infeasible. Firstly, a simpler distribution, which is relatively easy to sample from and covers the support of the target distribution, also known as the proposal distribution, is chosen. The samples are generated from this

distribution, and then a decision is made whether to accept or reject them based on their adherence to the characteristics of the target distribution. At each proposed sample point, the ratio of the probability density function of the target distribution is calculated to that of the proposal distribution. Accordingly, it is decided whether the sample is accepted or rejected. The purpose of this selective approach is to ensure that the samples generated are consistent with the properties of the target distribution. The effectiveness of rejection sampling depends on choosing an appropriate proposal distribution. Overly simplistic or complex choices can lead to inefficiency.

3.4. Graphlet Counting Algorithms

This section overviews the methods for the exact or approximate graphlet counting problem. While exact algorithms try to count all occurrences of induced graphlet, which is computationally challenging for large datasets, approximate algorithms try to give an approximate value close to the exact value, which uses sampling, probabilistic, or heuristic approaches as explained in Section 3.3.3 to reduce the computational cost.

3.4.1. Exact Graphlet Counting Algorithms

Exact algorithms for counting cliques fall into two main categories: those that enumerate cliques, where each is explicitly identified in the graph, and those that count cliques without explicitly listing each as explained in Section 3.3.3.1. Identifying each clique is computationally intensive due to the combinatorial explosion. Counting-based algorithms determine the total number of cliques using combinatorial methods. It is optional to list all cliques, especially when there is a need for more efficient algorithms. This section outlines and compares the exact graphlet counting algorithms in the literature.

3.4.1.1 RAGE

RAGE algorithm (Marcus and Shavitt 2012) counts non-induced, connected subgraphs and orbits of sizes 3 and 4. This algorithm is based on merging the neighborhoods of the vertex pairs to ensure that a given quartet of vertices has the desired edges to create a particular subgraph. RAGE is the first practical method to use combinatorial methods

for subgraph counting.

3.4.1.2 ORCA

ORCA algorithm (Hočevar and Demšar 2014) is an analytic approach based on matrix multiplication for graphlet counting. It is based on orbits and not directly graphlets. It counts the orbits in graphlets up to 5 vertices and counts specifically induced, connected subgraphs. The algorithm builds a system of linear equations connecting the orbit counts from graphlets of up to five nodes; this allows the calculation of all orbit counts by enumerating only one. It enumerates (k-1)-node graphlets and a single k-node graphlet to count the k-node graphlets orbits.

3.4.1.3 PGD

PGD algorithm (Ahmed et al. 2015) is a parallel algorithm that counts undirected subgraphs up to 4 nodes. It is one of the fastest algorithms in the literature. It is based on the classical triangle counting algorithm to count with several primitives, which are then used to obtain the frequency of each subgraph. It provides a new theoretical analysis of combinatorial arguments explaining the relationships between graphs. It enumerates a few graphlets for each edge and obtains the count of others in constant time using combinatorial arguments. Thus, it achieves a significant improvement in the scalability of graphlet counting.

3.4.1.4 E-CLOG

E-CLOG (Dave, Ahmed, and Al Hasan 2017) is a parallel local subgraph counting algorithm that counts all 3, 4, and 5-size local graphlets, taking into account all possible edge orbits of a graph. This algorithm only enumerates only a subset of local graphlets. Using these graphlet counts obtained by enumeration and combinatorial methods, the remaining graphlet counts are obtained in constant time.

3.4.1.5 ESCAPE

The ESCAPE algorithm (Pinar, Seshadhri, and Vishal 2017) is a single thread algorithm based on a divide and conquer approach that counts the undirected subgraphs up to 5 nodes and defines the substructures of each counting subgraph to partition into

smaller patterns. This is a very general method, and it is possible to explain several formulas to calculate the frequency of each subgraph when the correct decomposition choices are made.

3.4.1.6 EVOKE

EVOKE algorithm (Pashanasangi and Seshadhri 2020) counts the vertex orbits, that is, for every vertex v of the input graph, counts the graphlets involving v. The difference between EVOKE and E-Clog algorithms is that Evoke is a vertex-centric local subgraph counting algorithm. At the same time, E-Clog is an edge-centric local subgraph counting algorithm. EVOKE algorithm counts vertex orbits for all 5-vertex graphlets. This algorithm is built on the ESCAPE algorithm and divides graphlets like it into smaller graphlets. Rather than enumerating all the graphlets, it simply enumerates a few small graphlets and obtains the frequency of much larger graphlets with combinatorial methods.

Table 3.3. Categorization of exact graphlet counting algorithms.

Algorithm	**Local/ global counting**	**tested k**	**Input type**	**Output type**	**Algorithmic Design**
RAGE (Marcus and Shavitt 2012)	both	≤4	directed	both	sequential
ORCA (Hočevar and Demšar 2014)	both	≤5	undirected	both	single threaded
PGD (Ahmed et al. 2015)	global	≤4	undirected	induced	parallel
E-CLOG (Dave, Ahmed, and Al Hasan 2017)	local	≤5	undirected	induced	both
ESCAPE (Pinar, Seshadhri, and Vishal 2017)	global	≤5	undirected	both	sequential
EVOKE (Pashanasangi and Seshadhri 2020)	local	≤5	undirected	both	both

Table 3.3 categorizes exact graphlet counting algorithms. They are classified based on whether they perform graphlet (global counting) or orbit (local counting) counting, with some algorithms capable of both. The variable k denotes the maximum number of vertices in a graphlet considered in their respective research papers. These algorithms can handle input data in the form of either directed or undirected graphlets. While induced graphlets provide significantly more information than non-induced graphlets, counting them is considerably more challenging. Consequently, some algorithms count non-induced graphlets and convert these counts to obtain induced graphlet counts, whereas

others directly count induced graphlets. To enhance scalability and improve runtime, many studies have developed parallel algorithms.

RAGE and ORCA algorithms count graphlets and orbits. The difference is that RAGE works with up to 4 nodes, while ORCA works with graphs with up to 5 nodes. Another difference is that instead of directly counting the induced subgraphs, RAGE reconstructs them from the noninduced subgraph counts. In addition, studies have shown that ORCA's performance is better than RAGE's.

PGD and ESCAPE are the fastest algorithms that count graphlets exactly, but PGD counts graphlets up to 4 nodes while ESCAPE counts up to 5. PGD algorithm outperforms RAGE and ORCA algorithms. Also, RAGE and ORCA work on connected subgraphs, while PGD and ESCAPE algorithms work on connected and unconnected graphlets. Since the ESCAPE algorithm is serial, its comparison with PGD is based on the execution of PGD on a single thread. ESCAPE has been observed to be much more scalable than PGD as the dataset size increases.

E-CLOG and EVOKE algorithms do local subgraph counting. While E-CLOG counts edge orbit, EVOKE counts vertex orbit. EVOKE algorithm provides both serial and parallel implementations. However, EVOKE and ORCA are compared with the serial implementation of EVOKE to get a fair comparison with ORCA. EVOKE algorithm has shown that it is more scalable and performs better than ORCA.

3.4.2. Approximate Graphlet Counting Algorithms

While exact approaches are computationally challenging for large, dense datasets, approximate methods have become a practical alternative. These techniques try to avoid exhaustive enumeration estimates; instead, they use sampling and statistical methods, explained in Section 3.3.3.2. A representative subset of nodes and edges are selected, so essential graph properties are preserved while reducing computational overhead. Choosing an appropriate sample size is critical, as it directly impacts both the accuracy of results and the algorithm's efficiency. Such approaches enable the analysis of large-scale datasets, which are otherwise impractical to handle with exact algorithms. This section reviews key approximate graphlet counting methods in the literature.

3.4.2.1 Doulion

The Doulion algorithm (Tsourakakis et al. 2009) offers a parallel approximate triangle counting method within the MapReduce framework (Dean and Ghemawat 2004). The algorithm is not considered as powerful as other triangle counting algorithms but rather viewed as a complementary tool. This algorithm can be utilized as a preprocessing step when dealing with graph data that either fits or does not fit in memory. In this approach,

- A coin is tossed for each edge; if the probability is p, this edge is preserved in the sparsified graph; if 1-p, it is eliminated. Since each edge has a probability p, and a triangle consists of three edges, the probability that a triangle will be preserved in the sparsified graph is p^3.
- The weight of each protected edge is re-weighted by multiplying it by 1/p. (The initial graph is unweighted.)
- Count each triangle in the sparse graph as the product of the weights of the edges that constituent the triangle.

It is stated that any triangle counting algorithm can be applied after the coin toss. However, some situations need to be addressed. If the original graph is too large, the sparse graph may still need to fit in memory. In this case, it states that stream or semi-stream algorithms should be preferred. It is also stated that the random sampling method, like the reject method (Vitter 1984), can be used in cases where the dataset significantly exceeds the memory. This method is based on sequentially selecting random records from a file on a hard disk. According to the results of the experiments, it was observed that the accuracy was more than %99 for the probability varying between 0.1 and 0.9. A simple integer algorithm using DOULION as the first step has been observed to speed up about 130 times compared to itself.

3.4.2.2 Approximating the Number of Network Motifs

This algorithm (Gonen and Shavitt 2009) approximates, for each vertex, the number of graphlets that the vertex participates in. The approximate amount of that graphlet in the entire graph is obtained by summing the information obtained from each edge. In this

study, an algorithm is presented for each graphlet, and some of these algorithms are based on the color-coding technique. According to the color-coding technique, each node is colored with one of the k-specified colors, and if all nodes of a graphlet consist of different colors, it is called colorful.

To find the k-cycle count for each edge (u,v), in this method, the number of colorful paths of length k-1 between u and v are counted. Since the nodes u and v are neighbors, the k-cycle is obtained with nodes u and v and k-1 colorful path.

To find the count of k-cycle with a chord (Figure 2.8e is an example of 4-cycle (Figure 2.8d) with a chord), let assume a chord edge (u,v), and l be the distance between the endpoints of the chord u, v on the cycle. On the other hand, the path between u and v to complete the circle is $k - l$. Like the cycle algorithm, the final result is obtained by calculating the number of these two colored paths that make up the cycle.

Tailed triangles are calculated over that node (let us say u) connecting the triangle and the edge. So, to find the triangle that is formed by node u, it uses the approximate k-cycle algorithm for $k = 3$. Then, with this triangle information, the neighbors of node u are traversed, and the tailed triangles are counted.

Also, this work presents an exact 4-clique (Figure 2.8f) algorithm that counts 4-cliques for each edge. It checks for each edge (u,v) and whether the neighbors of u and v form a triangle. It detects cliques by examining whether the third nodes other than u and v, which form a triangle, are adjacent. 4-clique consists of 6 edges, so the total value is divided by 6.

3.4.2.3 GUISE

This publication (Bhuiyan et al. 2012) proposes a method using a uniform sampling method to generate an approximate GFD (graphlet frequency distribution) of an extensive network. The Graphlet Frequency Distribution (GFD) is a vector of a graph that characterizes the relative frequencies of the various graphlets in the graph (Bhuiyan et al. 2012). GFD can be viewed as a fingerprint created from local topological templates to analyze large networks. In this method, the GFD vector is generated for graphlets up to five nodes due to the complexity of the problem, and Markov Chain Monte Carlo (MCMC) sampling is used as the sampling method. The frequencies of the graphlets are needed to generate the GFD. The algorithm uses the relative frequencies of the graphlets to construct

GFD using the sampling strategy rather than the exact frequencies of each graphlet.

Suppose a set S contains all k-graphlets (3 ≤k≤ 5) (see figure 2.10). $|S| = \sum_1^{29} f(i)$ ($f(i)$ denotes the count of graphlet i). The task is to sample one of these k-graphlets embeddings from S uniformly at random. The probability of selecting each graphlet is $1/|S|$. However, since exact graphlet counting is not practical, this method uniformly samples a graphlet without enumerating all graphlet embeddings. The problem's characteristics are similar to those solved by Monte Carlo Markov Chain (MCMC) algorithms.

MCMC algorithms perform a random walk, and a distribution is sampled in small steps over the sample space, such that the $stationary$ distribution of the random walk is aligned with the iid probability distribution. (*Independent and identically distributed random variables*: if each random variable has the same probability distribution as the others and all are mutually independent, we can say they are independent and identically distributed.)

In this method, GUISE selects a particular graphlet from S, then walks to one of the neighboring states with the probability defined by an appropriate state transition probability matrix. The probability of transition between two non-adjacent graphlets is zero. This algorithm constructs a symmetric transition probability matrix T to achieve stationary distribution. This method considers each graphlet as a node. It assumes an edge between them if it is adjacent to another graphlet. The random walk can be viewed along the graph's edges. It assumes that the degree of a graphlet is equal to the sum of its neighbors in the random walk. When calculating the probability of transitioning from one graphlet to another, the algorithm examines the degrees of two adjacent graphlets (x, y) and defines $T(x, y) = T(y, x) = \min(1/\deg(x), 1/\deg(y))$.

The first step of this method is to select a particular graphlet (g_x) from S. Then, all potential neighbor graphs of this graph are generated. A graphlet g_y is chosen from this graphlet (g_x) neighbor list with uniform distribution (probability is $1/|d_{g_x}|$) and finds its neighbors. Then acceptance probability is calculated with $min(|d_{g_x}|/|d_{g_y}|, 1)$. if the move is accepted, the current graphic g_x is replaced by g_y. Also, the frequencies of g_x are incremented by 1. These steps are repeated for a user-defined value. A comparison is not presented, as no other studies sampled graphlets from a large graph and generated a GFD.

3.4.2.4 Colorful Triangle Counting

This algorithm (Pagh and Tsourakakis 2012) is a parallel randomized algorithm that sparsifies the graph and sample triples based on vertex degree partitioning. The basic idea of the algorithm is that if two edges of a triangle are sampled, the third edge is always sampled. Let an edge $e = (u, v)$; if the colors of nodes u and v are the same, this edge is called a $monochromatic$ edge. This method selects each monochromatic edge with probability N=1/p (n number of colors) and processes a triangle counting algorithm among these selected edges. Since the algorithm assumes that when the two edges forming the triangle are sampled, the third one will be sampled, the probability of choosing two edges is p^2, so the probability of choosing a triangle is p^2. Therefore, the number of triangles multiplied by $1/p^2$. This method is implemented in the MapReduce framework (Dean and Ghemawat 2004).

3.4.2.5 Sahad

This paper (Zhao et al. 2012) proposes an algorithm for relational subgraph analysis using the MapReduce framework where the subgraph is in the form of a tree. It is a randomized approximation algorithm based on the color coding method for subgraph counting. It counts graphlets (tree-formed ones) up to 12 nodes. According to the color coding method, graph G is colored with k colors ($k >$ the number of vertices of the searched graphlets), and then colorful embeddings (where each node has a different color) of the searched graphlet are counted. In this method, the graphlet is divided into sub-parts and a dynamic program is used to count the frequency of these sub-parts. In this method, each sub-part is counted only once, so it is well suited for computing graphlets where common subparts such as single nodes, edges, or simple paths are shared since it reduces the computational cost. However, in this study, disk I/O is observed to be a bottleneck for scalability.

3.4.2.6 GRAFT

GRAFT (Rahman, Bhuiyan, and Al Hasan 2014) counts the frequencies of graphs up to five nodes to generate the GFD vector. This method processes a random subset of edges iteratively. Checks if each edge is part of a graphlet. The total frequency of a graphlet is calculated by summing the partial counts obtained from each edge. A specific

edge e_g of the graphlet g is selected and aligned with an edge e from the sampled edges, and all the embeddings of a graphlet are enumerated based on edge e. Two different way to enumerate a graphlet is proposed according to graphlet type (tree (i.e. g_1 on Figure 2.10) or cyclic (i.e. g_{22} on Figure 2.10) graphlet). The choice of this specific edge is not random. If the graphlet type is a tree graphlet, the edge with the largest edge orbit should be selected.

The GRAFT selects a subset of the edges from an iid distribution, but the edges adjacent to nodes with higher degrees are much more part of the graph. This results in high variance and low sampling accuracy. GRAFT adopts the edge sampling algorithm. It divides the edges into homogeneous sub-population of edges corresponding to a set of edges that have similar partial counts for a given graphlet. Finally, a set of sampled edges is selected by the p-fraction of the edges from each of the subpopulations.

3.4.2.7 Wedge Sampling

This algorithm (Seshadhri, Pinar, and Kolda 2014) proposes a sampling method that samples wedge patterns from a uniform distribution. Consider a vertex v and a wedge centered with this vertex v. The number of wedges can be calculated with this formula: $W_v = \binom{d_v}{2}$. So, total wedge count is calcuted with $W = \sum_v \binom{d_v}{2}$. This algorithm assigns a probability p to each vertex ($p = W_v/W$). It selects the vertex v with probability p_v and takes the uniform random neighbor pair of v. Thus assuming it produces a uniform random wedge.

3.4.2.8 Path Sampling

This algorithm (Jha, Seshadhri, and Pinar 2015) proposes a sampling algorithm that approximates the frequencies of all 4-vertex pattern subgraphs (see Figure 2.8. It is a randomized algorithm based on 3-path sampling and makes no distributional assumption.

The algorithm begins by sampling the 3-path patterns uniformly. For each edge, it computes τ_e (an edge $e(u, v)$, $\tau_e = (d_u-)(d_v - 1)$ and $W = \sum_e \tau_e$). Then a probability $p_e = \tau_e/W$ is assigned to each edge. An edge is selected according to its probability. Then, a 3-path graphlet is sampled by choosing uniformly random neighbors among the nodes of the selected edge. This step is repeated k times to create a set S. 4-node graphlets are searched on this set.

It has been observed that this method does not perform well on 4-cycle containing graphlets, because it requires too much sample to take. So, a better algorithm called as centered 3-path is proposed to estimate them. According to this algorithm, nodes are sorted according to their degrees and ids. Let assume an edge $e = (u, v)$, $L_{u,v}$ denotes the number of neighbors of u greater than v and $L_{v,u}$ denotes the number of neighbors of v greater than u. For each edge, it computes λ_e as $L_{u,v}.L_{v,u}$. Let $\Lambda_e = \sum_e \lambda_e$. Then a probability $p_e = \lambda_e/\Lambda_e$ is assigned each edge. And similar steps as above are followed.

The results of the method presented in this study are compared with GUISE (Bhuiyan et al. 2012), GRAFT (Rahman, Bhuiyan, and Al Hasan 2014) and some color coding methods (Hormozdiari et al. 2007), (Betzler et al. 2011), (Zhao et al. 2012). This algorithm shows that it predicts graphlets within %1 relative error and is much faster than counting and other sampling-based algorithms.

3.4.2.9 GSparsify

This algorithm (Zhao 2015) provides a graph sparsification algorithm that preserves a small subset of edges from a graph that is more likely to be in clusters and eliminates others with little or no structure correlation to clusters. It states that densely knitted edges in a cluster often form frequently small-sized graphlets. Based on this approach, a sparsification algorithm has been developed to preserve the clustering significant edges without sacrificing the clustering quality. It decides the importance of an edge for clustering by examining the short-path cycle patterns of which that edge is a part. Specifically, edges within the cluster participate in the cycle pattern more than edges between clusters. In addition, two other methods of counting short-length cycles are proposed to calculate graph motif-based cluster importance.

According to the proposed method, let an edge (u, v) constituent the cycle pattern. This pattern is divided into three sub-parts: An edge (u, v), a path from u to w, and another path from v to w. These two paths are enumerated to get the cycle count formed by these paths and edge (u, v).

Gsparsify first calculates the cluster significance score based on a series of cycle motifs for each edge. It then ranks these edges according to that score. gSparsify first calculates the cluster importance score for each edge. It then ranks these edges according to that score. This algorithm implements a node-centered sparsification method. The top

d_u^{γ} (user-specified threshold value $0 \leq \gamma < 1$) edge of each node is preserved while the others are eliminated (d_u is the degree of node u).

3.4.2.10 4-Prof-Dist

Many algorithms have been introduced in the literature that try to take advantage of the power of parallel architectures to provide scalable approaches that can reduce the required computation time. One of them is the distributed estimation of graph 4-profiles (4-Prof-Dist) (Elenberg et al. 2016). It is a novel distributed algorithm for counting all 4-node-induced subgraphs. This algorithm also estimates the local 4-profiles centered at each vertex of the graph. The algorithm has been implemented using the GraphLab framework (Low et al. 2014) based on a message-passing scheme between nodes and their neighbors. There are 11 possible graphlets with 4-nodes (including unconnected ones). It creates a vector (a profile) with 11 elements for each vertex depending on whether it is part of 11 different 4-node graphlets. That means this algorithm thinks of each node as an embedding in an 11-dimensional space that characterizes the local geometry of its neighbor. The algorithm estimates local 4-node graphlets using the profiles obtained.

3.4.2.11 Parallel Five-Cycle Counting Algorithms

This algorithm (Shi, Huang, and Shun 2022) proposes two exact and two approximate parallel algorithms to count five cycle patterns. Two exact algorithms are parallel implementations of Kowalik's algorithm (Kowalik 2003) and the ESCAPE algorithm (Pinar, Seshadhri, and Vishal 2017). Finding the five-cycle pattern requires finding all directed two paths and three paths.

In the parallel version of Kowalik's algorithm, all nodes are sorted and processed in non-increasing degree order. Then, the graph is oriented by arboricity (degeneracy) orientation. All five-cycles are counted once and only over the highest-ranked node. The difference between the parallel ESCAPE algorithm and serial ESCAPE is that the serial algorithm orients the graph according to degree orientation. However, in the parallel version, the graph is oriented according to arboricity orientation.

This chapter also presents two different approximation algorithms based on edge sparsification and color coding techniques. In the edge sparsification algorithm, each edge is preserved uniformly at random with probability p, and then two exact parallel

algorithms are proposed to run over the sparsified graph. The color coding-based method colors the vertices of graphs with c colors and preserves monochromatic edges. It runs two exact parallel algorithms over the sparsified graph.

Table 3.4. Categorization of approximate graphlet counting algorithms.

Algorithm	**tested k**	**Graphlet type**	**Algoritmic Design**	**Approx. method**
Doulion (Tsourakakis et al. 2009)	3	triangle	parallel	edge sampling
Approx. network motifs (Gonen and Shavitt 2009)	O(log n)*	k-cycle	sequential	color coding
Guise (Bhuiyan et al. 2012)	≤ 5	all	sequential	uniform sampling
Colorful triangle (Pagh and Tsourakakis 2012)	3	triangle	parallel	color coding
Sahad (Zhao et al. 2012)	≤12	k-graphlets in tree form	parallel	color coding
GRAFT (Rahman, Bhuiyan, and Al Hasan 2014)	≤5	all	parallel	edge sampling
Wedge sampling (Seshadhri, Pinar, and Kolda 2014)	3	wedges	sequential	wedge sampling
Path sampling (Jha, Seshadhri, and Pinar 2015)	4	k-graphlets	sequential	path sampling
gSparsify (Zhao 2015)	k ≤4	4-cycle	sequential	edge sampling
4-Prof-Dist (Elenberg et al. 2016)	k ≤ 4	all	parallel	edge sampling
Parallel 5-cycle (Shi, Huang, and Shun 2022)	5	cycle	sequential / parallel	edge sampling color coding

Table 3.4 categorizes approximate graphlet counting algorithms based on several criteria: the size of the graphlets, the types of graphlets they count, whether the algorithms are designed sequentially or in parallel, and the approximation methods used. In summary, this chapter details the proposed methods and techniques for identifying and examining small subgraph patterns within extensive networks. This understanding is essential for advancing k-clique counting algorithms and forms a robust basis for the enhancements our method introduces. The approaches discussed align with those utilized in existing literature. As we progress, this foundation will enable a deeper comprehension of algorithmic advancements and their practical applications in network research, as well as their future potential.

CHAPTER 4

CLIQUE COUNTING

This section first explains the problem, emphasizes the challenges of the clique counting problem, and presents the practical applications of cliques in different domains. Then, a comprehensive review of clique counting algorithms in literature is presented. In light of the challenges and the importance of cliques, in the last section, we explain the motivation of our work. Our proposed algorithm uses these extremal graph theorems, and the novelty of our approach lies in their application to clique counting.

4.1. Problem Definition and Challenges

K-clique refers to a fully connected graph with k nodes; every pair of nodes is connected with an edge. The clique counting algorithm aims to obtain the number of k-cliques per vertex/edge or in the entire dataset. The k-clique counts can be obtained by exact or approximate methods similar to strategies explained in Section 3.3.3.

Counting k-cliques leads to combinatorial explosion as k grows; the number of potential cliques grows exponentially, making the search space unmanageable for large graphs. This problem has high computational complexity; exact algorithms typically have exponential time complexity. As the size of the dataset increases, the number of possible subgraphs needed to check for the presence of cliques also increases, making the computational burden even heavier.

4.2. Applications of Clique Counts Across Domains

Cliques in social or communication networks play an important role in detecting communities, spams, or groups of individuals with strong connections (Holland and Leinhardt 1977; Jackson, Rodriguez-Barraquer, and Tan 2012; Cleemput 2012; Yildiz and Kruegel 2012; Nedioui et al. 2020; Shi et al. 2021; Gregori, Lenzini, and Mainardi 2012; Jayanthi 2012; Foucault Welles, Van Devender, and Contractor 2010; Son et al. 2012). Similarly, in biological networks, cliques are used to analyze functional modules, iden-

tify protein complexes and reveal critical interactions that help to understand biological processes (X.-L. Li et al. 2005; H. Yu et al. 2006; Chen et al. 2013; Milik, Szalma, and Olszewski 2003; Betzler et al. 2011; Pržulj, Corneil, and Jurisica 2004; Saha et al. 2010). This also sheds light on biological pathways and disease mechanisms such as epilepsy prediction (Prokopyev et al. 2007; Iasemidis et al. 2003).

In recommender systems, cliques improve the accuracy of recommendations by identifying closely connected groups of users or items with similar preferences (Vilakone, Xinchang, and Park 2019; Vilakone et al. 2018; Manoharan et al. 2020; Gong et al. 2019; Tsourakakis et al. 2011). In fraud detection applications, cliques are useful for revealing hidden groups of actors engaged in fraudulent activities (J. Yu et al. 2023). In addition, clique counting has several applications in areas such as graph compression (Buehrer and Chellapilla 2008) and clustering (Duan et al. 2012).

4.3. Clique Counting Algorithms

This section introduces two fundamental algorithms: the Bron-Kerbosch algorithm (Bron and Kerbosch 1973), the foundation for maximal clique listing, and the ARBO algorithm (Chiba and Nishizeki 1985), designed explicitly for k-clique listing. Many subsequent algorithms either build upon these base algorithms or incorporate enhancements inspired by them. Thus, it is essential to present these foundational algorithms first before delving into the details of others.

The methodologies for clique identification and enumeration are categorized based on their level of precision: Exact or approximate algorithms. Exact techniques meticulously identify all cliques or count their existence within a graph, whereas approximate counting methods provide estimations that are especially useful for analyzing large, intricate graphs. The research objectives and the graph's scale determine the methodology choice. Exact techniques are well-suited for smaller or medium-sized graphs, while approximate counting is advantageous for exploring larger network structures. Within the scope of this study, the investigated approximate approaches rely on different sampling strategies, including random sampling, rejection sampling, and color-based sampling (explained in Section 3.3.3.2). This section presents a review of exact and approximate clique counting algorithms.

4.3.1. Base Algorithms

This section introduces two basic algorithms: the Bron-Kerbosch algorithm (Bron and Kerbosch 1973), which is the basis of maximal clique listing, and the ARBO algorithm (Chiba and Nishizeki 1985), which is designed explicitly for k-clique listing. Many subsequent algorithms have either built on these basic algorithms or included improvements inspired by them. It is, therefore, essential to present these basic algorithms before going into the details of the others.

4.3.1.1 Algorithm 457 (Bron and Kerbosch 1973)

Algorithm 457 (Bron and Kerbosch 1973) presents a seminal method for identifying all maximal cliques within an undirected graph. This algorithm, commonly called the Bron-Kerbosch algorithm, employs a backtracking strategy to systematically traverse the graph's vertices and edges, rigorously enumerating all possible maximal cliques. The initial approach of the Bron-Kerbosch algorithm makes a recursive call for every clique, so this causes inefficiency, especially in graphs with many non-maximal cliques. To improve efficiency, a strategy is provided that involves strategically selecting a pivot vertex from the graph. The vertices with higher degrees are prioritized for this selection. This strategy eliminates the redundant checks by focusing the search on the pivot's neighboring vertices. Any maximal cliques among the pivot's neighbors would also be found when testing the pivot itself or its non-neighboring vertices. The vertices adjacent to the current clique are incrementally added to explore the maximum possible expansion of the clique until no more vertices can be appended. All possible maximal cliques are explored by traversing exhaustively and avoiding redundant paths. The presented Algorithm 3 (Bron and Kerbosch 1973) finds all maximal cliques, including all R vertices, some P, and none in X. The time complexity of this algorithm is $O(3^{n/3})$, where n is the number of vertices.

4.3.1.2 ARBO (Chiba and Nishizeki 1985)

The article ARBO (Chiba and Nishizeki 1985) introduces graphlet counting algorithms for triangles, quadrangles, complete subgraphs, and cliques using the arboricity concept. It discusses efficient methods for computing the arboricity of a graph. The

Algorithm 3 BronKerboschWithPivoting(R, P, X) (Bron and Kerbosch 1973)

1: **if** $P = \{\}$ and $X = \{\}$ **then**
2: **report** R as a maximal clique
3: **end if**
4: **choose** a pivot vertex u in $P \cup X$
5: **for** each vertex v in $P \setminus N(u)$ **do**
6: BronKerboschWithPivoting($R \cup \{v\}, P \cap N(v), X \cap N(v)$)
7: $P \leftarrow P \setminus \{v\}$
8: $X \leftarrow X \cup \{v\}$
9: **end for**

algorithm ARBO operates by selecting a vertex v within the graph and scanning the edges of the subgraph induced by v's neighbors to identify pattern subgraphs containing v. Notably, this algorithm employs an iterative search for each vertex v in a non-increasing order of degree. Then, v is removed after processing to prevent duplication and provide computational efficiency. Thus, it also ensures a systematic and comprehensive enumeration of subgraphs. The ARBO has a time complexity of $O(km\alpha^{k-2})$, where α represents the arboricity of the graph, m is the number of edges, k is the size of the cliques being examined.

The algorithm 4 is the pseudocode of ARBO, which starts by sorting vertices by degree order. Then, constructs and induced subgraphs from the neighbors of each vertices. The algorithm recursively searches (k-1)-cliques in the neighborhood of the current vertex using the $ListCliques$ function. The processed vertices are removed from the graph at the end of each iteration to eliminate the duplicate discovery of cliques in subsequent steps.

These two algorithms, the Bron-Kerbosch and ARBO, can be seen as the foundation of other methods for clique enumeration. Therefore, their detailed analysis and pseudocodes are presented to make the other sections more understandable. Although both algorithms serve the same purpose, they differ in their innovative strategies, efficiency, and computational characteristics. Bron-Kerbosch uses a backtracking approach and explores all potential maximal cliques by traversing the graph's vertices and edges. It chooses a pivot vertex to maximize performance and avoid duplicate recursive calls by focusing the search on neighboring vertices of the pivot. On the other hand, using the arboricity concept, ARBO introduces algorithms for enumerating various types of subgraphs within a graph, including cliques. The algorithm ARBO operates by iterative scanning subgraphs induced by vertices in a non-increasing order of degree, efficiently identifying cliques containing each vertex. Due to the ARBO algorithm's complexity

Algorithm 4 The Algorithm ARBO (Chiba and Nishizeki 1985)

```
1:  function ARBO(G, k)
2:      Let Cliq denotes ∅.
3:      Let V denotes the list of degree ordered vertices in G.
4:      for each vertex v in V do
5:          NbrList_v ← getNeighbors(G, v)
6:          G_{N_v} ← getInducedSubgraph(G, NbrList_v)
7:          Cliq_v ← ListCliques(G_{NbrList_v}, k − 1, {v})
8:          Cliq ← Cliq ∪ Cliq_v
9:          DeleteVertex(G, v)
10:     end for
11:     return kCliques
12: end function

13: function ListCliques(G, l, C)
14:     if l = 2 then
15:         return {{u, v} ∪ C | (u, v) ∈ E(G)}
16:     end if
17:     lCliq ← ∅
18:     for u in V(G) do
19:         NbrList_u ← getNeighbors(G, u)
20:         G_{N_u} ← getInducedSubgraph(G, NbrList_u)
21:         lCliq_u ← ListCliques(G_{N_u}, l − 1, C ∪ {u})
22:         lCliq ← lCliq ∪ lCliq_u
23:         DeleteVertex(G, u)
24:     end for
25:     return lCliq
26: end function
```

and the graph's arboricity being closely related, it provides practical efficiency to graphs with low arboricity. Both algorithms form the basis of many algorithms in literature and offer critical contributions to clique counting problems with different computational advantages.

4.3.2. Exact Clique Counting Algorithms

Exact clique counting algorithms either enumerate cliques by identifying each clique within a graph or count only cliques present without identifying each. Enumeration-based algorithms explicitly list all cliques. Identifying each clique is computationally intensive due to the combinatorial explosion. Counting-based algorithms determine the total number of cliques using analytic approaches. It is optional to list all cliques, especially when there is a need for more efficient algorithms. The following subsections presents

clique counting algorithms based on either enumeration or counting based methodologies.

4.3.2.1 Akkoyunlu's Algorithm (Akkoyunlu 1973)

The algorithm proposed by Akkoyunlu (Akkoyunlu 1973) is equivalent to the Bron-Kerbosch algorithm, even though they are explained differently. Both algorithms build the same search tree structure and yield the same results. The algorithm proposed by Akkoyunlu efficiently finds maximal cliques in large graphs by decomposing the problem into smaller, non-overlapping sub-problems and managing them using a stack-based approach. It begins by splitting the problem into smaller, disjoint sub-problems to avoid generating duplicate or sub-maximal cliques. Subsequently, it employs a push-down stack to store partially solved sub-problems, minimizing memory usage by focusing on the current task. The algorithm iteratively divides each sub-problem into two disjoint parts — one including a selected element and one excluding it — before pushing them onto the stack for processing. This iterative refinement continues until specific criteria are met. At this point, the algorithm applies a particular method to determine the maximal clique associated with the subset. The algorithm systematically explores the graph structure through these steps to identify all maximal cliques efficiently.

4.3.2.2 MACE (Makino and Uno 2004)

The paper MACE (Makino and Uno 2004) comprehensively explores algorithms tailored for enumerating both maximal and bipartite cliques within graphs. For maximal cliques, it introduces two distinct strategies. The first approach utilizes matrix multiplication, capitalizing on the parent-child relationship inherent in maximal cliques to efficiently compute their children. This method constructs adjacency matrices to identify valid child cliques, resulting in a streamlined computation process that significantly improves efficiency, especially in denser graphs. The algorithm's complexity is $O(knm\alpha^{k-2})$, where n is the number of vertices, m is the number of edges, α is the arboricity of the graph, and k is the clique size. The second algorithm for maximal cliques leverages the maximum degree of the graph. It recognizes that in sparse graphs, each maximal clique (except the lexicographically largest one) can have at most $\triangle^2$ children, in which $\triangle$ represents the maximum degree. By avoiding the explicit construction of the complete set of candidate child indices and checking candidates in lexicographic order, this method reduces computation

time, particularly benefiting graphs with small maximum degrees.

4.3.2.3 Memory Efficient Clique Counting (Tomita, Tanaka, and Takahashi 2006)

The paper proposed by (Tomita, Tanaka, and Takahashi 2006) introduces a depth-first search algorithm for efficiently generating all maximal cliques in an undirected graph, leveraging pruning techniques reminiscent of the Bron-Kerbosch algorithm (Bron and Kerbosch 1973). Unlike Bron-Kerbosch, which directly enumerates maximal cliques, this new algorithm outputs them in a tree-like structure, conserving memory space. It operates by iteratively expanding a global variable Q, representing the current clique, from an empty starting point to larger cliques. At each step, the algorithm examines the intersection of neighborhoods of vertices in Q, determining if it forms a maximal clique. If not, it explores potential extensions by recursively considering induced subgraphs. During the search process, the algorithm maintains two lists called FINI (processed vertices) and CAND (remaining candidates). The Q is expanded only to the vertices in CAND. This minimizes the unnecessary exploration. Another strategy to reduce the number of vertices needing further exploration is to choose vertices from the neighborhood intersection. When a maximal clique is discovered, the algorithm prints a marker instead of the clique itself. The cliques can be reconstructed from this output. The complexity of algorithm is $O(3^{(n/3)})$.

4.3.2.4 DegenClique (Eppstein, Löffler, and Strash 2010)

The algorithm (Eppstein, Löffler, and Strash 2010) presents a variation of the Bron-Kerbosch algorithm (Bron and Kerbosch 1973). This algorithm orders the vertices according to degeneracy ordering. Then, the neighbors of each vertex are divided into two sets: P and X. P is the vertices that follow the current vertex in order of degeneration, while X is the set of vertices that precede it. Thus, the size of P is limited by the graph's degeneracy. This algorithm uses the Bron-Kerbosch algorithm with the parameters P, X, and current vertex. The pivot vertex is selected from the P and X sets during the recursive iteration. This strategy optimizes the complexity of the Bron-Kerbosch algorithm by reducing the number of recursive calls. The time complexity is $O(dn3^{d/3})$, where d is the degeneracy of a graph.

4.3.2.5 pbitMCE (Dasari, Ranjan, and Mohammad 2014)

The pbitMCE method (Dasari, Ranjan, and Mohammad 2014) employs a degeneracy ordering strategy similar to that presented by Eppstein et al. However, it uses a different strategy to represent the subgraphs. The algorithm presents a data structure called a partial bit adjacency matrix (pbam). This pbam comprises sets of bit vectors. It facilitates representing the necessary information for efficient vertex processing. This algorithm is implemented on the Hadoop framework. The algorithm starts by ordering vertices and determining the degeneracy d of the graph. Thus, each vertex has at most d neighbors appearing later in the ordering. Subsequently, each vertex's adjacency list is partitioned into pre and post-lists, containing vertices with lower and higher degeneracy orders. This pbam consists of sets of bit vectors, each corresponding to vertices in the pre and post-lists, encoding connections between the post-list and those in the candidate set P. The pbam is generated using a renumbering technique to assign unique identities to vertices in P and X. Following a similar structure to the algorithm proposed by (Eppstein, Löffler, and Strash 2010), pbitMCE counts maximal cliques by computing the sets P and X for each vertex v in the degeneracy ordering and using the algorithm introduced by (Tomita, Tanaka, and Takahashi 2006) for efficient exploration of the v-rooted search tree. Each clique is associated with the node with the lowest number. This facilitates the unique reporting of each maximal clique. The complexity is $O(kn3^{k/3})$, where the k-degree of a graph is defined as the minimum value such that every vertex v has at most $k$neighbors with a degree greater than or equal to the degree of v. It is hard to compute degeneracy ordering of vertices on a distributed environment. So, implementing pbitMCE on Hadoop is a challenging task. This task requires extensive inter-node communication. To reduce this complexity, the paper proposes to explore alternative vertex orderings, such as degree-based ordering in some scenarios.

4.3.2.6 kClist (Danisch, Balalau, and Sozio 2018)

The kClist (Danisch, Balalau, and Sozio 2018) algorithm improves the ARBO algorithm (Chiba and Nishizeki 1985) for listing all k-cliques. The degeneracy orientation is used, and a directed acyclic graph (DAG) is constructed to eliminate duplicate discovery of cliques. Besides, the algorithm utilizes parallelization techniques and special data structures to improve its performance for large-scale graph analysis. The time complexity

of kClist is $O(km\frac{d}{2}^{k-2})$, where d represents the degeneracy of the graph G, m is the number of edges, k is the size of the cliques being examined.

4.3.2.7 Clique Counting with Ordering Heuristics (R. Li et al. 2020)

A new heuristic for k-clique listing and counting algorithms, using a color ordering method derived from greedy graph coloring techniques (Hasenplaugh et al. 2014; L. Yuan et al. 2017) are proposed by (R. Li et al. 2020). A graph colorization technique using a greedy coloring algorithm is employed to graph, and distinct color values are assigned to adjacent nodes from 1 to the chromatic number x. The nodes are sorted in descending order based on this color number, and then a DAG is constructed. Thus, inefficient search paths are eliminated during the iterative enumeration process. The complexity of $O(km(\frac{\Delta}{2})^{k-2})$, where k is the clique size, Δ is the maximum degree, and m is the number of vertices.

4.3.2.8 ARB-COUNT (Shi, Dhulipala, and Shun 2021)

The paper (Shi, Dhulipala, and Shun 2021) introduces a series of parallel algorithms designed to address challenges in k-clique counting and densest subgraph detection. At its core, the ARB-COUNT algorithm enhances Chiba-Nishizeki's approach (Chiba and Nishizeki 1985) by leveraging low out-degree orientations of graphs, achieved through efficient parallel implementations of algorithms such as those by Goodrich-Pszona (Goodrich and Pszona 2011) and Barenboim-Elkin (Barenboim and Elkin 2008). This orientation reduces total work by peeling vertices parallel, leading to a poly-logarithmic span. ARB-COUNT exploits parallelism by recursively intersecting out-neighbors of vertices to build k-cliques efficiently. Utilizing parallel hash tables, filtering, and reduction operations, it achieves notable speed-ups, particularly for large graphs and values of k. The complexity of ARB-COUNT is $O(m\alpha^{k-2})$. Additionally, the paper presents ARB-PEEL and ARB-APPROX-PEEL algorithms for approximating k-clique densest subgraphs, which capitalize on parallel k-clique counting to peel vertices in parallel based on their k-clique counts iteratively. Colorful sparsification technique is employed to estimate k-cliques by drawing inspiration from earlier work on approximating triangle and butterfly (bi-clique) counts (Pagh and Tsourakakis 2012) (Sanei-Mehri, Sariyuce, and Tirthapura 2018). It leverages the proposed ARB-COUNT algorithm as a subroutine to achieve this approximation. The time complexity of the approximate algorithm is $O(pm\alpha^{k-2} + m)$, where m

is the number of edges $p = 1/c$, and c is the number of colors used.

4.3.2.9 Clique Counting with MapReduce (Finocchi, Finocchi, and Fusco 2015)

The paper proposed by (Finocchi, Finocchi, and Fusco 2015) presents two exact and approximate solutions for the issue of counting the number of k-cliques in large-scale graphs by focusing on theoretical and experimental aspects. It introduces parallel solutions using the degree orientation technique in the MapReduce framework. First, it provides an exact approach, then presents a sampling-based approach that significantly reduces the exact approach's computational demands. The second step identifies all the other nodes with a lower degree than those nodes and forms a triangle with them. In the third and final stage, the reduce phase, the algorithm finds clique patterns for each node using the information collected in the previous round. The paper explores approximate counting using two sampling strategy variants and the exact counting approach. The exact algorithm of this study is efficient for counting up to 7-node cliques in relatively small datasets; approximate counts are provided for larger datasets due to computational complexity. This algorithm requires $O(m^{k/2})$ computational effort, where m represents the number of edges in the graph, and k is the size of the examined cliques.

4.3.2.10 Pivoter (Jain and Seshadhri 2020)

The Pivoter is designed by the (Jain and Seshadhri 2020) to deal with the challenge of exact counting k-cliques in graphs, especially as the size of k increases. Pivoter utilizes pivoting to construct a Succinct Clique Tree (SCT), which provides a compressed representation of all cliques in the graph. SCT provides a strategy different from existing methods that explicitly enumerate every clique. Using SCT, Pivoter counts k-cliques cliques of any size without complete enumeration and reduces the recursion tree of backtracking algorithms. Thus, Pivoter states it overcomes scalability issues and achieves accurate clique counts in large graphs. Key contributions are the counting cliques for both globally and locally, for each vertices and edges, and the creation of SCTs through pivoting. The algorithm constructs the SCT by finding the graph's degeneracy orientation and then iteratively partitioning cliques based on pivot vertices. The SCT is constructed using recursive calls, avoiding redundant computations. Besides, a parallel version is also presented to enhance the algorithms' performance and scalability on large datasets. The

Pivoter has $O(n\alpha 3^{\alpha/3})$ time complexity, n represents the number of vertices and α the degeneracy of the graph (Ye et al. 2023). However, as stated in the research paper, even the parallel version of the algorithm has limitations on large graphs. Pivoter needs help to compute clique counts beyond $k = 10$.

4.3.2.11 GPU-accelerated Clique Counting (Almasri et al. 2022)

In this paper,(Almasri et al. 2022) integrates the graph orientation and pivoting (Jain and Seshadhri 2020) techniques to GPU accelerate existing algorithms for counting k-cliques in graphs. These algorithms are based on vertex-centric and edge-centric parallelization strategies, with binary coding and sub-warp partitioning methods that optimize memory usage and maximize parallel resources. One process that requires the most effort in clique counting algorithms is intersection operations. If we give an example of intersection operations on a triangle, which is the simplest of cliques (3-clique), to find the triangle formed by an edge, the intersection of the neighbors of the two nodes forming the edge is needed. This paper uses binary encoding to facilitate the intersection process and represents each vertex's induced subgraph with binary encoding. This strategy facilitates the intersection processes with bit-wise operations. On the parallelization side, sub-warp partitioning divides thread blocks into smaller groups, allowing tasks to be executed more efficiently on the GPU and helping to increase the level of parallelism. Additionally, it facilitates operations like list intersections. A hybrid version of degree and degeneracy orientation techniques is employed while orienting the graph. A comparison of vertex-centric and edge-centric parallelization strategies is provided regarding load balancing and weighing of parallelism granularity trade-offs. This paper also provides solutions for GPU memory constraints by using memory management techniques like binary encoding, pre-allocating memory for the largest potential-induced subgraph size, and substituting recursive tree traversal with an iterative method using a shared stack.

4.3.2.12 SDegree and BitCol (Z. Yuan et al. 2022)

The paper (Z. Yuan et al. 2022) presents two k-clique listing algorithms: SDegree and BitCol. These algorithms aim to accelerate the k-clique listing algorithms with merge-based set intersections and parallelism. For this purpose, the paper proposes two pre-processing techniques: Pre-Core and Pre-List. First, Pre-Core reduces the search

space by removing redundant vertices not contained in a k-clique. Then, Pre-List checks all connected components; if a connected component is a clique, it directly lists and removes cliques. The SDegree algorithm employs degree-based orientation and constructs a DAG. The novelty of this algorithm is to use the merge-join strategy while merging two vertex sets rather than the hash join, which is used by the other algorithms in the literature. If the vertex sets are ordered, the merge-join strategy efficiently merges these sets. The BitCol algorithm improves the SDegree algorithm by employing degeneracy and color-based ordering techniques and compressing the vertex sets using bitmaps. First, the input graph is converted to DAG using degeneracy orientation; then, the algorithm iteratively searches each vertex neighborhood. An induced subgraph is obtained from the current node's neighborhood, and the DAG of this subgraph using color-based orientation is constructed. This algorithm also uses advanced parallelization strategies for efficient k-clique listing. The time complexity of both algorithm is $O(km\left(\frac{\Delta}{2}\right)^{k-2})$. The space complexity of SDegree is $O(m + kN\Delta)$ and BitCol is $O(m + N\frac{\Delta^2}{L})$, where k is the clique size, N is the number of threads, Δ is the maximum out-degree, L is the size of nodes that each number can represent, and m is the number of edges.

4.3.2.13 EBBkC (Wang, Yu, and Long 2024)

The k-clique listing algorithms traditionally employ a vertex-based branching strategy, where larger cliques are constructed incrementally by adding a single vertex to an existing clique. A new algorithm EBBkC (Wang, Yu, and Long 2024) in the literature proposes an edge-based branching strategy. Instead of adding a single vertex, it tries to obtain larger cliques more efficiently by adding two nodes with edges between them, thus narrowing the search space. The algorithm incorporates three distinct edge-ordering methods to optimize the branching process. The first is truss-based edge ordering, which leverages truss decomposition to order edges to minimize the size of the resulting subgraphs, thus enhancing efficiency. The second is color-based edge ordering, which utilizes vertex coloring to prune branches, effectively reducing the number of candidate cliques and further improving performance. The third method is a hybrid approach, combining the strengths of truss-based and color-based ordering to provide theoretical and practical improvements. Besides, the paper introduces a method to terminate branches early if the subgraph is a dense structure like a clique or each vertex is connected to at least $k - 2$

other vertices within the subgraph, leveraging efficient combinatorial algorithms to list cliques in these cases to increase efficiency. This algorithm has the $O(md + k \cdot m \cdot \left(\frac{\tau}{2}\right)^{k-2})$ time complexity, where d is the degeneracy of the graph, τ is the maximum truss number of the graph (Wang, Yu, and Long 2024), k is the clique size, m is the number of edges. This algorithm presents better time complexity than the vertex-based branching algorithms (Chiba and Nishizeki 1985; Makino and Uno 2004; Danisch, Balalau, and Sozio 2018), which have $O(km\alpha^{k-2})$. The authors demonstrate that the τ is smaller than d, leading to better performance in edge-based branching. The space complexity is $O(m + n)$. This paper also incorporates parallelism techniques to further enhance the efficiency of the proposed algorithm and provides a comparison with the state-of-the-art k-clique listing algorithms depending on vertex-based branching strategy.

We review various enumeration-based k-clique algorithms, focusing on their methodology and complexity. Bron-Kerbosch (Bron and Kerbosch 1973) and Akkoyunlu (Akkoyunlu 1973) are similar algorithms because they construct similar search trees but are presented in different terms. They utilize recursive backtracking and stack-based approaches to identify maximal cliques. A matrix multiplication technique and the information of maximum degree in the graph are used for efficient clique enumeration in the MACE algorithm (Makino and Uno 2004). (Tomita, Tanaka, and Takahashi 2006) proposes an improvement based on a depth-first search approach to Bron-Kerbosch, and they use pruning techniques for optimal memory usage. (Eppstein, Löffler, and Strash 2010), and pbitMCE (Dasari, Ranjan, and Mohammad 2014) algorithms leverage degeneracy ordering and parallel processing for the performance in large-scale graph analysis based on Bron-Kerbosch algorithm. The kClist algorithm also (Danisch, Balalau, and Sozio 2018) provides similar contributions based on the ARBO algorithm. A heuristic method by (R. Li et al. 2020) uses color order to optimize the search process by pruning unproductive paths. The paper (Z. Yuan et al. 2022) presents two parallel k-clique listing algorithms, SDegree and BitCol, which use merge-based set intersections and preprocessing techniques to provide a time and space-efficient approach than the algorithms proposed by (R. Li et al. 2020). The EBBkC (Wang, Yu, and Long 2024) introduces a new edge-based branching strategy and edge-based ordering techniques to improve ARBO and presents better time complexity.

4.3.3. Approximate Clique Counting Algorithms

In this context, we investigate various approximate clique counting algorithms. This section outlines these algorithms.

4.3.3.1 Turán-shadow (Jain and C. Seshadhri 2017)

The Turán-shadow (Jain and C. Seshadhri 2017) algorithm is a randomized approach that aims to estimate the number of k-cliques (where $k \leq 10$) in a graph based on Turán's theorem (Turán 1941). Turán's theorem provides insights about a graph's maximum number of edges without having a $(k+1)$-clique according to the number of vertices. Formally, Turán's theorem states that in a graph $G = (V, E)$ with n vertices that do not contain a clique of size $k+1$ (where k is greater than zero), the number of edges is bounded by $\left(1 - \frac{1}{k}\right) \frac{n^2}{2}$. That means if the edge density of a graph exceeds $\left(1 - \frac{1}{k-1}\right)$, it must contain a k-clique. The algorithm starts by orienting the graph according to degeneracy ordering to reduce the search space and then continues creating the TuránShadow. It explores the neighborhoods of vertices iteratively to identify denser subgraphs. If the edge density of an out neighborhood exceeds the Turán density threshold for $(k-1)$-cliques, the induced subgraph is added to the TuránShadow. Otherwise, the process is applied recursively to find denser sets. The resulting TuránShadow comprises sets with densities above the Turán threshold, forming a collection of potential k-cliques. A sampling strategy is employed to randomly select subsets of vertices from these sets, which are then checked for clique formation. The complexity is $O(n\alpha^{k-1})$, where n vertices, α is the degeneracy and k is the clique size.

4.3.3.2 YACC (Jain and Tong 2022)

This YACC algorithm (Jain and Tong 2022) is an extension of the Turán-shadow algorithm (Jain and C. Seshadhri 2017) to improve the counting of large cliques in graphs. Algorithms Turán-shadow and Pivoter (Jain and Seshadhri 2020) previously proposed algorithm by the authors excel at counting small cliques, but they face challenges with larger cliques. YACC improves approximate clique counting by reducing the recursion tree's size and exploiting insights from real-world graph structures. YACC relaxes the stopping condition of Turán-shadow, which relies on a fixed density threshold from Turán's theorem.

Thus, YACC efficiently identifies dense subgraphs with a more adaptable stopping condition by significantly reducing the size of the recursion tree and the computation time. This improvement makes clique counting possible for challenging graphs like com-lj (Leskovec and Krevl 2014). The framework enhances control over the construction-sampling balance by introducing a parameter, μ, which affects the balance between computational complexity and accuracy of results. This parameter redefines what is needed for a graph to be considered dense. The algorithm divides the graph into two regions: dense and sparse. A sampling strategy is employed to estimate the clique counts in dense regions. The cliques in the sparse area counted exactly in a recursive manner. As a result, the efficiency and adaptability of Turán-shadow are enhanced by the YACC with some heuristics in practical applications with a computational complexity similar to TuránShadow.

4.3.3.3 ERS (Eden, Ron, and Seshadhri 2018)

(Eden et al. 2017) introduce a sublinear-time algorithm for triangle counting that defies the conventional linear-time norm for such computations. The paper uses degree, neighbor, and pair queries within the standard query model for sublinear algorithms on general graphs. Building on this significant advancement, an algorithm called ERS (Eden, Ron, and Seshadhri 2018) is introduced to extend its application beyond triangle counting. It aims to approximate the count of k-cliques within sublinear time, thus covering the previously established result for $k = 3$. The algorithm selects a sample set of vertices to estimate the number of k-cliques connected to this subset. A crucial aspect is to sample each k-clique connected to the sample set with almost equal probability. However, random sampling can compromise the selection of high-degree vertices likely to form cliques. The algorithm randomly samples high-degree vertices and tries to strike a careful balance between predicting cliques formed by high-degree vertices and cliques formed by low-degree vertices. A uniform edge (u, v) is sampled, and more vertices are added to that edge iteratively to attempt to form a k-clique. Depending on whether v is a low- or high-order vertex, the algorithm employs different sampling strategies, including uniform neighbor selection and rejection sampling. The complexity of algorithm $\tilde{O}\left(\frac{n}{C_k^{1/k}} + \frac{m^{k/2}}{C_k}\right)$, where n is the number of vertices, C_k is the number of k-cliques, and m is the number of edges.

4.3.3.4 SR-kCCE (Chang, Gamage, and Yu 2024)

The Turán-shadow, YACC (extended version of Turán-shadow), and DPColorPath (Ye et al. 2023), which will be explained in the color-based sampling section, are outstanding approximate k-clique counting algorithms based on a sampling strategy. A typical step of these algorithms is constructing a sampling space consisting of dense subgraphs containing k-cliques. These algorithms then sample a fixed element from the sampling space to estimate k-clique counts. However, this fixed sampling does not guarantee accuracy. The SR-kCCE (Chang, Gamage, and Yu 2024) algorithm presents a sampling-stopping strategy that guarantees accuracy while providing efficiency. Like the Turán-shadow, YACC, and DPColorPath algorithms, this algorithm consists of two steps: constructing the sampling space and sampling randomly from that space to estimate k-clique counts. The algorithms in the literature become inefficient when the sampling space construction time is high, especially for large datasets. On the other hand, if the sampling space is not refined from the non-cliques, the relative error worsens. The SR-kCCE algorithm constructs a balance between these two steps. This algorithm estimates the expected duration of the sampling phase. When this expected duration is approximately the same as the construction/refinement sampling space, called a shadow, the algorithm stops the refinement sampling space. Thus, it ensures that both phases are balanced regarding computational effort. This algorithm improves k-clique estimation compared to previous methods, especially for large datasets.

4.3.3.5 DPColorPath Clique Counting (Ye et al. 2023)

The paper DPColorPath proposed by (Ye et al. 2023) combines exact and approximate solutions to enable working with large and dense datasets. The graph is partitioned into dense and sparse regions. The algorithms that implement exact clique counting use the efficiency of the Pivoter algorithm for sparse areas and adapted sampling-based techniques for denser areas. Initially, a linear-time greedy coloring process is employed (Hasenplaugh et al. 2014; L. Yuan et al. 2017), establishing a non-decreasing node ordering based on color values and constructing a Directed Acyclic Graph (DAG) accordingly. Following the computation of a DAG, nodes are partitioned into sparse and dense regions based on the average degree of the neighborhood subgraph. The Pivoter algorithm accurately computes (k-1)-clique counts in the sparse areas. At the same time, dense regions are addressed

using three sampling-based methods: k-color set sampling, k-color path sampling, and k-triangle path sampling. These three algorithms utilize dynamic programming techniques and conduct uniform sampling. In the k-color set sampling method, k-color sets are selected, each consisting of k nodes with unique colors. The k-color path sampling samples from connected k-color sets are called k-color paths. It ensures the induced subgraph by the k nodes stays connected. The most effective of the trio, k-triangle path sampling, selects connected k-color sets where any three consecutive nodes form a triangle, known as k-triangle paths. The time complexity of k-color set sampling is $O(\chi^k)$, the time complexity of k-color path sampling is $O(\chi^{nk} + m)$, and the time complexity of k-triangle path sampling is $O(k\Delta)$ where χ is the number of colors of the graph G obtained by the greedy coloring algorithm (Hasenplaugh et al. 2014; L. Yuan et al. 2017), k is the clique size,$n$is the number of vertices, m is the number of edges and Δ is the number of triangles of the input graph.

The Turán-shadow algorithm (Jain and C. Seshadhri 2017) and its extension, YACC (Jain and Tong 2022), propose randomized sampling-based solutions based on Turán's theorem (Turán 1941) for the k-clique counting problem. While Turán-shadow is suitable for k values less than 10 and relatively small datasets, the YACC algorithm can handle k values up to 40 and has shown results for large datasets that were not previously reported in the literature. The algorithm ERS presents the sublinear-time solution (Eden, Ron, and Seshadhri 2018) defying traditional linear-time norms. The algorithms proposed by DPColorPath (Ye et al. 2023) combine exact and sampling-based techniques to handle large and dense graphs. The fixed number of samples used by algorithms like Turán-shadow, YACC, and DPColorPath impacts their accuracy; the SR-kCCE (Wang, Yu, and Long 2024) algorithm addresses this limitation by balancing the construction sampling space and sampling phases. The algorithm both provides efficiency for k-clique estimation and guarantees accuracy. Unlike these approximation algorithms, the BDAC algorithm provides a boundary instead of an estimation for the k-clique count without relying on any sampling strategy or recursive process.

4.3.4. Discussion on Clique Counting Algorithms

This section compares the algorithms according to results reported in related research papers. Algorithms are categorized according to the main features of their approach. Table 4.1 and Table 4.2 display this categorization, and each column entry provides detailed information. For better readability, the categorization is given in separate tables, which can be viewed side by side. The column *Approximate* indicates whether an algorithm uses an approximation technique, typically sampling strategies, and specifies the type of sampling strategy if available. Otherwise, there is no entry. A similar approach is employed to *Exact* column. An algorithm may provide both approximate and exact algorithms; both exact and approximate columns indicate the corresponding strategies. The *Parallelization* column indicates whether the algorithm supports parallel execution; if the corresponding entry for the algorithm is empty, it does not support parallelization. If an algorithm is based on one of the two base algorithms mentioned in this publication and provides suggestions for improving this algorithm, the *Base algorithm* column specifies this base algorithm. Most clique counting algorithms use an orientation technique such as degree, degeneracy, or color-based method as a pre-processing step to eliminate the duplicate exploration of cliques. The undirected input graph is converted to a directed acyclic graph (DAG) using one of these orientation techniques. The column *Orientation* indicates the orientation methods used by the algorithm; an empty entry indicates that no orientation techniques are used. The column *Objective* details the specific goal of each algorithm, indicating whether the algorithm is designed to enumerate maximal cliques or count k-cliques. Maximal and k-clique counting tasks overlap since a maximal clique can contain several smaller k-cliques. While counting maximal cliques, one can indirectly gather information about k-cliques. Many algorithms designed for k-clique counting have been inspired by maximal clique counting algorithms, particularly the Bron-Kerbosch algorithm. Several k-clique counting algorithms built upon innovations introduced by maximal clique counting techniques and improvements to Bron-Kerbosch (Bron and Kerbosch 1973) have been adapted for k-clique counting. For this reason, Table 4.2 also covers both maximal and k-clique counting algorithms, which are differentiated from the *Objective* column. The *Time complexity* column specifies the computational complexity of the algorithms, while the *Space complexity* column outlines the memory requirements if

such information is available; otherwise, the entry states "not reported." The explanations of complexity parameters are also included in the section detailing the algorithm.

The Bron-Kerbosch algorithm (Bron and Kerbosch 1973) and ARBO (Chiba and Nishizeki 1985) represent distinct approaches to the problem of enumerating cliques in a graph. Bron-Kerbosch uses a backtracking strategy, leveraging pivot vertices to thoroughly identify all maximal cliques. In contrast, ARBO relies on arboricity, scanning subgraphs induced by vertices to a decreasing degree to enumerate cliques efficiently. While the Bron-Kerbosch algorithm guarantees exhaustive coverage of all maximal cliques, the time complexity of ARBO is linked to the arboricity of the graph, making it particularly efficient for real-world graphs with low arboricity.

Akkoyunlu's algorithm (Akkoyunlu 1973), although described differently, essentially mirrors Bron-Kerbosch by generating an identical search tree (Johnston 1976).

The algorithms proposed by (Tomita, Tanaka, and Takahashi 2006), (Eppstein, Löffler, and Strash 2010), and (Dasari, Ranjan, and Mohammad 2014) (pbitMCE) each present unique approaches to finding all maximal cliques in an undirected graph, sharing a common heritage rooted in the Bron-Kerbosch algorithm (Bron and Kerbosch 1973). (Tomita, Tanaka, and Takahashi 2006)'s algorithm is notable for using depth-first search combined with effective pruning techniques. While these techniques are reminiscent of the Bron-Kerbosch method, (Tomita, Tanaka, and Takahashi 2006) algorithm structures the output in a memory-efficient tree-like format, unlike Bron-Kerbosch, which directly enumerates cliques without such organization.

If we compare the algorithms in terms of time and space complexity, the classic algorithms Bron-Kerbosch (Bron and Kerbosch 1973) and (Akkoyunlu 1973) have exponential time complexity due to the combinatorial nature of the problem and require linear space mainly to store recursion data. They are suitable for small to moderate-sized graphs where exact enumeration of maximal cliques is feasible. The algorithm proposed by (Tomita, Tanaka, and Takahashi 2006) has a similar complexity to Bron-Kerbosch. However, it has the advantage of slightly improved performance due to incorporating pivoting techniques. These algorithms are not suitable for large datasets due to exponential time complexity.

(Eppstein, Löffler, and Strash 2010) introduced a significant variation by incorporating degeneracy ordering, which optimizes vertex processing. This strategic ordering

ensures that each vertex is processed more locally and efficiently, considering its neighbors. Building on this, (Dasari, Ranjan, and Mohammad 2014) developed the pbitMCE algorithm, which also leverages degeneracy ordering but diverges in its approach by using a partial bit adjacency matrix (pbam) to handle subgraphs. This data structure enhances vertex processing efficiency in a distributed computing environment like Hadoop, highlighting pbitMCE's suitability for large-scale graph data. The time complexity of the algorithm (Eppstein, Löffler, and Strash 2010) is affected by degeneracy, so it is suitable for sparse graphs with low degeneracy. The pbitMCE algorithm employs pivoting and uses parallelism to enhance the efficacy of Bron-Kerbosch-style algorithms; however, the resulting complexity remains exponential in k.

The paper MACE (Makino and Uno 2004) proposes two strategies to list all maximal cliques but different than the Bron-Kerbosch, leveraging the matrix multiplication and maximum degree of the graph. This algorithm does not apply an ordering strategy and uses a depth-first backtracking procedure. The MACE algorithm has a similar complexity to ARBO (Chiba and Nishizeki 1985). However, its efficiency decreases when the graphs get larger due to an additional factor, n. With a new strategy, the EBBkC (Wang, Yu, and Long 2024) presents better time complexity for k-clique listing than the ARBO, MACE, and kClist algorithms. It presents an edge-based branching strategy that explores larger cliques by adding connected vertex pairs. Besides, it also introduces three-edge sorting methods and early branch termination and incorporates parallelization techniques for improved performance over traditional vertex-based approaches.

To summarize, all the algorithms discussed so far are algorithms that count maximal cliques efficiently, using their own strategies and data structures for this purpose. These differences reflect trade-offs between memory usage, computational efficiency, and suitability for various computational environments. Bron-Kerbosch (Bron and Kerbosch 1973) and Akkoyunlu (Akkoyunlu 1973) emphasize direct enumeration, (Tomita, Tanaka, and Takahashi 2006) focus on memory-efficient structuring of the output, (Eppstein, Löffler, and Strash 2010) optimize through degeneracy ordering, and (Dasari, Ranjan, and Mohammad 2014) extend the strategy further with specialized data structures and distributed processing. While these algorithms use a similar approach to those proposed in Bron-Kerbosch, except the ones proposed in the paper MACE, they offer different strategies.

(Finocchi, Finocchi, and Fusco 2015) presents a MapReduce-based version of the ARBO algorithm and provides an approximate solution to relax the limitation of the exact approach. This algorithm complexity grows exponentially with k but efficiently handles smaller cliques (lower k) in dense graphs.

Chiba and Nishizeki's ARBO (Chiba and Nishizeki 1985) framework is improved by the kClist (Danisch, Balalau, and Sozio 2018) algorithm incorporating degeneracy ordering and parallelization techniques to enhance performance, particularly in handling large-scale graphs.

(R. Li et al. 2020) presents a k-clique listing and counting approach based on the color orientation technique, which differs from the typical degree and degeneracy orientation methods. Similar to ordering-based k-clique algorithms such as kClist, this method deviates from the traditional degeneracy ordering approach. Instead, it employs color ordering to list k-cliques within graphs effectively. And (R. Li et al. 2020) presents a decision tree to help select the most suitable k-clique listing algorithm based on different scenarios. This algorithm has similar time complexity with kClist but scales with maximum degree Δ instead of arboricity, making it suitable for sparse graphs. The SDegree and BitCol algorithms (Z. Yuan et al. 2022) claim to have comparable time complexity and slightly better space efficiency than the algorithms proposed by (R. Li et al. 2020).

Exact k-clique counting algorithms depend on k-clique enumeration, which becomes infeasible for large graphs with high k values (e.g., $k \geq 8$) due to combinatorial explosion. The Pivoter algorithm (Jain and Seshadhri 2020), inspired by the Bron-Kerbosch algorithm (Bron and Kerbosch 1973), tackles this issue. The critical innovation of Pivoter is its ability to implicitly construct a succinct clique tree (SCT) using a pivoting technique during the search process. This SCT structure provides a unique and compact representation of all k-cliques, significantly reducing the space required compared to the total number of k-cliques. However, the authors of Pivoter note that there are certain datasets, such as $com-lj$ (Leskovec and Krevl 2014); even the parallel version of Pivoter struggled to count beyond $k = 10$. The Pivoter algorithm is also suitable for large and sparse graphs due to its time complexity, which depends on arboricity.

In response to the challenges of combinatorial explosion, there has been a shift towards approximation solutions using sampling methods. The Turán-shadow algorithm (Jain and C. Seshadhri 2017) is the state-of-the-art sampling-based approximate k-clique

algorithm for k values up to 10. This algorithm constructs a recursion tree, the Turán-shadow, to create dense subgraphs covering the entire graph, followed by an unbiased estimator to count the cliques. However, this process is time-intensive due to the construction of the shadow. Building on this approach, the YACC algorithm (Jain and Tong 2022) reduces the recursion tree size to handle larger k values (up to 40) by relaxing the stopping condition during tree creation, improving efficiency but at the cost of accuracy. This reduction requires a significant increase in the number of samples to maintain reliable estimates. Both algorithms represent substantial advancements in approximate k-clique counting, balancing efficiency and accuracy through innovative techniques. The Turán-shadow algorithm is highly efficient for counting cliques in large, sparse graphs with low arboricity. Since the time and space complexity depends on the arboricity of the graph, it is suitable for sparse graphs. The YACC algorithm provides the same time and space complexity as Turán-shadow.

The algorithm ERS developed by (Eden, Ron, and Seshadhri 2018) is a randomized method for approximating the number of k-cliques in a given graph. ERS uses a query model, unlike the Turán-shadow algorithm (Jain and C. Seshadhri 2017), which relies on constructing the Turán-shadow. ERS is more space-efficient compared to the memory-intensive Turán-shadow. However, while ERS theoretically achieves a $1+\varepsilon$ approximation, its practical accuracy tends to be lower than that of Turán-shadow, as demonstrated in experiments reported by (R. Li et al. 2020). Additionally, (R. Li et al. 2020) illustrates that for approximation algorithms, the worst-case time complexity of Turán-shadow is typically higher than that of ERS. Nonetheless, Turán-shadow's time overhead remains substantially lower than most exact algorithms.

The ARB-Count (Shi, Dhulipala, and Shun 2021) demonstrates it significantly outperforms the state-of-the-art parallel kClist algorithm and the parallel version of Pivoter. While Pivoter can handle all cliques in some large graphs, it could be more efficient for fixed k values and faces substantial slowdowns, particularly for smaller k. Moreover, Pivoter requires considerable memory and help with large graphs, often running out of space and failing to compute k-clique counts for higher k values. In contrast, ARB-Count shows superior performance and efficiency, making it more suitable for practical use.

The paper (Almasri et al. 2022) compares GPU implementations with two CPU baselines: ARB-Count (Shi, Dhulipala, and Shun 2021), the top parallel graph orientation

method, and Pivoter (Jain and Seshadhri 2020), the leading parallel pivoting method. Two key observations are presented. First, for small values of k, the graph orientation approach outperforms pivoting, which excels for larger k values; this pattern holds for both CPU and GPU. Typically, the pivoting approach becomes superior around $k = 7$. Second, the best GPU implementation consistently outperforms the best parallel CPU implementation across all k values. The ARB-Count and kClist (Danisch, Balalau, and Sozio 2018) algorithms are based on the ARBO (Chiba and Nishizeki 1985) algorithm, and ARB-Count has better space complexity than kClist. The exact version is suitable for large and sparse graphs with low arboricity. The approximate version is suitable for large, sparse datasets where exact counting is infeasible.

(Ye et al. 2023) proposes a framework for estimating the number of k-cliques by integrating Pivoter with three novel dynamic programming and color-based sampling techniques. The k-color set sampling algorithm's time and space complexities are affected by the number of colors used and k values. It can be suitable for small k values and graphs with low chromatic numbers (χ). The k-color path sampling algorithm is suitable for small and moderate-size graphs and smaller k values, as the complexity increases exponentially regarding n and k values. The complexity of the k-triangle algorithm depends on the number of triangles in the input graph and the desired clique size. This algorithm can suffer from scalability problems, especially for large, dense datasets with many triangles. This paper states that these sampling techniques outperform kClist and Pivoter across various datasets and k values. Additionally, they note that the space overheads of their algorithms and Pivoter are comparable, while kClist consumes slightly more space than their methods. The SR-kCCE (Chang, Gamage, and Yu 2024) algorithm provides an efficient approximate k-clique counting algorithm with guaranteeing accuracy. It does not specify explicit time and space complexity but claims to generally outperform DPColorPath in execution time while being compatible with both Pivoter and DPColorPath regarding memory usage.

The algorithms whose time complexity depends on degeneracy or arboricity often perform well on large real-world graphs because the degeneracy and arboricity of a graph are much smaller than the maximum degree of the graph.

Table 4.1. A comparison of algorithms based on different characteristics.

Algorithm	Approximate	Exact	Parallelization	Base algorithm
Bron-Kerbosch (Bron and Kerbosch 1973)		enumeration		*
(Akkoyunlu 1973)		enumeration		
ARBO (Chiba and Nishizeki 1985)		enumeration		*
MACE (Makino and Uno 2004)		enumeration		
(Tomita, Tanaka, and Takahashi 2006)		enumeration		Bron-Kerbosch
(Eppstein, Löffler, and Strash 2010)		enumeration		Bron-Kerbosch
pbitMCE (Dasari, Ranjan, and Mohammad 2014)		enumeration	MapReduce	Bron-Kerbosch
(Finocchi, Finocchi, and Fusco 2015)	color-based	enumeration	MapReduce	ARBO
Turán-shadow (Jain and C. Seshadhri 2017)	random			
kClist (Danisch, Balalau, and Sozio 2018)		enumeration	shared memory	ARBO
(Eden, Ron, and Seshadhri 2018)	rejection			
(R. Li et al. 2020)		enumeration	shared memory	ARBO
Pivoter (Jain and Seshadhri 2020)		counting	shared memory	Bron-Kerbosch
ARB-Count (Shi, Dhulipala, and Shun 2021)	color-based	enumeration	shared memory	ARBO
YACC (Jain and Tong 2022)	random			Turán-shadow
(Z. Yuan et al. 2022)		enumeration	shared memory	ARBO
(Almasri et al. 2022)		counting	GPU	Bron-Kerbosch
DPColorPath (Ye et al. 2023)	color-based		shared memory	
EBBkC (Wang, Yu, and Long 2024)		enumeration	shared memory	
SR-kCCE (Chang, Gamage, and Yu 2024)	random			
BDAC (Çalmaz and Bostanoğlu 2024a)	without sampling			

* These algorithms themselves are base algorithms.

Table 4.2. A comparison of algorithms based on additional characteristics.

Algorithm	Orientation	Objective	Time complexity	Space complexity
Bron-Kerbosch (Bron and Kerbosch 1973)		maximal	$O(3^{n/3})$	$O(m+n)$
(Akkoyunlu 1973)		maximal	$O(3^{n/3})$	$O(m+n)$
ARBO (Chiba and Nishizeki 1985)		k-clique	$O(km\alpha^{k-2})$	$O(m+n)$
MACE (Makino and Uno 2004)		maximal	$O(knm\alpha^{k-2})$	$O(m+n)$
(Tomita, Tanaka, and Takahashi 2006)		maximal	$O(3^{n/3})$	$O(m+n)$
(Eppstein, Löffler, and Strash 2010)	degeneracy	maximal	$O(dn3^{d/3})$	$O(m+n)$
pbitMCE (Dasari, Ranjan, and Mohammad 2014)	degree/degeneracy	maximal	$O(kn3^{k/3})$	not reported
(Finocchi, Finocchi, and Fusco 2015)	degree	k-clique	$O(m^{k/2})$	$O(m+n)$
Turán-shadow (Jain and C. Seshadhri 2017)	degeneracy	k-clique	$O(n\alpha^{k-1})$	$O(n\alpha^{k-2}+m)$
kClist (Danisch, Balalau, and Sozio 2018)	degeneracy	k-clique	$O(m\alpha^{k-2})$	$O(m+P\alpha^2)$
(Eden, Ron, and Seshadhri 2018)	degree	k-clique	$\tilde{O}\left(\frac{n}{C_k^{1/k}}+\frac{m^{k/2}}{C_k}\right)$	$O(m+n)$
(R. Li et al. 2020)	color	k-clique	$O(km\frac{\Delta}{2}^{k-2})$	$O(m+n)$
Pivoter (Jain and Seshadhri 2020)	degeneracy	k-clique	$O(n\alpha 3^{\alpha/3})$	$O(m+n)$
ARB-Count (Shi, Dhulipala, and Shun 2021)	degeneracy	k-clique	exact: $O(m\alpha^{k-2})$ approx.:$O(pm\alpha^{k-2}+m)$	$O(m+P\alpha)$
YACC (Jain and Tong 2022)	degeneracy	k-clique	$O(n\alpha^{k-1})$	$O(n\alpha^{k-2}+m)$
(Z. Yuan et al. 2022)	degree/color	k-clique	$O(km\left(\frac{\Delta}{2}\right)^{k-2})$	SDegree: $O(m+kN\Delta)$ BitCol: $O(m+N\frac{\Delta^2}{L})$
(Almasri et al. 2022)	degeneracy/degree	k-clique	not reported	$O(d^2max)$
DPColorPath (Ye et al. 2023)	degeneracy	k-clique	k-color set: $O(\chi^k)$ k-color path: $O(\chi^{nk}+m)$ k-triangle path: $O(k\Delta)$	k-color set: $O(m+n+\chi^k)$ k-color path: $O(kn+m)$ k-triangle path: $O(km)$
EBBkC (Wang, Yu, and Long 2024)	color	k-clique	$O(md+k\cdot m\cdot(\frac{\tau}{2})^{k-2})$	$O(m+n)$
SR-kCCE (Chang, Gamage, and Yu 2024)	degeneracy	k-clique	not reported	not reported
BDAC (Çalmaz and Bostanoğlu 2024a)	degeneracy	k-clique	$O(\alpha^2)$	$O(m+n+\alpha^2)$

4.4. Motivation

Developing and analyzing graphlet counting algorithms that gracefully scale poses a significant challenge, mainly when dealing with graphs of substantial dimensions. Graphlet counting, in particular, demands intensive computational resources, and the running times of algorithms can quickly become impractical, even when assuming the input graphs are manageable by standard hardware. In the literature, even the ESCAPE algorithm (Pinar, Seshadhri, and Vishal 2017), recognized as the state-of-the-art for exact graphlet counting, handles graphlets up to five nodes. Although exact counting is viable for smaller graphs, such as those found in bioinformatics, it becomes increasingly unfeasible for the larger, more complex graphs prevalent in social and information networks or larger graphlet sizes due to combinatorial explosion.

Various algorithms have been developed to tackle the scalability challenge, with approximation emerging as a prevalent strategy. Often leveraging sampling and statistical methods, approximation involves selecting a subset of nodes and edges from a graph to encapsulate its essential structural characteristics, thereby ensuring computational feasibility. Another standard method is to speed up computation by taking advantage of parallelism.

Graphlets are crucial for understanding local network structure variations and comparing and characterizing real-world networks. Due to their importance, certain graphlets, such as cycles and cliques, have garnered significant attention in the literature. The cycle pattern (Figure 2.8d) is used for fraud detection (Qiu et al. 2018). Among these patterns, cliques stand out due to their numerous applications across different application areas. Clique analysis is vital in network analysis, with even the simplest form, the triangle, being a central focus of recent research. Section 4.2 provides a usage of clique counts in diverse application areas.

The literature contains numerous algorithms designed to list or count k-cliques (Chiba and Nishizeki 1985; Finocchi, Finocchi, and Fusco 2015; Danisch, Balalau, and Sozio 2018; R. Li et al. 2020; Jain and Seshadhri 2020). However, for large datasets or high values of k, the combinatorial growth of the problem makes exact and listing-based methods computationally infeasible. The exact solutions can be applied to relatively small-scale datasets where k is less than 10. Even parallel versions of these algorithms struggle

when k exceeds 10. Approximation methods have gained attention to address this, as they are practical when exact counts are infeasible (Jain and C. Seshadhri 2017; Eden, Ron, and Seshadhri 2018; Shi, Dhulipala, and Shun 2021; Jain and Tong 2022; Ye et al. 2023; Chang, Gamage, and Yu 2024). Most approximation algorithms use sampling strategies to select a representative sample set and draw samples from it to estimate k-cliques. These methods often rely on recursive search trees due to the structure of the problem. Constructing a recursive search tree becomes impractical for large datasets and higher k-values due to the exponential growth of the search space. In large and dense datasets, the branches of a recursive search tree grow exponentially, leading to combinatorial explosion and memory overhead. However, achieving good sampling results relies on obtaining high-quality samples from recursive search tree. Some algorithms enhance the performance of existing approaches through parallel strategies or offer parallel solutions to improve scalability and handle larger datasets more efficiently (Finocchi, Finocchi, and Fusco 2015; Danisch, Balalau, and Sozio 2018; R. Li et al. 2020; Jain and Seshadhri 2020; Shi, Dhulipala, and Shun 2021; Almasri et al. 2022; Ye et al. 2023; Wang, Yu, and Long 2024). An extensive survey (Çalmaz and Bostanoğlu 2024b) presents a comprehensive analysis of k-clique counting algorithms, examining their methodologies, applicability across various scenarios, and limitations.

After discussing the limitations and challenges outlined earlier, this study notes that extremal graph theorems (Turán 1941; Erdős 1969) in the literature have previously been applied to address this problem (Jain and C. Seshadhri 2017). These theorems typically focus on determining whether a graph can contain cliques, given certain density conditions. In our work, we explore and apply other theorems (Zykov 1949; Kruskal 1963; Katona 1987; Reiher 2016) that provide insight into the number of cliques a graph with a specific density can hold. Some of these extremal theorems give minimum and maximum clique counts a graph, which satisfies a specific density, can contain. Consequently, we plan to eliminate the recursive search tree and sampling phase, instead providing a boundary for clique counts.

As a summary, this chapter has outlined the clique counting problem, its challenges, application across various domains. It also presents the development of the clique counting problem by discussing the two main algorithms and reviewing other methods that build on them. We have given a thorough overview of clique counting strategies, covering both exact

and approximate methods. We also discussed the pros and cons of different techniques and highlighted the importance of clique counting in understanding the structure of complex networks. The knowledge gained here is essential for understanding our proposed method. Finally, the motivation of this study is explained.

CHAPTER 5

PROPOSED K-CLIQUE APPROXIMATIONS ALGORITHMS

This section introduces the proposed approximate k-clique counting algorithm, BDAC (Boundary-driven approximation of k-cliques), with a numerical example. Then, it presents the algorithms' time and space complexity compared with BDAC. In the last section, the parallel version of BDAC is introduced.

The main challenge of counting k-cliques arises from the overwhelming number of possibilities, known as a combinatorial explosion. Hence, methods have been developed to sample large, dense subgraphs, mainly to count larger k-cliques. However, these sampling methods require assumptions about the distribution or sufficiency of the sample count.

This algorithm aims to approximate the number of k-cliques (where $k \geqslant 3$) in the graph G. The methodology estimates lower and upper bounds for k-cliques through extremal combinatorics, based on each node's edge and triangle density. We use these measures to estimate the total number of k-cliques in the entire dataset.

The proposed method avoids relying on sampling by leveraging insights from established research in extremal combinatorics. By setting boundaries, we aim to eliminate the need to make any assumptions about distributional properties. We also enhance the algorithms performance and scalability by employing CPU-based parallelization.

5.1. BDAC (Boundary-Driven Approximations of K-cliques)

The BDAC (Boundary-driven approximations of k-cliques) algorithm draws inspiration from the Turán-shadow algorithm (Jain and C. Seshadhri 2017), which leverages classic extremal combinatorics principles concerning clique densities. Turán's (Turán 1941) and Erdős (Erdős 1969) established lower bounds on the number of cliques within sufficiently dense graphs. The novelty of our algorithm lies in integrating additional theorems such as Zykov (Zykov 1949), Kruskal-Katona (Katona 1987), (Kruskal 1963), and Reiher (Reiher 2016) to provide both lower and upper bounds for dense graphs.

Turán-shadow (Jain and C. Seshadhri 2017) method involves constructing a set of dense subgraphs, known as a Turán-shadow, to cover graph G comprehensively and include all cliques. Then, it uses standard techniques to design an unbiased estimator for the clique count. The Turán-shadow algorithm dedicates a significant portion of its overall runtime to constructing the shadow. In contrast, sampling constitutes a small fraction of the total execution time.

The BDAC algorithm innovatively bypasses the requirement for a Turán-shadow and sampling techniques, offering a direct and efficient method for estimating k-clique counts in complex networks. By traversing each node in the graph, we establish lower and upper bounds for potential k-cliques formed by each vertex (local bound per vertex), aggregating these bounds to determine the approximation of k-cliques without needing a Turán-shadow (global bound). This streamlined approach ensures high-speed performance, significantly reducing computational overhead and facilitating efficient approximation of k-clique counts. Besides, this algorithm also provides insights into local k-clique counts formed by each vertex.

We compare the BDAC algorithm and Turán-shadow regarding their execution time and results, mainly focusing on relatively small datasets. When dealing with large datasets, the applicability of Turán-shadow diminishes significantly. As a result, we compared the BDAC algorithm with YACC (Jain and Tong 2022), a modified version of Turán-shadow designed for such scenarios. Unfortunately, we cannot access the source code of YACC, which prevents a direct comparison of their execution times. As a result, we can only compare algorithm estimation results across identical datasets. The outcomes of the Pivoter (Jain and Seshadhri 2020) algorithm provide exact values for comparing the algorithm in terms of relative error. However, Pivoter is not scalable to handle larger datasets (like com-lj (Leskovec and Krevl 2014)). We also compare BDAC with the DPColorPath (Ye et al. 2023) regarding the estimation results and execution time. A comprehensive comparison of these state-of-the-art algorithms is provided, focusing on k values ranging from 8 to 50, to analyze the effect of k values on the datasets. As a result, the requirement for exact values for large datasets prevents a comparative assessment.

Besides, we enhance the performance of the BDAC algorithm through CPU-based parallelization techniques. By utilizing parallelization, this study aims to improve the computational efficiency of BDAC, offering a more effective solution for large-scale

graph analysis. We compare BDAC's execution time with the most recent parallel k-clique counting algorithm, DPColorPath. The evaluations demonstrate that the parallel version of BDAC is highly compatible with DPColorPath, particularly when handling larger datasets and larger k values ($k < 50$). Notably, the BDAC provides a unique advantage by offering a boundary at the vertex level, a feature not found in other algorithms in the literature, except for Pivoter (Jain and Seshadhri 2020), which provides an exact solution but is only suitable for $k < 10$ on large datasets.

The primary innovation of BDAC occurs with its extraction of edge density information from the neighborhood of a node, facilitating the determination of the triangle density ($d(K_3)_v$) associated with that node. Because the count of edges connecting the neighbors of a given node corresponds to the number of triangles formed by that node, at this point, we use Zykov's theorem instead of Turán's theorem. Zykov's theorem extends Turán's theorem. Formally, for any positive integers r and s such that $2 \leq r \leq k$, if the density of r-clique satisfies the Zykov threshold, then the graph contains at least one k-clique.

The pseudocode illustrating BDAC is shown in the Algorithm 5. **Line 5** in the pseudocode, H, represents the subgraph of the current node's neighbors with edges between them. **Line 8** calculates the edge density of subgraph H ($d(K_2; H)$), which is also equal to the triangle density of the current node.

Now that we have triangle density information $d(K_3)$ for each node, we can determine the presence of k-clique (K_k). By using triangle density information $d(K_3)$ instead of edge density $d(K_2)$, we obtain a relatively lower threshold. Suppose a node's triangle density $d(K_3)$ satisfies the Zykov threshold. In that case, the next step involves deriving the upper bound of k-cliques obtained from that node and its neighborhood using the Kruskal-Katona theorem explained above in Theorem 2.4.4.

In **line 12**, $calMaxCliq$ is a procedure that computes the maximum cliques in subgraph H utilizing Kruskal-Katona's theorem. The algorithm requires the density of subgraph H, the number of nodes in subgraph H, and the size of the clique we search in H as input parameters.

If the density satisfies the threshold, we establish a lower k-cliques bound. We initially employed Reiher's theorem to verify the lower bound for K_k, as outlined in Theorem 2.4.5.

In **line 15**, $calMinCliqReiher$ is a procedure that calculates the minimum clique counts in subgraph H utilizing Reiher's theorem. The inputs of this procedure are the $gamma$ explained in Theorem 2.4.5, the number of nodes in subgraph H, and the size of the clique that we search in H. However, in instances where $s \leq k$ - 2, the diminished value of the binomial coefficient $\binom{s+1}{k}$ renders Reiher's theorem inadequate for determining the presence of K_s. In this scenario, the procedure returns -1, and then we verify if $d(K_2; H) > \frac{k-2}{k-1}$ holds. If it does, we apply Erdős's theorem (**line 19**). Otherwise, we accept the lower bound as 0.

An additional differentiation from the Turán-shadow algorithm occurs when the edge density of a node and its neighborhood fails to meet the Turán threshold. In such cases, the algorithm recurses on the set of vertices within that neighborhood, constructing a recursion tree named Turán-shadow within the Turán-shadow algorithm. Significantly, the algorithm dedicates a notable portion of its overall computational time to establishing this recursion tree.

Conversely, if a node and its triangle density do not satisfy the Zykov threshold, we omit that particular node from consideration. Consequently, in this scenario, we abstain from constructing any recursion tree.

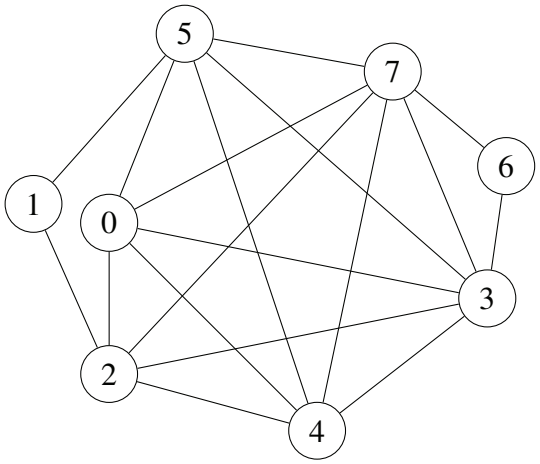
As a final step, once we have obtained both the lower and upper bounds from each node that meets the Zykov threshold, we combine these bounds to calculate the overall lower and upper bounds. These aggregated bounds enable us to estimate the final count of k-cliques (lines 13 and 24).

5.2. Example

This section presents an illustrative example aimed at elucidating the theorems discussed. Figure 5.1 illustrates a sample graph G and the out-degrees of each node after degeneracy ordering. In this example, we aim to estimate 5-cliques. Following the ordering, the algorithm traverses each node individually. If the neighbor count satisfies the desired clique count minus one, which is 4 (excluding the current node itself), we check whether the density satisfies the threshold. In this example, only node 0 has sufficient neighbors to form the desired clique.

Algorithm 5 BDAC

```
1: procedure APPROXIMATE_K_CLIQUE(Graph G, DAG D, Integer k)
2:     totalMax ← 0
3:     totalMin ← 0
4:     for all node in G do
5:         H ← neighbors of current node in D
6:         n ← the no. of vertices of subgraph H
7:         m ← the no. of edges of subgraph H
8:         density ← m/(n 2)
9:         k ← k-1
10:        zykovThres ← (k-1)(k-2)(k-3)/(k-1)^3
11:        γ ← m/n^2
12:        maxCliq ← calMaxCliq(n, density, k)
13:        totalMax+ ← maxCliq
14:        if density > zykovThres then
15:            minCliq ← calMinCliqReiher(n, γ, k)
16:            if minCliq == −1 then
17:                turanThres ← (k-2)/(k-1)
18:                if density > turanThres then
19:                    minCliq ← (n/(k−1))^(k−2)
20:                else
21:                    minCliq = 0
22:                end if
23:            end if
24:            totalMin+ ← minCliq
25:        end if
26:    end for
27:    PRINT totalMin
28:    PRINT totalMax
29: end procedure
```



(a) A sample graph G.

$0 \rightarrow 3,4,7$

$1 \rightarrow 2,5$

$2 \rightarrow 0,3,4,7$

$3 \rightarrow 4,7$

$4 \rightarrow 7$

$5 \rightarrow 0,3,4,7$

$6 \rightarrow 3,7$

$7 \rightarrow$

(b) Out-degree neighbors of each node after degeneracy orientation.

Figure 5.1. Visualization of a graph (a) and its node relationships after applying degeneracy ordering (b).

The following Figure 5.2 illustrates the neighbors of node 2 and node 5, along with the edges between these neighbors (induced graph) represented as a subgraph H. Both nodes yield the same induced subgraph. Then, the density of this subgraph H is calculated (density= $6/\binom{4}{2} = 2$) and checked to see if it satisfies the Zykov threshold for $k = 4, r = 3$ (threshold= 0.22). As previously mentioned, the edge density of subgraph H provides the triangle density of nodes 2 and 5, $d(K_2; H) = d(K_3)_2$ and $d(K_2; H) = d(K_3)_5$. Therefore, the r-value is 3. Later, the algorithm utilizes Reiher's theorem to establish the lower bound, yielding $s = 3$ and $\beta = 0$, with a **minimum clique count** of 1. Kruskal-Katona's theorem provides the **maximum clique count** as 1. The same results are generated by node 2 and 5, the **totalMin** is 2 and **totalMax** is 2, with the **exact** value also being 2. The 5-cliques in the G are $0 - 2 - 3 - 4 - 7$ and $0 - 3 - 4 - 5 - 7$.

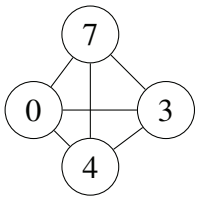


Figure 5.2. An induced subgraph H formed by nodes 2 and 5 separately.

5.3. Time and Space Complexity Comparisons

Both the Turán-shadow and YACC algorithms share the same time and space complexity. Each algorithm involves iterating over all vertices, which takes $O(n)$ time, where n is the number of vertices. During each iteration, the algorithm recursively searches for $(k-1)$-cliques within the neighborhoods of vertices. This recursive search operation takes $O(\alpha^{k-1})$ time, where α represents the degeneracy of the graph, indicating that each subgraph has at most α neighbors. Also, in each recursive search, the algorithm calculates the edge density. That means it checks the neighborhood of a current node, whether any of two neighbors form an edge $O(\alpha^2)$. Therefore, constructing the recursion tree for these operations has a time complexity of $O(n(\alpha^2 + \alpha^{k-1}))$. So, total complexity is $O(n\alpha^{k-1} + m)$, $O(m)$ for degeneracy orientation, m indicates the number of edges.

This complexity suggests that the algorithm's performance scales linearly with the number of vertices (n) but exponentially with the size of the structure (k), adjusted by the degeneracy (α). The (α^{k-1}) term indicates that for each vertex, the algorithm explores combinations of neighbors, but the degeneracy (α) limits the growth of these combinations, making the algorithm more efficient than a naive approach for dense graphs.

In summary, this complexity indicates an algorithm that is efficient for sparse graphs (where α is low) and for finding relatively small structures (where k is not too large), as the cost grows significantly with larger k values, especially in denser graphs where (α) is higher. The space complexity is $O(n\alpha^{k-2} + n + m)$, $O(n\alpha^{k-2})$ for the recursion tree and storing subsets of neighbors at each level, $O(n + m)$ for storing the original graph.

The time complexity of algorithms proposed by Ye et al. (Ye et al. 2023) are k-color sampling is $O(\chi^k)$, k-color path sampling $O(\chi^{nk} + m)$, and k-triangle sampling algorithms $O(k\Delta)$, where where χ is the number of colors of the graph G obtained by the greedy coloring algorithm (Hasenplaugh et al. 2014; L. Yuan et al. 2017), k is the clique size, n is the number of vertices, m is the number of edges and Δ is the number of triangles of the input graph. The k-color sampling algorithm considers all possible sets of k different colors, and in the worst case, there are χ^k such sets. For each set, it checks whether it forms a k-clique, which leads to $O(\chi^k)$ complexity. The space complexity is $O(n+m+\chi^k)$, $O(n+m)$ to store graphs and colors, $O(\chi^k)$ to store dynamic programming

table (DP). In the k-color path algorithm, χ^{nk} denotes the possible coloring of paths of length k over n vertices, and m denotes the traversal of each edge. The space complexity of its $O(nk+m)$ DP and DAG. The k-triangle algorithm samples a triangle and extends it to the k-clique. So, the time complexity is $O(k\Delta)$, and the space complexity is $O(km)$ to store the DP table.

The BDAC algorithm also iterates through each vertex in the graph. For every vertex, it examines pairs of its neighbors to determine if they form an edge. This checking operation, for each vertex, has a time complexity of $O(\alpha^2)$. Consequently, the overall time complexity of BDAC is $O(n\alpha^2)$, where n represents the number of vertices in the graph.

This complexity indicates that the time it takes for the algorithm to run scales linearly with the number of vertices (n). However, the time taken for each vertex scales quadratically with the degeneracy (α). This is because the algorithm checks pairs of neighbors for each vertex to determine whether they form an edge. The space complexity is $O(n+m+\alpha)$, $O(\alpha)$ for storing the nodes in induced subgraphs of each vertex, $O(n+m)$ for storing the original graph.

The BDAC gives better time and space complexity than Turán-shadow as it eliminates the recursion tree construction. If we compare the k-color set sampling with BDAC, ($O(\chi^k)$ vs. $O(n\alpha^2)$), in the worst-case scenario where χ close to n and causes $O(n^k)$ which is the higher time complexity of compared $O(n\alpha^2)$ complexity of the BDAC. Compared with the k-color path, the term χ^{nk} can grow extremely large, making it impractical for large datasets (n) and larger k. So, compared to BDAC, it is less efficient than BDAC for larger k and n. The time complexity of the k-triangle algorithm depends on the number of triangles of input and the clique size. Compared with BDAC, this algorithm can be more efficient for datasets with fewer triangles. Still, this algorithm can be comparable to or worse than BDAC for dense datasets with larger triangles.

As a result, The BDAC algorithm is generally more efficient than the sampling-based methods for large graphs because α is typically much smaller than n. Sampling approaches are effective in certain settings, such as when the number of colors is small. However, they become more complex for larger graphs, especially as k increases.

5.4. Parallel BDAC

Algorithm 6 provides the pseudocode for the parallel implementation of BDAC. In this study, the parallel implementation was developed using OpenMP (Dagum and Menon 1998), an API specifically designed for shared-memory architectures. This algorithm aims to process a graph G and its associated directed acyclic graph (DAG) D. The algorithm exploits the density information obtained from the subgraphs induced by the neighborhood of each vertex. It computes the minimum and maximum k-clique that these subgraphs can contain using the principles of extremal graph theories.

In this work, the computations for each node are independent. Each node's density and k-clique boundaries are calculated separately at each iteration. The OpenMP API simplifies parallelization by handling thread management, synchronization, and workload distribution, making it well-suited for tasks with no inter-dependencies. Algorithm 6 distributes the tasks performed by each node across the threads, with each task executed by an individual thread. It divides the workload (calculating the clique boundaries for each node) between existing threads that can run concurrently. This parallel approach speeds up the process, especially with large graphs. Each thread calculates local maximum and minimum clique counts of subgraphs induced by a neighborhood of the current node, and the results are then combined to give global bounds. In **line 2**, *num_threads* indicates the number of threads. In **line 4**, the array of *localMax* and *localMin* keeps each thread's execution results. Each thread has a unique id (*thread_id*), and this id is obtained by calling *GetThreadId()* function. The steps of this algorithm are similar to those of the BDAC algorithm, so there is no need to explain each step between **line 10-36** again. The primary difference is that this algorithm parallelizes the task for each node, distributing the work across multiple threads. Finally, the local counts for each thread are summed to obtain the final results (**line 37 and 38**).

Algorithm 6 parallel BDAC

```
1: procedure PARALLEL_BDAC(Graph G, DAG D, Integer k)
2:     num_threads ← Set to max available
3:     totalMax ← 0, totalMin ← 0,
4:     localMax = [0.0] * num_threads, localMin = [0.0] * num_threads
5:     Parallel Section:
6:       Divide the work across multiple threads
7:     for each thread in parallel do
8:         thread_id ← GetThreadId()
9:         for all node in G do
10:            H ← Construct induced subgraph of current node's out-nbrs in D
11:            k ← k-1
12:            n ← the no. of vertices of subgraph H
13:            if n < k then
14:                continue
15:            end if
16:            m ← the no. of edges of subgraph H
17:            density ← m/(n 2)
18:            zykovThres ← (k-1)(k-2)(k-3)/(k-1)^3
19:            γ ← m/n^2
20:            γ ← m/n^2
21:            maxCliq ← MaxCliq(n, density, k)
22:            localMax[thread_id]+ ← maxCliq
23:            if density > zykovThres then
24:                minCliq ← calMinCliqReiher(n, γ, k)
25:                if minCliq == −1 then
26:                    turanThres ← (k-2)/(k-1)
27:                    if density > turanThres then
28:                        minCliq ← (n/(k−1))^(k−2)
29:                    else
30:                        minCliq = 0
31:                    end if
32:                end if
33:                localMin[thread_id]+ ← minCliq
34:            end if
35:        end for
36:    end for
37:    totalMax ← Σ_{t=1}^{num_threads} localMax[t]
38:    totalMin ← Σ_{t=1}^{num_threads} localMin[t]
39:    PRINT totalMin
40:    PRINT totalMax
41: end procedure
```

CHAPTER 6

EXPERIMENTS

This chapter presents the properties of the environment and datasets used for experiments. It subsequently compares the sequential and parallel versions of the proposed algorithm, BDAC, with other clique counting algorithms.

6.1. Experimental setup

We perform a comparative analysis of BDAC, including its parallel version, results against the Turán-shadow (Jain and C. Seshadhri 2017) algorithm, Pivoter (Jain and Seshadhri 2020), and DPColorPath (Ye et al. 2023) utilizing publicly available C++ implementations. The codes of neither the DPColorTriangle (Ye et al. 2023) nor YACC (Jain and Tong 2022) codes are not publicly available. So, we can not compare our algorithm with the DPColorTriangle. We also compare BDAC, YACC, and DPColorPath algorithms regarding the estimation results. The results for YACC are used as detailed in the paper (Jain and Tong 2022). Throughout our experiments, we leverage and refine the existing C++ implementations of the Turán-shadow algorithm. We thank the authors of Turán-shadow, Pivoter, and DPColorPath for generously sharing their codes. The assessment of all algorithms occurs on a PC equipped with two 2.2 GHz Intel(R) Xeon(R) Silver 4114 CPUs (10 cores), 640KB L1 cache, 10MB L2 cache, 13MB L3 cache, and 32GB of memory.

We utilize a diverse selection of datasets obtained from both SNAP (Leskovec and Krevl 2014) and Network Repository (Rossi and Ahmed 2015). Table 6.1 illustrates the datasets and their respective properties. The value labeled as α in 6.1 indicates the degeneracy of the dataset, which is further explained in Section 2. The provided datasets display a wide range of characteristics in terms of their size and density. "Web-Stanford" and "soc-pokec" are notable for their relatively low degeneracy values, indicating sparser graphs with nodes of lower degrees, suggesting minimal interconnectivity. On the other hand, "com-lj" and "soc-LJ" demonstrate high degeneracy values, signifying denser graphs with significantly higher connectivity, even within their subgraphs. "Web-BerkStan"

shows moderate to high degeneracy, suggesting graphs with varying levels of density and connectivity. "as-skitter" and "com-orkut" exhibit moderate degeneracy values, indicating graphs with relatively high connectivity but not as dense as the "com-lj" or "soc-LJ" datasets. "Uk-2002" is the most degenerate of the datasets we are working on. This may indicate that it contains dense structures and large cliques, which are computationally challenging. This comparison highlights the diverse nature of the datasets, showcasing variations in graph density and connectivity across different network structures.

Table 6.1. Dataset properties.

graph	n	m	α
web-Stanford	2.82E+05	1.99E+06	71
web-BerkStan	6.85E+05	6,65E+06	201
as-skitter	1.7E+06	4.3E+06	111
soc-pokec	1.6E+06	1.1E+07	47
com-lj	4.0E+06	2.2E+07	360
soc-LJ	4.8E+06	4.2E+07	372
com-orkut	3.0E+06	1.1E+08	253
uk-2002	1.8E+07	2.9E+08	1885

6.2. Experimental results of BDAC

We compare the performance of the BDAC algorithm with Turán-shadow, Pivoter, and DPColorPath in terms of results, including estimations, exact values, and boundaries, as shown in Table 6.2.The BDAC algorithm does not provide an estimation. Instead, it determines the boundaries, specifying the minimum and maximum number of k-cliques a graph can contain. The Pivoter algorithm provides exact values for comparing algorithm results. If exact values are unavailable, Table 6.2 indicates the corresponding entry as "unknown". The BDAC is also compared with another sampling-based DPColorPath algorithm. The results of the DPColorPath algorithm are obtained from 500K samples.

Table 6.2. Comparison of the boundaries provided by BDAC, the exact values (if available) from Pivoter, and the estimated values from TuránShadow and DPColorPath.

		Approx.	Exact	Approx.	Estimation	
dataset	k	BDAC-min	Pivoter	BDAC-max	Turán	DPColorPath
tech	8	7,67E+10	4,82E+11	5,59E+12	4,80E+11	4,81E+11
	15	2,70E+15	1,11E+16	8,80E+18	1,10E+16	1,14E+16
	20	3,48E+17	1,28E+18	2,38E+22	1,28E+18	1,13E+18
	25	7,60E+18	3,04E+19	1,43E+25	3,01E+19	2,36E+19
	40	8,91E+18	5,09E+19	2,41E+30	5,12E+19	5,09E+19
	50	4,23E+15	3,05E+16	4,96E+31	2,94E+16	3,05E+16
web-Stanford	8	1,39E+11	2,18E+11	3,53E+11	2,18E+11	2,18E+11
	15	9,98E+14	2,95E+15	1,28E+16	2,95E+15	2,95E+15
	20	6,04E+16	1,53E+17	1,52E+18	1,53E+17	1,53E+17
	25	6,46E+17	1,30E+18	2,83E+19	1,29E+18	1,30E+18
	40	7,68E+16	1,09E+17	1,08E+19	1,07E+17	1,09E+17
	50	2,43E+12	3,95E+12	1,25E+15	3,77E+12	3,95E+12
web-BerkStan	8	1,83E+14	1,84E+14	1,84E+14	1,83E+14	1,84E+14
	15	3,48E+22	3,48E+22	3,49E+22	3,48E+22	3,48E+22
	20	3,95E+27	3,96E+27	3,97E+27	3,95E+27	3,96E+27
	25	1,16E+32	1,16E+32	1,70E+32	1,16E+32	1,16E+32
	40	6,12E+42	6,14E+42	6,25E+42	6,13E+42	6,14E+42
	50	1,50E+48	1,51E+48	1,55E+58	1,50E+48	1,51E+48
soc-pokec	8	2,06E+07	1,11E+08	3,06E+10	1,11E+08	1,13E+08
	15	2,80E+08	3,54E+08	5,57E+12	3,54E+08	3,55E+08
	20	3,14E+07	4,49E+07	9,41E+12	4,49E+07	4,49E+07
	25	5,05E+04	1,22E+05	3,62E+12	1,22E+05	1,22E+05
com-lj	8	1,54E+16	1,69E+16	2,09E+16	unknown	1,69E+16
	15	2,20E+26	unknown	2,64E+26	unknown	2,27E+26
	20	5,28E+32	unknown	6,77E+32	unknown	5,48E+32
	25	3,32E+38	unknown	4,94E+38	unknown	3,55E+38
	40	1,96E+53	unknown	6,13E+53	unknown	2,51E+53

(cont. on next page)

Table 6.2 (cont.).

	50	2,92E+61	unknown	1,76E+62	unknown	4,34E+61
soc-LJ	8	3,00E+16	unknown	4,24E+16	unknown	3,28E+16
	15	4,50E+26	unknown	8,19E+26	unknown	5,23E+26
	20	1,05E+33	unknown	2,67E+33	unknown	1,31E+33
	25	6,47E+38	unknown	2,39E+39	unknown	8,58E+38
	40	3,50E+53	unknown	5,10E+54	unknown	5,86E+53
	50	4,56E+61	unknown	2,10E+63	unknown	9,69E+61
com-orkut	8	1,52E+11	unknown	3,89E+15	unknown	1,56E+12
	15	1,98E+14	unknown	2,24E+23	unknown	8,58E+15
	20	2,24E+15	unknown	1,04E+28	unknown	1,99E+17
	25	**2,93E+15**	unknown	1,39E+32	unknown	**2,43E+14**
	40	2,30E+09	unknown	1,10E+42	unknown	1,16E+13
	50	0	unknown	1,07E+47	unknown	1355
uk-2002	8	1,86E+19	unknown	1,87E+19	unknown	1,86E+19
	15	2,95E+32	unknown	2,95E+32	unknown	2,95E+32
	20	1,07E+41	unknown	1,07E+41	unknown	1,07E+41
	25	1,11E+49	unknown	1,11E+49	unknown	1,11E+49
	40	5,29E+70	unknown	5,29E+70	unknown	5,29E+70
	50	4,92E+83	unknown	4,92E+83	unknown	4,92E+83

The BDAC, Turán-shadow, Pivoter, and DPColorPath algorithms are also compared based on their execution time, as shown in Table 6.3. For large datasets, Turán-shadow and Pivoter algorithms are terminated after a specified duration, as indicated by "terminated" in Table 6.3.

Table 6.3. Execution time comparison of BDAC, Pivoter, TuránShadow, and DPColorPath algorithms.

		Time (sec)			
dataset	k	BDAC	Pivoter	Turán	DPColorPath
tech	8	130	88	188	6
	15	125	88	257	6
	20	126	88	318	7
	25	116	88	387	8
	40	127	88	424	22
	50	129	88	235	41
web-Stanford	8	13	3	53	1
	15	12	3	53	1
	20	11	3	48	1
	25	13	3	41	1
	40	13	3	31	1
	50	12	3	5	1
web-BerkStan	8	105	13	5	3
	15	106	13	257	4
	20	106	13	208	4
	25	105	13	162	4
	40	106	13	338	4
	50	106	13	328	4
soc-pokec	8	25	47	46	15
	15	26	47	44	13
	20	24	47	40	12
	25	26	47	38	10
com-lj	8	68	terminated	terminated	27
	15	66	terminated	terminated	29
	20	68	terminated	terminated	31
	25	68	terminated	terminated	34
	40	68	terminated	terminated	35

(cont. on next page)

Table 6.3 (cont.).

	50	69	terminated	terminated	39
soc-LJ	8	128	terminated	terminated	36
	15	128	terminated	terminated	46
	20	126	terminated	terminated	51
	25	128	terminated	terminated	57
	40	128	terminated	terminated	96
	50	**128**	terminated	terminated	**166**
com-orkut	8	1715	terminated	terminated	212
	15	1715	terminated	terminated	292
	20	1715	terminated	terminated	411
	25	1715	terminated	terminated	580
	40	1715	terminated	terminated	1233
	50	**1713**	terminated	terminated	**1742**
uk-2002	8	2073	terminated	terminated	137
	15	2074	terminated	terminated	201
	20	2073	terminated	terminated	260
	25	2074	terminated	terminated	311
	40	2074	terminated	terminated	670
	50	2074	terminated	terminated	944

Besides, we compare the results of the BDAC algorithm and DPColorPath with YACC, an adapted version of Turán-shadow designed for such scenarios (see Table 6.4). However, our inability to access the source code of YACC prevents a direct comparison of their execution times. Therefore, the assessment is limited to comparing the sampling results reported in the YACC paper (Jain and Tong 2022) across identical datasets. The results are consistent for relatively smaller datasets, so we compare these three algorithms on large datasets with k=20, 40. The results are obtained for 500K samples, as the YACC states its results under 500K samples.

Table 6.4. Comparison of BDAC, YACC, and DPColorPath regarding estimation results for k=20,40.

	k	BDAC-min	BDAC-max	YACC	DPColorPath
com-lj	20	5,28E+32	6,77E+32	5,49E+32	5,48E+32
	40	1,96E+53	6,13E+53	2,51E+53	2,51E+53
soc-LJ	20	1,05E+33	2,67E+33	1,31E+33	1,31E+33
	40	**3,50E+53**	5,10E+54	**2,51E+53**	**5,86E+53**
com-orkut	20	**2,24E+15**	1,04E+28	**3,38E+17**	**1,99E+17**
	40	**2,30E+09**	1,10E+42	**2,61E+13**	**1,16E+13**
uk-2002	20	1,07E+41	1,07E+41	1,07E+41	1,07E+41
	40	5,29E+70	5,29E+70	5,29E+70	5,29E+70

6.3. Experimental results of parallel BDAC

We compare the execution times of parallel version of BDAC with the latest parallel k-clique counting algorithm, DPColorPath. Table 6.5 shows the execution times of both the sequential and parallel versions of the BDAC and DPColorPath algorithms. It is also observed whether the parallel version improves on the sequential algorithm, especially as the dataset increases. The results of both algorithms were obtained using 20 threads.

Table 6.5. Execution time comparison of both sequential and parallel versions of BDAC and DPColorPath.

	k	BDAC time (sec)	Parallel BDAC time (sec)	DPColorPath time (sec)	Parallel DPColorPath time (sec)
com-lj	20	68	38	31	3
	40	68	38	35	8
	50	68	38	39	8
soc-LJ	20	128	45	51	4
	40	128	45	96	30
	50	**128**	**45**	**166**	**80**
com-orkut	20	1715	218	411	30
	40	1715	218	1233	98
	50	**1715**	**218**	**1742**	**324**
uk-2002	20	2073	297	656	17
	40	2073	**297**	694	**29984**
	50	2073	**297**	665	**29710**

6.4. Discussion of results

This section discusses the results of our algorithm compared to state-of-the-art algorithms, the performance of BDAC, and its limitations.

Comparison BDAC with the other algorithms: Turán-shadow is suitable for relatively small datasets since its effectiveness significantly diminishes when dealing with larger datasets. The BDAC algorithm's capacity to address dense subgraphs provides a distinct advantage in specific contexts The BDAC consistently demonstrates a notably wider gap between the minimum and maximum values in certain cases when compared to the Tur'an-shadow algorithm (refer to Table 6.2). Its robust capability to handle large dense subgraphs exceeds that of both the Pivoter and Tur'an-shadow algorithms. It is capable of

handling large dense subgraphs, which goes beyond what the Pivoter and Tur'an-shadow algorithms can do. The scalability of Pivoter hinders its applicability to larger datasets. As a result, the need for exact values in larger datasets poses challenges in conducting a comparative evaluation.

Mostly, the execution time of the DPColorPath algorithm outperforms the BDAC, especially for larger datasets and larger k; it requires a larger sampling size. Also, it is observed that the execution time of the DPColorPath algorithm grows with the k on large datasets and gets closer to the BDAC execution time. However, the BDAC algorithm has approximately the same running time on a dataset for all k values because it does not build a recursion tree or apply sampling strategies.

Table 6.4 shows that the YACC and DPColorPath results on "soc-LJ" for k=40 and "com-orkut" for k=20 and 40 are inconsistent. Without knowing the exact values, we cannot determine which algorithm's sampling results are more accurate or which algorithm requires more samples. Our algorithm provides minimum and maximum k-clique counts in such cases, offering guarantees based on theoretical foundations.

Based on the dataset results, the exact values obtained do not surpass our estimated maximum value on any dataset. However, for some datasets, there is a significant difference between the BDAC's minimum and maximum k-clique counts. The limitation part below explains this variance.

Comparison of parallel BDAC with parallel DPColorPath: We compare the parallel version of BDAC with the DPColorPath algorithm, the state-of-the-art parallel k-clique counting algorithm, and their sequential versions. The results show that the parallel version of BDAC performs notable enhancements compared to the sequential version. The parallel BDAC shows significant speed up and outperforms its sequential versions for all k values. The parallel version is also similar to the sequential version independent of k values, showing constant execution time across different values of k. The DPColorPath algorithm improves its sequential versions but struggles with large, dense datasets "uk-2002" and larger $k > 40$ values. This situation shows that the parallelization might be inefficient or poorly scaled. If we compare the parallel versions of both BDAC and DPColorPath, the DPColorPath outperforms most cases of BDAC, but both versions of DPColorPath struggle with large datasets and larger k values. The parallel BDAC exhibits better scalability and overall performance for large, dense datasets and larger k values.

Performance of BDAC: Applying theorems in the algorithmic process lets us determine the dataset's minimum and maximum k-clique counts per vertex and global. This approach ensures rigorous analysis and guarantees that the generated k-cliques adhere to established mathematical principles. Additionally, understanding the range between the minimum and maximum values offers insights into the diversity and distribution of k-cliques, facilitating informed decision-making in data analysis and interpretation.

The execution time of the sampling-based algorithm mostly outperforms the BDAC. However, their accuracy depends on the sample sizes, and the complexity of these algorithms grows with k. The BDAC algorithm complexity is not dependent on k; it depends on the degeneracy α of the graph. The BDAC algorithm is well-suited for large and densely connected datasets such as com-lj, soc-pokec, soc-LJ, and uk-2002. In these datasets, the node densities typically meet the given threshold, allowing us to obtain both the minimum and maximum k-clique counts per node. This leads to a smaller difference between the lowest and highest values overall. This work represents the first attempt to provide lower and upper bounds and results for $k = 50$. Our algorithm is not restriced with $k < 50$; it can work for any $k > 3$.

Limitation of BDAC: For some datasets, there is a significant disparity between the estimated minimum and maximum k-clique compared to other datasets. This discrepancy highlights a limitation of our algorithm. Specifically, accurate estimation of the minimum k-cliques size becomes problematic when the edge density of node neighbors can be at most the threshold. This factor also affects the final estimation of k-cliques. Additionally, if a node has a high degree but its density is lower than the given threshold, it indicates that the node yields a sparse induced subgraph. This can lead to a significantly higher potential maximum k-clique count, particularly for large n but lower k, because of the binomial coefficients of $\binom{n}{k}$ used in calculating the maximum value. This observation emphasizes a crucial aspect of our algorithm's performance and offers valuable insights into its limitations. In cases where there is a substantial variance in estimates.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this work, we analyze graphlet counting algorithms, highlighting the problems' associated challenges and discussing the advantages and disadvantages of both exact and approximate approaches for handling large datasets. As part of the thesis, we conduct an in-depth analysis of graphlet counting techniques and algorithms. The goal is understanding how handling large datasets becomes impractical for larger graphlet sizes. Consequently, the focus shifts from exact methods to approximate solutions, given that exact algorithms are not feasible for large datasets, and there is a lack of research on counting graphlets of size $k > 5$.

We try to find potential research direction and concentrate on cliques, the most widely studied graphlet type across various disciplines. Within the scope of this thesis, we thoroughly survey clique counting methods, analyzing their strengths and weaknesses to offer insights into existing approaches and guide future research directions. This work is published as part of our contribution to the field (Çalmaz and Bostanoğlu 2024b).

Then, after the comprehensive literature analysis, we introduce BDAC using extremal graph theorems and propose a boundary-based method for k-clique approximation (Çalmaz and Bostanoğlu 2024a). The goal is to improve graph analysis capabilities and provide an efficient tool applicable to various fields such as social network analysis, recommender systems, and bioinformatics.

The BDAC algorithm efficiently approximates k-cliques for $k \leqslant 50$, providing minimum and maximum bounds locally (per vertex) and globally across various values of k, mainly focusing on $k = 8$, 15, 25, 40, and 50. The aim of providing results on different k values is to measure the capability of the algorithms from lower to large k values. Using extremal graph theorems, which provide insights about clique existence, in the algorithmic process, we can determine the boundaries of k-cliques for per-vertex and within the entire datasets. Our work is the first attempt to provide both lower and upper bounds and results for k=50, contributing to the advancement of k-clique counting algorithms. BDAC is unique in handling large dense subgraphs and offers reliable bounds for larger k value compared to other algorithms like Turán-shadow, YACC, and DPColorPath despite some

limitations in datasets like "com-orkut." This method advances k-clique counting without relying on sampling or recursion, also the complexity remains unaffected by values of k unlike other algorithms, enhancing the analysis of complex networks.

Comparison with state-of-the-art algorithms, including Turán-shadow, Pivoter, YACC, and DPColorPath, reveals distinctive characteristics of BDAC. When compared to Turán-shadow, its ability to handle large dense subgraphs offers a unique advantage, addressing a crucial limitation of existing algorithms and underscoring the potential utility of BDAC in specific contexts. Similarly, compared to YACC, our BDAC algorithm demonstrates competitive performance, delivering dependable estimations across datasets by offering lower and upper bounds. The DPColorPath algorithm mostly outperforms the BDAC regarding execution time, but it requires a much larger sample size for large datasets and larger k. In such a situation, the execution time of the BDAC and DPColorPath algorithm becomes competitive, as is shown in the experimental results.

We also improve the performance and scalability of BDAC through parallelization techniques. The results demonstrate that the parallel version outperforms the sequential BDAC for all values of k, making BDAC scalable for large datasets. The parallel BDAC also outperforms the state-of-the-art algorithm in literature on large and dense datasets and larger k values.

However, the BDAC exhibits limitations, particularly in datasets like "com-orkut." The significant disparity between estimated minimum and maximum k-cliques highlights challenges in accurately estimating minimum k-clique sizes, especially when the edge density of node neighbors falls below a Turán threshold.

In future work, we plan to prepare a conference paper focusing on the parallel version of BDAC. Furthermore, we intend to leverage datasets that estimate minimum and maximum counts of k-cliques to train a supervised machine learning model. This model will enable us to predict minimum values in cases where they are currently unknown, thereby enhancing the accuracy and comprehensiveness of our estimation methodology. We will try to integrate clique counting algorithms with Graph Neural Networks (GNNs). GNNs may predict or enhance clique counting in large-scale graphs by incorporating node embeddings and graph features learned during training.

In summary, we present a direct method for estimating k-cliques (where $k \leq 50$) without reliance on sampling techniques or the construction of recursion trees. Using

established theorems, the BDAC provides upper and lower bounds for k-cliques per vertex and globally, providing a reliable and efficient alternative to traditional methods. This advancement significantly enhances the accuracy and speed of analyzing complex networks and graph structures. We also improve the algorithm performance and scalability for large datasets using parallelization techniques.

BIBLIOGRAPHY

Abou Jamra, Hiba, Marinette Savonnet, and Éric Leclercq. 2021. "Detection of Event Precursors in Social Networks: A Graphlet-Based Method." In *International Conference on Research Challenges in Information Science,* 205–220. Springer.

Acosta-Mendoza, Niusvel, Andrés Gago-Alonso, and José E Medina-Pagola. 2012. "Frequent approximate subgraphs as features for graph-based image classification." *Knowledge-Based Systems* 27:381–392.

Ahmed, Nesreen K, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. "Efficient graphlet counting for large networks." In *2015 IEEE international conference on data mining,* 1–10. IEEE.

Akkoyunlu, Eralp Abdurrahim. 1973. "The enumeration of maximal cliques of large graphs." *SIAM Journal on Computing* 2 (1): 1–6.

Almasri, Mohammad, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. 2022. "Parallel k-clique counting on gpus." In *Proceedings of the 36th ACM International Conference on Supercomputing,* 1–14.

Barenboim, Leonid, and Michael Elkin. 2008. "Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition." In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing,* 25–34.

Batson, Joshua, Daniel A Spielman, Nikhil Srivastava, and Shang-Hua Teng. 2013. "Spectral sparsification of graphs: theory and algorithms." *Communications of the ACM* 56 (8): 87–94.

Becchetti, Luca, Carlos Castillo, Debora Donato, Stefano Leonardi, and Ricardo A. Baeza-Yates. 2006. "Link-Based Characterization and Detection of Web Spam." In *Adversarial Information Retrieval on the Web.* https://api.semanticscholar.org/CorpusID:208960374.

Betzler, Nadja, Rene van Bevern, Michael R. Fellows, Christian Komusiewicz, and Rolf Niedermeier. 2011. “Parameterized Algorithmics for Finding Connected Motifs in Biological Networks.” *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8 (5): 1296–1308. https://doi.org/10.1109/TCBB.2011.19.

Bhuiyan, Mansurul A., Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. “GUISE: Uniform Sampling of Graphlets for Large Graph Analysis.” In *2012 IEEE 12th International Conference on Data Mining,* 91–100. https://doi.org/10.1109/ICDM.2012.87.

Borthakur, Dhruba. 2007. “The hadoop distributed file system: Architecture and design.” *Hadoop Project Website* 11 (2007): 21.

Bron, Coen, and Joep Kerbosch. 1973. “Algorithm 457: finding all cliques of an undirected graph.” *Communications of the ACM* 16 (9): 575–577. https://doi.org/10.1145/362342.362367.

Buehrer, Gregory, and Kumar Chellapilla. 2008. “A scalable pattern mining approach to web graph compression with communities.” In *Proceedings of the 2008 international conference on web search and data mining,* 95–106. https://doi.org/10.1145/1341531.1341547.

Çalmaz, Büşra, and Belgin Ergenç Bostanoğlu. 2024a. “BDAC: Boundary-Driven Approximations of K-Cliques.” *Symmetry* 16 (8). ISSN: 2073-8994. https://doi.org/10.3390/sym16080983. https://www.mdpi.com/2073-8994/16/8/983.

———. 2024b. “k-Clique counting on large scale-graphs: a survey.” *PeerJ Computer Science* 10:e2501.

Chakraborty, Anwesha, Trina Dutta, Sushmita Mondal, and Asoke Nath. 2018. “Application of graph theory in social media.” *International Journal of Computer Sciences and Engineering* 6 (10): 722–729.

Chang, Lijun, Rashmika Gamage, and Jeffrey Xu Yu. 2024. “Efficient k-Clique Count Estimation with Accuracy Guarantee.” *Proceedings of the VLDB Endowment* 17 (11): 3707–3719.

Chen, Bolin, Jinhong Shi, Shenggui Zhang, and Fang-Xiang Wu. 2013. “Identifying protein complexes in protein–protein interaction networks by using clique seeds and graph entropy.” *Proteomics* 13 (2): 269–277.

Chiba, Norishige, and Takao Nishizeki. 1985. “Arboricity and subgraph listing algorithms.” *SIAM Journal on computing* 14 (1): 210–223. https://doi.org/10.1137/0214017.

Cleemput, Katrien Van. 2012. “FRIENDSHIP TYPE, CLIQUE FORMATION AND THE EVERYDAY USE OF COMMUNICATION TECHNOLOGIES IN A PEER GROUP.” *Information, Communication & Society* 15 (8): 1258–1277. https://doi.org/10.1080/1369118X.2011.606327. eprint: https://doi.org/10.1080/1369118X.2011.606327. https://doi.org/10.1080/1369118X.2011.606327.

Dagum, Leonardo, and Ramesh Menon. 1998. “OpenMP: An Industry-Standard API for Shared-Memory Programming.” *IEEE Comput. Sci. Eng.* (Washington, DC, USA) 5, no. 1 (January): 46–55. ISSN: 1070-9924. https://doi.org/10.1109/99.660313. https://doi.org/10.1109/99.660313.

Danisch, Maximilien, Oana Balalau, and Mauro Sozio. 2018. “Listing k-cliques in sparse real-world graphs.” In *Proceedings of the 2018 World Wide Web Conference,* 589–598. https://doi.org/10.1145/3178876.3186125.

Dasari, Naga Shailaja, Desh Ranjan, and Zubair Mohammad. 2014. “Maximal clique enumeration for large graphs on hadoop framework.” In *Proceedings of the first workshop on Parallel programming for analytics applications,* 21–30.

Dave, Vachik S, Nesreen K Ahmed, and Mohammad Al Hasan. 2017. “E-CLoG: Counting edge-centric local graphlets.” In *2017 IEEE International Conference on Big Data (Big Data),* 586–595. IEEE.

Dean, Jeffrey, and Sanjay Ghemawat. 2004. “MapReduce: Simplified data processing on large clusters.”

Dourisboure, Yon, Filippo Geraci, and Marco Pellegrini. 2009. “Extraction and classification of dense implicit communities in the web graph.” *ACM Transactions on the Web (TWEB)* 3 (2): 1–36.

Duan, Dongsheng, Yuhua Li, Ruixuan Li, and Zhengding Lu. 2012. “Incremental K-clique clustering in dynamic social networks.” *Artificial Intelligence Review* 38 (2): 129–147. https://doi.org/10.1007/s10462-011-9250-x.

Eden, Talya, Amit Levi, Dana Ron, and C Seshadhri. 2017. “Approximately counting triangles in sublinear time.” *SIAM Journal on Computing* 46 (5): 1603–1646.

Eden, Talya, Dana Ron, and C Seshadhri. 2018. “On approximating the number of k-cliques in sublinear time.” In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing,* 722–734.

Elenberg, Ethan R, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G Dimakis. 2016. “Distributed estimation of graph 4-profiles.” In *Proceedings of the 25th International Conference on World Wide Web,* 483–493.

Eppstein, David, Maarten Löffler, and Darren Strash. 2010. “Listing all maximal cliques in sparse graphs in near-optimal time.” In *Algorithms and Computation: 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I 21,* 403–414. Springer.

Erdős, Pál. 1969. “On the number of complete subgraphs and circuits contained in graphs.” *Časopis pro pěstování matematiky* 94 (3): 290–296.

Faust, Katherine. 2010. “A puzzle concerning triads in social networks: Graph constraints and the triad census.” *Social Networks* 32 (3): 221–233. https://doi.org/10.1016/j.socnet.2010.03.004.

Finocchi, Irene, Marco Finocchi, and Emanuele G Fusco. 2015. “Clique counting in mapreduce: Algorithms and experiments.” *Journal of Experimental Algorithmics (JEA)* 20:1–20. https://doi.org/10.1145/2794080.

Foucault Welles, Brooke, Anne Van Devender, and Noshir Contractor. 2010. "Is a friend a friend? investigating the structure of friendship networks in virtual worlds." In *CHI '10 Extended Abstracts on Human Factors in Computing Systems,* 4027–4032. CHI EA '10. Atlanta, Georgia, USA: Association for Computing Machinery. ISBN: 9781605589305. https://doi.org/10.1145/1753846.1754097. https://doi.org/10.1145/1753846.1754097.

Garey, Michael R, David S Johnson, and Larry Stockmeyer. 1974. "Some simplified NP-complete problems." In *Proceedings of the sixth annual ACM symposium on Theory of computing,* 47–63.

Gibson, David, Ravi Kumar, and Andrew Tomkins. 2005. "Discovering large dense subgraphs in massive graphs." In *Proceedings of the 31st international conference on Very large data bases,* 721–732.

Gonen, Mira, and Yuval Shavitt. 2009. "Approximating the number of network motifs." *Internet Mathematics* 6 (3): 349–372.

Gong, Yu, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, and Kenny Q Zhu. 2019. "Exact-k recommendation via maximal clique optimization." In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining,* 617–626.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* Http://www.deeplearningbook.org. MIT Press.

Goodrich, Michael T, and Paweł Pszona. 2011. "External-memory network analysis algorithms for naturally sparse graphs." In *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19,* 664–676. Springer.

Granovetter, Mark. 1983. "The strength of weak ties: A network theory revisited." *Sociological theory,* 201–233.

Gregori, Enrico, Luciano Lenzini, and Simone Mainardi. 2012. “Parallel k-clique community detection on large-scale networks.” *IEEE Transactions on Parallel and Distributed Systems* 24 (8): 1651–1660.

Hamann, Michael, Gerd Lindner, Henning Meyerhenke, Christian L Staudt, and Dorothea Wagner. 2016. “Structure-preserving sparsification methods for social networks.” *Social Network Analysis and Mining* 6 (1): 1–22.

Han, Jiawei, J Pei, and M Kamber. 2006. “Graph mining, social network analysis, and multirelational data mining.” *Data Mining: Concepts and Techniques,* 535–589.

Hasenplaugh, William, Tim Kaler, Tao B Schardl, and Charles E Leiserson. 2014. “Ordering heuristics for parallel graph coloring.” In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures,* 166–177.

Hočevar, Tomaž, and Janez Demšar. 2014. “A combinatorial approach to graphlet counting.” *Bioinformatics* 30 (4): 559–565.

Holland, Paul W., and Samuel Leinhardt. 1977. “A Method for Detecting Structure in Sociometric Data.” In *Social Networks,* edited by Samuel Leinhardt, 411–432. Academic Press. ISBN: 978-0-12-442450-0. https://doi.org/https://doi.org/10.1016/B978-0-12-442450-0.50028-6. https://www.sciencedirect.com/science/article/pii/B9780124424500500286.

Hormozdiari, Fereydoun, Petra Berenbrink, Nataša Pržulj, and S Cenk Sahinalp. 2007. “Not all scale-free networks are born equal: the role of the seed graph in PPI network evolution.” *PLoS Comput Biol* 3 (7): e118.

Huang, Hao, Nati Linial, Humberto Naves, Yuval Peled, and Benny Sudakov. 2016. “On the densities of cliques and independent sets in graphs.” *Combinatorica* 36 (5): 493–512. https://doi.org/10.1007/s00493-015-3051-9.

Huber, Wolfgang, Vincent J Carey, Li Long, Seth Falcon, and Robert Gentleman. 2007. “Graphs in molecular biology.” *BMC bioinformatics* 8 (6): 1–14.

Iasemidis, Leonidas D, Deng-Shan Shiau, Wanpracha Chaovalitwongse, J Chris Sackellares, Panos M Pardalos, Jose C Principe, Paul R Carney, Awadhesh Prasad, Balaji Veeramani, and Kostas Tsakalis. 2003. "Adaptive epileptic seizure prediction system." *IEEE transactions on biomedical engineering* 50 (5): 616–627. https://doi.org/10.1109/TBME.2003.810689.

Jackson, Matthew O., Tomas Rodriguez-Barraquer, and Xu Tan. 2012. "Social Capital and Social Quilts: Network Patterns of Favor Exchange." *American Economic Review* 102, no. 5 (August): 1857–97. https://doi.org/10.1257/aer.102.5.1857. https://www.aeaweb.org/articles?id=10.1257/aer.102.5.1857.

Jahnke, Marko, Christian Thul, and Peter Martini. 2007. "Graph based metrics for intrusion response measures in computer networks." In *32nd IEEE Conference on Local Computer Networks (LCN 2007),* 1035–1042. IEEE.

Jain, Shweta, and C Seshadhri. 2020. "The power of pivoting for exact clique counting." In *Proceedings of the 13th International Conference on Web Search and Data Mining,* 268–276. https://doi.org/10.1145/3336191.3371839.

———. 2017. "A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem." In *Proceedings of the 26th International Conference on World Wide Web,* 441–449. WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee. ISBN: 9781450349130. https://doi.org/10.1145/3038912.3052636. https://doi.org/10.1145/3038912.3052636.

Jain, Shweta, and Hanghang Tong. 2022. "YACC: A Framework Generalizing Turán-Shadow for Counting Large Cliques." In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM),* 684–692. SIAM.

Jayanthi, SK. 2012. "Clique-attacks detection in web search engine for spamdexing using k-clique percolation technique." *International Journal of Machine Learning and Computing* 2 (5): 648.

Jha, Kanchan, Sriparna Saha, and Hiteshi Singh. 2022. "Prediction of protein–protein interaction using graph neural networks." *Scientific Reports* 12 (1): 8360.

Jha, Madhav, C Seshadhri, and Ali Pinar. 2015. “Path sampling: A fast and provable method for estimating 4-vertex subgraph counts.” In *Proceedings of the 24th international conference on world wide web,* 495–505.

Johnston, HC. 1976. “Cliques of a graph-variations on the Bron-Kerbosch algorithm.” *International Journal of Computer & Information Sciences* 5 (3): 209–238.

Katona, G. 1987. “A Theorem of Finite Sets.” In *Classic Papers in Combinatorics,* edited by Ira Gessel and Gian-Carlo Rota, 381–401. Boston, MA: Birkhäuser Boston. ISBN: 978-0-8176-4842-8. https://doi.org/10.1007/978-0-8176-4842-8_27. https://doi.org/10.1007/978-0-8176-4842-8_27.

Kowalik, Łukasz. 2003. “Short cycles in planar graphs.” In *International Workshop on Graph-Theoretic Concepts in Computer Science,* 284–296. Springer.

Kruskal, Joseph B. 1963. “The number of simplices in a complex.” *Mathematical optimization techniques* 10:251–278.

Leon-Suematsu, Yutaka I, Kentaro Inui, Sadao Kurohashi, and Yutaka Kidawara. 2011. “Web spam detection by exploring densely connected subgraphs.” In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology,* 1:124–129. IEEE. https://doi.org/10.1109/WI-IAT.2011.152.

Leskovec, Jure, and Andrej Krevl. 2014. *SNAP Datasets: Stanford Large Network Dataset Collection.* http://snap.stanford.edu/data, June.

Li, Ronghua, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. “Ordering Heuristics for k-clique Listing.” *Proc. VLDB Endow.*

Li, Xiao-Li, Chuan-Sheng Foo, Soon-Heng Tan, and See-Kiong Ng. 2005. “Interaction graph mining for protein complexes using local clique merging.” *Genome Informatics* 16 (2): 260–269.

Li, Xiaodong, Tsz Nam Chan, Reynold Cheng, Caihua Shan, Chenhao Ma, and Kevin Chang. 2019. “Motif Paths: A New Approach for Analyzing Higher-order Semantics between Graph Nodes.” https://api.semanticscholar.org/CorpusID:204953830.

Low, Yucheng, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. 2014. "Graphlab: A new framework for parallel machine learning." *arXiv preprint arXiv:1408.2041*.

Lu, Zhenqi, Johan Wahlström, and Arye Nehorai. 2018. "Community detection in complex networks via clique conductance." *Scientific reports* 8 (1): 1–16. https://doi.org/10.1038/s41598-018-23932-z.

Mackay, David John Cameron. 1998. "Introduction to monte carlo methods." In *Learning in graphical models,* 175–204. Springer.

Makino, Kazuhisa, and Takeaki Uno. 2004. "New algorithms for enumerating all maximal cliques." In *Algorithm Theory-SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory, Humlebæk, Denmark, July 8-10, 2004. Proceedings 9,* 260–272. Springer.

Manoharan, Samuel, et al. 2020. "Patient diet recommendation system using K clique and deep learning classifiers." *Journal of Artificial Intelligence* 2 (02): 121–130.

Marcus, Dror, and Yuval Shavitt. 2012. "Rage–a rapid graphlet enumerator for large networks." *Computer Networks* 56 (2): 810–819.

McQuillan, J. M. 1977. "Graph theory applied to optimal connectivity in computer networks." *SIGCOMM Comput. Commun. Rev.* (New York, NY, USA) 7, no. 2 (April): 13–41. ISSN: 0146-4833. https://doi.org/10.1145/1024857.1024860. https://doi.org/10.1145/1024857.1024860.

Milenković, Tijana, and Nataša Pržulj. 2008. "Uncovering biological network function via graphlet degree signatures." *Cancer informatics* 6:CIN–S680.

Milik, Mariusz, Sándor Szalma, and Krzysztof A. Olszewski. 2003. "Common Structural Cliques: a tool for protein structure and function analysis." *Protein Engineering, Design and Selection* 16, no. 8 (August): 543–552. ISSN: 1741-0126. https://doi.org/10.1093/protein/gzg080. eprint: https://academic.oup.com/peds/article-pdf/16/8/543/4330604/gzg080.pdf. https://doi.org/10.1093/protein/gzg080.

Nedioui, Med Abdelhamid, Abdelouahab Moussaoui, Bilal Saoud, and Mohamed Chaouki Babahenini. 2020. "Detecting communities in social networks based on cliques." *Physica A: Statistical Mechanics and its Applications* 551:124100. ISSN: 0378-4371. https://doi.org/https://doi.org/10.1016/j.physa.2019.124100. https://www.sciencedirect.com/science/article/pii/S0378437119322642.

Nocaj, Arlind, Mark Ortmann, and Ulrik Brandes. 2015. "Untangling the hairballs of multi-centered, small-world online social media networks." *Journal of Graph Algorithms and Applications: JGAA* 19 (2): 595–618.

Pagh, Rasmus, and Charalampos E Tsourakakis. 2012. "Colorful triangle counting and a mapreduce implementation." *Information Processing Letters* 112 (7): 277–281.

Pan, Rusheng, Yunhai Wang, Jiashun Sun, Hongbo Liu, Ying Zhao, Jiazhi Xia, and Wei Chen. 2023. "Simplifying social networks via triangle-based cohesive subgraphs." *Visual Informatics* 7 (4): 84–94. https://doi.org/10.1016/j.visinf.2023.07.003.

Pashanasangi, Noujan, and C Seshadhri. 2020. "Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke." In *Proceedings of the 13th International Conference on Web Search and Data Mining,* 447–455.

Pinar, Ali, Comandur Seshadhri, and Vaidyanathan Vishal. 2017. "Escape: Efficiently counting all 5-vertex subgraphs." In *Proceedings of the 26th international conference on world wide web,* 1431–1440.

Prokopyev, Oleg A., Vladimir L. Boginski, Wanpracha Chaovalitwongse, Panos M. Pardalos, J. Chris Sackellares, and Paul R. Carney. 2007. "Network-Based Techniques in EEG Data Analysis and Epileptic Brain Modeling." In *Data Mining in Biomedicine,* edited by Panos M. Pardalos, Vladimir L. Boginski, and Alkis Vazacopoulos, 559–573. Boston, MA: Springer US. ISBN: 978-0-387-69319-4. https://doi.org/10.1007/978-0-387-69319-4_28. https://doi.org/10.1007/978-0-387-69319-4_28.

Pržulj, Natasa, Derek G Corneil, and Igor Jurisica. 2004. "Modeling interactome: scale-free or geometric?" *Bioinformatics* 20 (18): 3508–3515. https://doi.org/10.1093/bioinformatics/bth436.

Qiu, Xiafei, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. "Real-time constrained cycle detection in large dynamic graphs." *Proceedings of the VLDB Endowment* 11 (12): 1876–1888.

Rahman, Mahmudur, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. "Graft: An efficient graphlet counting method for large graph analysis." *IEEE Transactions on Knowledge and Data Engineering* 26 (10): 2466–2478.

Ralaivola, Liva, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. 2005. "Graph kernels for chemical informatics." *Neural networks* 18 (8): 1093–1110.

Reiher, Christian. 2016. "The clique density theorem." *Annals of Mathematics* 184 (3): 683–707. ISSN: 0003486X, accessed April 9, 2024. http://www.jstor.org/stable/44072027.

Ribeiro, Pedro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. 2021. "A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets." *ACM Computing Surveys (CSUR)* 54 (2): 1–36.

Rosselló, Francesc, and Gabriel Valiente. 2005a. "Chemical graphs, chemical reaction graphs, and chemical graph transformation." *Electronic Notes in Theoretical Computer Science* 127 (1): 157–166.

———. 2005b. "Graph transformation in molecular biology." In *Formal Methods in Software and Systems Modeling: Essays Dedicated to Hartmut Ehrig on the Occasion of His 60th Birthday,* 116–133. Springer.

Rossi, Ryan A., and Nesreen K. Ahmed. 2015. "The Network Data Repository with Interactive Graph Analytics and Visualization." In *AAAI.* https://networkrepository.com.

Saha, Barna, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. "Dense subgraphs with restrictions and applications to gene annotation graphs." In *Research in Computational Molecular Biology: 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings 14,* 456–472. Springer. https://doi.org/10.1007/978-3-642-12683-3_30.

Sanei-Mehri, Seyed-Vahid, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. "Butterfly counting in bipartite networks." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* 2150–2159.

Satuluri, Venu, Srinivasan Parthasarathy, and Yiye Ruan. 2011. "Local graph sparsification for scalable clustering." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data,* 721–732.

Schaeffer, Satu Elisa. 2007. "Graph clustering." *Computer science review* 1 (1): 27–64.

Seeland, Madeleine, Tobias Girschick, Fabian Buchwald, and Stefan Kramer. 2010. "Online structural graph clustering using frequent subgraph mining." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 213–228. Springer.

Seshadhri, C, Ali Pinar, and Tamara G Kolda. 2014. "Wedge sampling for computing clustering coefficients and triangle counts on large graphs." *Statistical Analysis and Data Mining: The ASA Data Science Journal* 7 (4): 294–307.

Seshadhri, C, and Srikanta Tirthapura. 2019. "Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial." In *Proceedings of the Web Conference (WWW),* 2:75.

Seshadhri, Comandur, Ali Pinar, and Tamara G Kolda. 2013. "Fast triangle counting through wedge sampling." In *Proceedings of the SIAM Conference on Data Mining,* 4:5. Citeseer.

Shi, Dinghua, Zhifeng Chen, Xiang Sun, Qinghua Chen, Chuang Ma, Yang Lou, and Guanrong Chen. 2021. "Computing cliques and cavities in networks." *Communications Physics* 4 (1): 249.

Shi, Jessica, Laxman Dhulipala, and Julian Shun. 2021. "Parallel clique counting and peeling algorithms." In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21),* 135–146. SIAM.

Shi, Jessica, Louisa Ruixue Huang, and Julian Shun. 2022. "Parallel Five-cycle Counting Algorithms." *ACM Journal of Experimental Algorithmics* 27:1–23.

Son, Seokshin, Ah Reum Kang, Hyun-chul Kim, Taekyoung Kwon, Juyong Park, and Huy Kang Kim. 2012. "Analysis of context dependence in social interaction networks of a massively multiplayer online role-playing game." *PloS one* 7 (4): e33918. https://doi.org/10.1371/journal.pone.0033918.

Spielman, Daniel A, and Nikhil Srivastava. 2008. "Graph sparsification by effective resistances." In *Proceedings of the fortieth annual ACM symposium on Theory of computing,* 563–568.

Spielman, Daniel A, and Shang-Hua Teng. 2011. "Spectral sparsification of graphs." *SIAM Journal on Computing* 40 (4): 981–1025.

Sporns, Olaf, and Rolf Kötter. 2004. "Motifs in brain networks." *PLoS biology* 2 (11): e369.

Tomita, Etsuji, Akira Tanaka, and Haruhisa Takahashi. 2006. "The worst-case time complexity for generating all maximal cliques and computational experiments." *Theoretical computer science* 363 (1): 28–42.

Tsourakakis, Charalampos. 2015. "The k-clique densest subgraph problem." In *Proceedings of the 24th international conference on world wide web,* 1122–1132.

Tsourakakis, Charalampos E, Petros Drineas, Eirinaios Michelakis, Ioannis Koutis, and Christos Faloutsos. 2011. "Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation." *Social Network Analysis and Mining* 1:75–81. https://doi.org/10.1007/s13278-010-0001-9.

Tsourakakis, Charalampos E, U Kang, Gary L Miller, and Christos Faloutsos. 2009. "Doulion: counting triangles in massive graphs with a coin." In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining,* 837–846.

Turán, Paul. 1941. "On an external problem in graph theory." *Mat. Fiz. Lapok* 48:436–452.

Vilakone, Phonexay, Doo-Soon Park, Khamphaphone Xinchang, and Fei Hao. 2018. "An efficient movie recommendation algorithm based on improved k-clique." *Human-centric Computing and Information Sciences* 8:1–15.

Vilakone, Phonexay, Khamphaphone Xinchang, and Doo-Soon Park. 2019. “Personalized movie recommendation system combining data mining with the k-clique method.” *Journal of Information Processing Systems* 15 (5): 1141–1155.

Vishwanathan, S Vichy N, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. “Graph kernels.” *The Journal of Machine Learning Research* 11:1201–1242.

Vitter, Jeffrey Scott. 1984. “Faster methods for random sampling.” *Communications of the ACM* 27 (7): 703–718.

Wang, Kaixin, Kaiqiang Yu, and Cheng Long. 2024. “Efficient k-Clique Listing: An Edge-Oriented Branching Strategy.” *Proceedings of the ACM on Management of Data* 2 (1): 1–26.

Yan, Hao, Qianzhen Zhang, Deming Mao, Ziyue Lu, Deke Guo, and Sheng Chen. 2021. “Anomaly detection of network streams via dense subgraph discovery.” In *2021 International Conference on Computer Communications and Networks (ICCCN),* 1–9. IEEE. https://doi.org/10.1109/ICCCN52240.2021.9522263.

Yang, Fang, Kunjie Fan, Dandan Song, and Huakang Lin. 2020. “Graph-based prediction of protein-protein interactions with attributed signed graph embedding.” *BMC bioinformatics* 21:1–16.

Yang, Xiaoyu, Yuefei Lyu, Tian Tian, Yifei Liu, Yudong Liu, and Xi Zhang. 2021. “Rumor detection on social media with graph structured adversarial learning.” In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence,* 1417–1423.

Ye, Xiaowei, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2023. “Efficient k-Clique Counting on Large Graphs: The Power of Color-Based Sampling Approaches.” *IEEE Transactions on Knowledge and Data Engineering*.

Yildiz, Hakan, and Christopher Kruegel. 2012. “Detecting social cliques for automated privacy control in online social networks.” In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops,* 353–359. https://doi.org/10.1109/PerComW.2012.6197509.

Yu, Haiyuan, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. “Predicting interactions in protein networks by completing defective cliques.” *Bioinformatics* 22, no. 7 (February): 823–829. ISSN: 1367-4803. https://doi.org/10.1093/bioinformatics/btl014. eprint: https://academic.oup.com/bioinformatics/article-pdf/22/7/823/48841534/bioinformatics_22_7_823.pdf. https://doi.org/10.1093/bioinformatics/btl014.

Yu, Jianke, Hanchen Wang, Xiaoyang Wang, Zhao Li, Lu Qin, Wenjie Zhang, Jian Liao, and Ying Zhang. 2023. “Group-based fraud detection network on e-commerce platforms.” In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining,* 5463–5475.

Yuan, Long, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. “Effective and efficient dynamic graph coloring.” *Proceedings of the VLDB Endowment* 11 (3): 338–351.

Yuan, Zhirong, You Peng, Peng Cheng, Li Han, Xuemin Lin, Lei Chen, and Wenjie Zhang. 2022. “Efficient $k-$clique Listing with Set Intersection Speedup.” In *2022 IEEE 38th International Conference on Data Engineering (ICDE),* 1955–1968. https://doi.org/10.1109/ICDE53745.2022.00192.

Zhao, Peixiang. 2015. “gSparsify: Graph motif based sparsification for graph clustering.” In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management,* 373–382.

Zhao, Zhao, Guanying Wang, Ali R Butt, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. 2012. “Sahad: Subgraph analysis in massive networks using hadoop.” In *2012 IEEE 26th International Parallel and Distributed Processing Symposium,* 390–401. IEEE.

Zykov, Alexander Aleksandrovich. 1949. “On some properties of linear complexes.” *Matematicheskii sbornik* 66 (2): 163–188.

VITA

BÜŞRA ÇALMAZ

Teaching and Professional Experience

- **Research and Teaching Assistant**, İzmir Institute of Technology, 2016–2024.

Higher Education

- **MSc in Computer Engineering**, İzmir Institute of Technology, 2018.
- **BSc in Computer Engineering**, Selçuk University, 2015.

Selected Publications

- Çalmaz B, Ergenç Bostanoğlu B. **K-clique counting on large scale-graphs: A Survey**. PeerJ Computer Science. 2024; 10:e2501. https://doi.org/10.7717/peerj-cs.2501.
- Çalmaz B, Bostanoğlu BE. **BDAC: Boundary-driven approximations of k-cliques**. Symmetry. 2024; 16(8): 983. https://doi.org/10.3390/sym16080983.
- Güvenoglu B, Bostanoglu B. **A qualitative survey on frequent subgraph mining**. Open Computer Science. 2018; 8(1): 194-209. https://doi.org/10.1515/comp-2018-0018.