2015

Mining dense substructures from large deterministic and probabilistic graphs

Arko Provo Mukherjee
Iowa State University

Mining dense substructures from large deterministic and probabilistic graphs

by

Arko Provo Mukherjee

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Iowa State University

Ames, Iowa

2015

DEDICATION

I would like to dedicate this thesis to my parents who have always supported me in all my decisions and encouraged me to follow my heart. Without their love, support and unfailing belief in me, I could not have been able to pursue my interests. I would also like to dedicate this thesis to my fiancée Sneha Aman Singh, who inspired me to take on this journey. It would have been hard to keep focus on work so far away from home without her constant help and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take a moment and thank all those whose help and support made this thesis possible. First, I would like to sincerely thank Prof. Srikanta Tirthapura for his guidance and mentorship throughout my tenure as a graduate student. I have learnt a lot from him in many ways. I am much thankful to him for all his patience in all these years as I learnt my way through all my mistakes. I would also like to thank all my committee members for their help and support. I would like to thank Prof. Suraj Kothari for many helpful discussions and feedback regarding my work. I would like to thank Prof. David Fernandez-Baca with whom I had several discussions on MapReduce and also for developing my interests in algorithm design through his course. I would like to thank Prof. Chaudhuri for her support and igniting my interest in distributed systems through her course that I took. I thank Prof. Jacobson for helping me to explore Computer Security and helping me through the classes that I have taken with him. I would like to thank all Professors whose classes I have enjoyed. I would also like to thank my collaborators, Michael Svendsen and Pan Xu. I have enjoyed many fruitful discussions with both of them.

Next, I would like to thank all the good people in Iowa State University for their help and support. I must thank all the staff members in the Department of Electrical and Computer Engineering as well as in Graduate College for always answering my queries and clearing all my doubts. Special thanks goes to Vicky Thorland-Oster for giving a patient hearing all the time.

I would like to thank all my friends and neighbors in Ames, IA. Without their company, life would have been hard away from home. I can easily predict that the last five years would contribute to many of my most memorable memories.

Last, but certainly not the least, I would like to thank all my teachers from South Point High School who shaped me in my formative years and helped me to develop my interest in science. I will be indebted to them all my life for teaching me to always think logically, keep an open mind and helping me to grow my “Courage to Know”!

ABSTRACT

Graphs represent relationships. Some relationships can be represented as a deterministic graph while others can only be represented by using probabilities. Mining dense structures from graphs help us to find useful patterns in these relationships having applications in wide areas like social network analysis, bioinformatics etc. Arguably the two most fundamental dense substructures are Maximal Cliques and Maximal Bicliques. The enumeration of both these structures are central to many data mining problems. With the advent of “big data”, real world graphs have become massive. Recently systems like MapReduce have evolved to process such large data. However using these systems to mine dense substrucures in massive graphs is an open question. In this thesis, we present novel parallel algorithms using MapReduce for the enumeration of Maximal Cliques / Bicliques in large graphs. We show that our algorithms are work optimal and load balanced. Further, we present a detailed evaluation which shows that the algorithm scales to large graphs with millions of edges and tens of millions of output structures. Finally we consider the problem of Maximal Clique Enumeration in an Uncertain Graph, which is a probability distribution on a set of deterministic graphs. We define the notion of a maximal clique for an uncertain graph, give matching upper and lower bounds on the number of such structures and present a near optimal algorithm to mine all maximal cliques.

DECLARATION

Publications The work presented in this thesis has been published in the following conference proceedings and journals:

- The content of Chapter 3 has been published in *Arko Provo Mukherjee and Srikanta Tirthapura, Enumerating maximal bicliques from a large graph using mapreduce, In Proceedings IEEE 3rd International Congress on Big Data (2014). For details see Mukherjee and Tirthapura (2014).*

- The content of Chapter 4 has been published in *Michael Svendsen, Arko Provo Mukherjee and Srikanta Tirthapura, Mining maximal cliques from a large graph using mapreduce: Tackling highly uneven subproblem sizes, Journal of Parallel and Distributed Computing, Special Issue: Scalable Systems for Big Data Management and Analytics (2014). For details see Svendsen et al. (2014).*

- The content of Chapter 5 has been published in *Arko Provo Mukherjee, Pan Xu and Srikanta Tirthapura, Mining maximal cliques from an uncertain graph, In Proceedings IEEE 31st International Conference on Data Engineering (2015). For details see Mukherjee et al. (2015).*

My contributions This thesis is a result of work done in collaboration with my advisor Prof. Srikanta Tirthapura as well as Michael Svendsen and Dr. Pan Xu. My contributions are as follows:

- Parallel Algorithm design to enumerate all maximal bicliques from a large graph. This was done using Hadoop. We show that the algorithms presented are work optimal and are scalable to a high degree of parallelism. The results were published in Mukherjee and Tirthapura (2014). I worked on the design, analysis, implementation as well as experiments on this project under the guidance of my advisor Prof. Tirthapura.

- Comparitive analysis of Algorithms for maximal clique enumeration with MapReduce. I helped in performing comparitive analysis of algorithm due to the masters thesis of Michael Svendsen (see Svendsen (2012)) with the previous state of the art for this problem. I also contributed in tuning the implemetations for faster runtime and participated in the discussion of this project. All results were published in Svendsen et al. (2014). **Note:** Parts of this Chapter is from Master Thesis Svendsen (2012) and hence there is significant overlap between the same. My contribution to this Chapter is primarily the comparitive analysis fo the Algorithms and the rest of the material is provided for the purpose of clarity.

- Defining the problem of maximal clique enumeration for an uncertain graph. The bound on the number of structures was done in collaboration with Dr. Pan Xu and Prof. Srikanta Tirthapura. The design and analysis of the Algorithm was done under the guidance of Prof. Tirthapura. I also implemented the Algorithm and performed experiments to validate our theoretical analysis. This work was published in Mukherjee et al. (2015).

CHAPTER 1. INTRODUCTION

1.1 Motivation

A graph (or network) is a natural abstraction to model rich relationships in data, and massive graphs are ubiquitous in applications such as online social networks Mislove et al. (2007); Newman et al. (2002), information retrieval from the web Broder et al. (2000), citation networks An et al. (2004), and physical simulation and modeling Wodo et al. (2012), to name a few. Finding information from such data can often be reduced to a problem of mining features from massive graphs.

Working with large graphs presents many challenges. A large graph does not always fit in the memory of a single computer. Hence such a graph must be stored in a distributed fashion among multiple nodes of a cluster. Further, since the entire graph is not available on a single node in the cluster, any algorithm that uses the graph as an input does not have the luxury of having the entire input on every node. Each computer has to work only with a subset of the entire graph to produce the results. This means that large graphs need to be processed on multiple processors in parallel. In many other cases, we can construct such a large graph only with certain probability as all information about the large cannot be obtained. An example of such a massive graph can be the telecommunication network, where we can assume a link (edge) only with a certain probability Ghosh et al. (2007).

Mining patterns from such large graph is an important topic in today. As mentioned above graphs represent many real life scenarios. Pattern mining in a graph help us to find interesting relationships in all the different scenarios that a graph can represent. Many a times finding dense regions in the network can help us find interesting relationships. Dense connections imply that those regions in the graph are closely connected to each other and hence form strong relationships. Thus mining dense substructures such as cliques, quasi-cliques, bicliques, quasi-bicliques etc. is an important area of study Alexe et al. (2004); Gibson et al. (2005); Abello et al. (2002); Sim et al. (2006). Many different dense substructures have been defined and studied Lee et al. (2010). We concentrate on arguably the two most fundamental dense substructures that can be found in a graph, maximal clique and maximal biclique and then focus on the problems of mining maximal cliques and maximal bicliques. We call these problems as the Maximal Clique Enumeration (MCE) and Maximal Biclique enumeration (MBE).

Many graph mining tasks have relied on enumerating maximal cliques and bicliques to identify significant substructures within the graph. For instance, maximal clique enumeration is used in clustering and community detection in social and biological networks Palla et al. (2005), in the study of the co-expression of genes under stress Rokhlenko et al. (2007), in integrating different types of genome mapping data Harley and Bonner (2001), and other applications in bio-informatics and data mining Chen and Crippen (2005); Grindley et al. (1993); Jonsson and Bates (2006); Mohseni-Zadeh et al. (2004); Zhang et al. (2008a); Hattori et al. (2003); Zaki et al. (1997). Similarly enumerating maximal bicliques has been used to identify significant substructures within the graph. For instance the analysis of web search queries Yi and Maghoul (2009) considered the "click-through" graph, where there are two types of vertices, web search queries and web pages. There is an edge from a search query to every page that a user has clicked in response to the search query. MBE was used in clustering queries using the click through graph. MBE has been used in social network analysis, in detection of communities in social networks Lehmann et al. (2008), and in finding antagonistic communities in trust-distrust networks Lo et al. (2011). It has also been applied in detecting communities in the web graph Kumar et al. (1999); Rome and Haralick (2005). In bioinformatics, MBE has been used widely e.g. in construction of the phylogenetic tree of life Driskell et al. (2004); Sanderson et al. (2003); Yan et al. (2005); Nagarajan and Kingsford (2008), structure discovery and analysis of protein-protein interaction networks Bu et al. (2003); Schweiger et al. (2011), analysis of gene-phenotype relationships Xiang et al. (2012), prediction of miRNA regulatory modules Yoon and Micheli (2005), modeling of hot spots at protein interfaces Li and Liu (2009), and in analysis of relationships between genotypes, lifestyles, and diseases Mushlin et al. (2007). Other applications include Learning Context Free Grammars Yoshinaka (2011), finding correlations in databases Jermaine (2005), for data compression Agarwal et al. (1994), role mining in role based access control Colantonio et al. (2010), and process operation scheduling Mouret et al. (2011).

While it is easy to find a single maximal clique / biclique in a graph, enumerating all maximal cliques / bicliques are an NP-hard problems Valiant (1979); Peeters (2003). This does not however mean that typical cases are unsolvable. In fact, there are output-polynomial time algorithms for both these problems, whose theoretical runtime is bounded by a polynomial in the number of vertices in the graph, and the number of structures that are output Tsukiyama et al. (1977); Alexe et al. (2004). Thus it is reasonable to expect to be able to devise algorithms for MBE that work on large graphs, as long as the number of maximal bicliques output is not too high.

However, current methods for enumerating cliques / bicliques have the following drawbacks. Most methods are sequential algorithms that are unable to make use of the power of multiple processors. For handling large graphs, it is imperative to have methods that can process a graph in parallel. Next, they have been evaluated only on small graphs of a few thousands of vertices and a few hundred thousand maximal bicliques, and have not been shown to scale to large graphs. For instance, the popular "consensus" method for biclique enumeration Alexe et al. (2004) presents experimental data only on graphs of up to 2,000 vertices, and about 140,000 maximal bicliques, and other works Li et al. (2007); Liu et al. (2006) are also similar. Further little is know as to how to enumerate these structures for an Uncertain Graph. In this work we propose to solve some of these fundamental problems for large deterministic and uncertain graphs.

1.1.1 Big Data Software Tools

Recently parallel programming platforms have evolved that simplifies parallel programming. In this section we discuss a few such parallel programming frameworks. Recently **MapReduce** Dean and Ghemawat (2004, 2008); Karloff et al. (2010) has become a popular platform for parallel programming. The MapReduce programming model was designed to address a common problem with parallel programming and algorithm design. It was observed by the authors of the model that many of the problems that need to be solved with a parallel algorithm due to the size of the input are actually simple. However, the design and implementation of the parallel solution can quickly become very complex when trying to write parallel programs that can run on many nodes. The MapReduce model was developed to address just that.

The model is inspired by the *map* and *reduce* primitives from functional programming languages like Lisp. The algorithm design must be split up into one / multiple rounds of MapReduce. Each round must have a *map* and a *reduce* method. Effectively the entire algorithm is broken down into a series of map and reduce methods. The input to the program is processed by the map method and it emits a sequence of key / value pairs $< K, V >$. Then the system groups the values according to the keys and sorts them. Thus for each key $k_i \subset K$, the system generates $< k_i, list(V_i) >$. This is then passed on to the reduce methods as their input which will then process them as per the algorithm. The output from the *reduce* methods can then be used as the input to the next round.

The reason for the success of MapReduce is its relative simplicity with respect to many other popular parallel programming platforms. All the designer needs to do is to think and implement the solution to the problem as a sequence of map and reduce primitives. The system automatically breaks up the input into slices and feeds them into the map method. It also takes care of interprocess communication, load balancing, fault tolerance and data locality. All these complexities are hidden from the programmer. The MapReduce framework uses an underlying distributed file system known as the Google File System Ghemawat et al. (2003) which takes care of distributing the data and maintaining multiple replica of the data to make the system more robust and fault tolerant. The other important benefit of the MapReduce model is that it can be implemented on relatively inexpensive commodity clusters made of low cost commodity hardware. MapReduce assumes serial communication network like the Ethernet and hence no sophisticated parallel communication network is required. This helps in lowering the hardware cost considerably.

The original MapReduce implementation by Google is not publicly available for use or distribution. Hence, for this work, we use the popular Open Source implementation of MapReduce known as Hadoop White (2009). Hadoop is a JAVA implementation of the MapReduce programming model and provides a JAVA API. It also provides bindings to different other programming languages like C++, Python etc. Additionally it replaces the Google File system with its own implementation of a Distributed File System called HDFS Shvachko et al. (2010).

All these benefits come with few drawbacks — firstly the user is forced to think in the restrictive model of Map and Reduce. Every algorithm may not be naturally expressible in terms of MapReduce. This makes it difficult to express all algorithms in terms of map and reduce methods. Secondly, it is a batch processing system with low response time. Also, it is hard to do dynamic load balancing of the input graph.

While we evaluated an implementation on top of MapReduce, the ideas in this work are more generally applicable and can easily be adapted to other frameworks such as Pregel Malewicz et al. (2010) and Spark Zaharia et al. (2012).

1.1.2 Uncertain Graph Model

Large datasets often contain information that is uncertain in nature. For example, given people A and B, a question of the form "does A know B" may not be definitively answered using available information. In such a case, it is a common solution to use probability to quantify our confidence in this relation, and say that the relation exists with a probability of p, for some value p determined from the available information. Such uncertain graphs have often been used in modeling real data, for example, various communication networks Ghosh et al. (2007); Biswas and Morris (2005); Kawahigashi et al. (2005), in social networks Adar and Re (2007); Guha et al. (2004); Kempe et al. (2003); Liben-Nowell and Kleinberg (2003); Kuter and Golbeck (2010), protecting privacy while modeling social networks Boldi et al. (2012), protein interaction networks Asthana et al. (2004); Bader et al. (2004); Rhodes et al. (2005), analyzing regulatory networks in biological systems Jiang et al. (2006) and mining information from biological databases Sevon et al. (2006). While the notion of a maximal cliques or bicliques are well understood in a deterministic graph, it is not well defined or understood within an uncertain graph.

1.2 Literature review

1.2.1 Maximal Cliques

We first discuss related work on sequential MCE and then on parallel MCE. An early work due to Bron and Kerbosch (1973) is an algorithm based on depth-first-search with good experimental performance on typical inputs, but whose worst case behavior is poor. Other algorithms stemming from this work include Tomita et al. (2006). Another branch of enumeration algorithms provide output sensitive runtime guarantees, i.e. the runtime is proportional to the size of the output. These algorithms stem from the Tsukiyama et al. (1977) algorithm, which has a running time of $O(|V||E|\mu)$, where μ is the number of maximal cliques. However, these output sensitive algorithms tend not to perform as well as the worst case optimal algorithms in practice Tomita et al. (2006); Eppstein et al. (2010).

Early works in the area of parallel MCE include Zhang et al. (2005) and Du et al. (2006). Zhang *et al.* developed an algorithm based on the Kose et al. (2001) algorithm. Since these algorithms are based on breadth first search, they are able to enumerate maximal cliques in increasing order of size, but this makes the memory requirements very large. Du et al. (2006), present a parallel algorithm based on the output-sensitive class of algorithms. However, as also noted by Schmidt et al. (2009), this algorithm suffers from poor load balance; the graphs addressed by these experiments are quite small, they have about 150,000 maximal cliques and a million edges.

Schmidt et al. (2009) identify load balancing as a significant issue in parallel MCE and present a parallel algorithm that uses "work stealing" to dynamically distribute load among processors. Their algorithm is designed for use with MPI, where the user can control the actions of a process and the manner of parallelism to a high degree of detail, when compared with MapReduce. In their algorithm, processes explore tasks in parallel until they run out of work, at which point idle processes request for more work from busy processes (work stealing). This continues until all processes are idle. Such types of work stealing and dynamic load balancing are expensive to implement in the MapReduce model, since the processes are synchronized at each stage of Map and Reduce – for instance, all mappers need to complete before reducers start processing data. Our algorithm also implements effective load balancing, but in a more pre-determined and static manner.

Wu et al. (2009) present an MCE algorithm designed for MapReduce. The algorithm splits the input graph into many subgraphs, which are then independently processed to enumerate cliques. However this work does not address load balancing, and in addition, their algorithm may enumerate non-maximal cliques, so that an additional post-processing step is needed to only emit maximal cliques.

dMaximalCliques Lu et al. (2010) is another parallel MCE algorithm, based on the sequential algorithm of Tomita et al. (2006). This algorithm works in two phases. The first phase enumerates maximal, duplicate, and non-maximal cliques, and the second post-processing phase removes duplicate and non-maximal cliques from the output. However, this post-processing phase can be very expensive since the output prior to filtering can become much larger than the final output; for instance, on the wikitalk-3 graph the first enumeration phase takes 7 minutes (on 20 processors), and the second post-processing phase takes 228 minutes (on 80 processors). The algorithm is implemented for the Sector/Sphere Gu and Grossman (2009) framework.

1.2.2 Maximal Bicliques

Some previous work related to MBE are as follows. Makino and Uno (2004) describes methods to enumerate all maximal bicliques in a *bipartite graph*, with the delay between outputting two bicliques bounded by a polynomial in the maximum degree of the graph. Zhang et al. (2008b) describe a branch-and-bound algorithm for the same problem. However, these approaches do not work for general graphs, as we consider here.

There is a variant of MBE where we only seek *induced* maximal bicliques in a graph. An induced maximal biclique is a maximal biclique which is also an induced subgraph; i.e. a maximal biclique $\langle L, R \rangle$ in graph G is an induced maximal biclique if L and R are themselves independent sets in G. We consider the *non-induced* version, where edges are allowed in the graph between two vertices that are both in L, or both in R (such edges are of course, not a part of the biclique). The set of maximal bicliques that we output will also contain the set of induced maximal bicliques, which can be obtained by post-processing the output of our algorithm. Note that for a bipartite graph, every maximal biclique is also an induced maximal biclique. Algorithms for Induced MBE include work by Eppstein (1994), Dias et al. (2005), and Gaspers et al. (2008).

Alexe et al. (2004) present an iterative algorithm for non-induced MBE using the "consensus" method. Another technique for MBE is based on a recursive depth first search (DFS) Li et al. (2007); Liu et al. (2006). Li et al. (2007) presents an approach based on a connection with mining closed patterns in a transactional database, and apply the algorithm from Uno et al. (2004), which is based on depth first search. Liu et al. (2006) present a more direct algorithm for biclique enumeration based on depth first search.

Another approach to MBE is through a reduction to the problem of enumerating maximal cliques, as described by Gély et al. (2009). Given a graph G on which we need to enumerate maximal bicliques, a new graph G' is derived such that through enumerating maximal cliques in G' using an algorithm such as Tomita et al. (2006); Tsukiyama et al. (1977), it is possible to derive the maximal bicliques in

G. However, this approach is not practical for large graphs since in going from G to G', the number of edges in the graph increases significantly.

To our knowledge, the only prior work on parallel algorithms for MBE is by Nataraj and Selvan (2009), who use the correspondence between maximal bicliques and closed patterns Li et al. (2007) to derive a parallel method for enumerating maximal bicliques. A significant issue is that Nataraj and Selvan (2009) assumes that the input graph is presented as an adjacency matrix, which is then converted into a transactional database and distributed among the processors. In contrast, we do not assume an adjacency matrix, but assume that the graph is presented as a list of edges. Thus we are able to work on much larger graphs than Nataraj and Selvan (2009); the largest graph that they consider has 500 vertices and about 9000 edges.

1.2.3 Uncertain Graphs

There has been recent work in the database community on various problem on uncertain graphs, including probability-threshold based shortest paths Yuan et al. (2010), nearest neighbors Potamias et al. (2010), enumerating frequent and reliable subgraphs Hintsanen and Toivonen (2008); Zou et al. (2010c); Jin et al. (2011a); Zou et al. (2010a); Liu et al. (2012); Kollios et al. (2013), distance-constrained reachability Jin et al. (2011b). Note that our problem is different from the problems mentioned above. For example, finding reliable subgraphs problem is to find all subgraphs of the uncertain graph such which are connected with a high probability. However, these individual subgraphs may be sparse. We are interested instead in finding dense subgraphs whose probability of existence is greater than a given threshold.

There is relatively little known about mining dense substructures from an uncertain graph. To our knowledge, the only previous work on mining maximal cliques in an uncertain graph is by Zou et al. (2010b). Our work is different from theirs in significant ways. Mainly, while we focus on enumerating all α-maximal cliques in a graph, they focus on a different problem, that of enumerating the k maximal cliques with the highest probability of existence. We present bounds on the number of such cliques that could exist, while by definition, their algorithm outputs k cliques.

1.3 Contributions

The following are the contributions of this work.

- We present a generic parallel solution for MCE / MBE using the MapReduce framework Dean and Ghemawat (2008). At a high level, our approach clusters the input graph into overlapping subgraphs that can be processed independently in parallel, by different reducers.
- Applying the generic framework to solve MBE using MapReduce
- Designing optimizations helping in reducing the overlap in the work done by different subtasks. It is usually not possible to assign disjoint subgraphs to different processors, and the subgraphs assigned to different tasks will overlap, sometimes significantly. Through a careful partitioning of the search space among the different tasks, we reduce redundant work among the tasks (this partitioning depends on details of the sequential algorithm used at each task).
- Designing load balancing strategies for Maximal Biclique problem. With a graph analysis task such as biclique enumeration, the complexity of different subgraphs varies significantly, depending on the density of edges in the subgraph. Naively done, this can lead to a case where most reducers finish quickly, while only a few take a long time, leading to a poor parallel performance. We present a solution to keep the load more balanced, based on an ordering of vertices, which reduces enumeration load on subgraphs that are dense, and increases the load on subgraphs that are sparse, leading to a better parallel efficiency.
- A direct parallelization of the consensus sequential algorithm Alexe et al. (2004), using an approach different from clustering. We found that while this approach may use a smaller memory per node that the clustering approach, it requires substantially greater runtime.
- An experimental evaluation of the state of the art Maximal Clique Enumeration Algorithm on top of MapReduce from Svendsen (2012) with previous work. The result of the evaluation clearly demonstrates that the Algorithm described in Svendsen (2012) outperforms the previous MCE Algorithm for MapReduce.

- We consider a basic question on maximal cliques within uncertain graphs: *how many α-maximal cliques can be present within an uncertain graph?* For the case of deterministic graphs, this question was first considered by Moon and Moser (1965) in 1965, who presented matching upper and lower bounds for the largest number of maximal cliques within a graph; on a graph with n vertices, the largest possible number of maximal cliques is $3^{\frac{n}{3}}$[1]. For the case of uncertain graphs, we present the first matching upper and lower bounds for the largest number of α-maximal cliques in a graph on n vertices. We show that for $0 < \alpha < 1$, the maximum number of α-maximal cliques in an uncertain graph is $\binom{n}{\lfloor n/2 \rfloor}$, i.e. there is an uncertain graph on n vertices with $\binom{n}{\lfloor n/2 \rfloor}$ uncertain maximal cliques and no uncertain graph on n vertices can have more than $\binom{n}{\lfloor n/2 \rfloor}$ α-maximal cliques.

- We present a branch-and-bound based algorithm for enumerating all α-maximal cliques within an uncertain graph. We prove that the algorithm enumerates every α-maximal clique in the graph and provide an upper bound on the runtime. Our experimental analysis on synthetic and real-world graphs show that the algorithm can enumerate maximal cliques in an uncertain graph of tens of thousands of vertices, over hundred thousand edges and over two million α-maximal cliques. Interestingly, the observed runtime of this algorithm is proportional to the size of the output, i.e. the number of α-maximal cliques that are present in the graph.

- Application of the generic framework to MCE and comparison with previous work.

- Experimental validations for all the contributions stated above.

[1]This assumes that 3 divides n. If not, the expressions are slightly different

CHAPTER 2. PRELIMINARIES

In this Chapter we discuss some definitions and observations that we use throughout this thesis. We consider a simple undirected graph $G = (V, E)$ without self-loops or multiple edges, where V is the set of all vertices and E is the set of all edges of the graph. Let $n = |V|$ and $m = |E|$. Graph $H = (V_1, E_1)$ is said to be a sub-graph of graph $G = (V, E)$ if $V_1 \subset V$ and $E_1 \subset E$. H is known as an induced subgraph if E_1 consists of all edges of G that connect two vertices in V_1. For vertex $u \in V$, let $\eta(u)$ denote the vertices adjacent to u. For a set of vertices $U \subseteq V$, let $\eta(U) = \bigcup\limits_{u \in U} \eta(u)$. For vertex $u \in V$ and $k > 0$, let $\eta^k(u)$ denote all vertices that can be reached from u in k hops. For $U \subseteq V$, let $\eta^k(U) = \bigcup\limits_{u \in U} \eta^k(u)$. We call $\eta^k(U)$ as the k-neighborhood of U. For a set of vertices $U \subseteq V$, let $\Gamma(U) = \bigcap\limits_{u \in U} \eta(u)$.

An uncertain graph is a probability distribution over a set of deterministic graphs. Formally, an uncertain graph is a triple $\mathcal{G} = (V, E, p)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $p : E \to (0, 1]$ is the probability function that assigns a probability $p(e)$ to each edge $e \in E$. As in prior work on uncertain graphs, we assume the probabilities assigned to different edges are mutually independent.

Definition 1. *A set of vertices $C \subseteq V$ is a clique in a graph $G = (V, E)$, if every pair of vertices in C are connected by an edge in E.*

Definition 2. *A set of vertices $M \subseteq V$ is a maximal clique in a Graph $G = (V, E)$, if (1) M is a clique in G and (2) There is no vertex $v \in V \setminus M$ such that $M \cup \{v\}$ is a clique in G.*

Definition 3. *A* biclique $B = \langle L, R \rangle$ *is a subgraph of G containing two non-empty and disjoint vertex sets, L and R such that for any two vertices $u \in L$ and $v \in R$, there is an edge $(u, v) \in E$.*

Note that the definition on $B = \langle L, R \rangle$ does not impose any restriction on the existence of edges among the vertices within R or within L, i.e., we consider *non-induced* bicliques.

Definition 4. *A biclique $M = \langle L, R\rangle$ in G is said to be a maximal biclique if there is no other biclique $M' = \langle L', R'\rangle \neq \langle L, R\rangle$ such that $L \subset L'$ and $R \subset R'$.*

The **Maximal Clique Enumeration Problem (MCE)** is to enumerate the set of all maximal cliques in graph $G = (V, E)$.

The **Maximal Biclique Enumeration Problem (MBE)** is to enumerate the set of all maximal bicliques in graph $G = (V, E)$.

Definition 5. *In an uncertain graph $\mathcal{G}$, for a set of vertices $C \subseteq V$, the clique probability of C, denoted by $clq(C, \mathcal{G})$, is defined as the probability that in a graph sampled from $\mathcal{G}$, C is a clique. For parameter $0 \leq \alpha \leq 1$, C is called an α-clique if $clq(C, \mathcal{G}) \geq \alpha$.*

For any set of vertices $C \subseteq V$, let E_C denote the set of edges $\{e = (u, v) | e \in E, u, v \in C$ and $u \neq v\}$, i.e. the set of edges if C were a clique.

Observation 1. *For any set of vertices $C \subseteq V$, $clq(C, \mathcal{G}) = \prod_{e \in E_C} p(e)$*

Proof. Let G be a graph sampled from $\mathcal{G}$. The set C will be a clique in G iff every edge in E_C is sampled into G. Since the probability of selecting different edges is independent, the observation follows. □

Definition 6. *Given an uncertain graph $\mathcal{G} = (V, E, p)$, and a parameter $0 \leq \alpha \leq 1$, a set $M \subseteq V$ is defined as an α-maximal clique if (1) M is an α-clique in $\mathcal{G}$, and (2) There is no vertex $v \in (V \setminus M)$ such that $M \cup \{v\}$ is an α-clique in $\mathcal{G}$.*

Definition 7. *The* **Maximal Clique Enumeration** *problem in an Uncertain Graph $\mathcal{G}$ is to enumerate all vertex sets $M \subseteq V$ such that M is an α-maximal clique in $\mathcal{G}$.*

The following two observations follow directly from Observation 1.

Observation 2. *For any two cliques A and B in $\mathcal{G}$, if $B \subset A$ then, $clq(B, \mathcal{G}) \geq clq(A, \mathcal{G})$.*

Observation 3. *Let C be an α-clique in $\mathcal{G}$. Then for all $e \in E_C$ we have $p(e) \geq \alpha$.*

From the above observation, it is clear that we can drop all edges e such that $p(e) < \alpha$, prior to enumerating α-maximal cliques in $\mathcal{G}$.

CHAPTER 3. MINING ALL MAXIMAL BICLIQUES

3.1 Sequential Algorithms

We describe the two general approaches to sequential algorithms for MBE that we consider, one based on depth first search (see Liu et al. (2006)) and another based on a "consensus algorithm" (see Alexe et al. (2004)).

3.1.0.1 Sequential DFS Algorithm

The basic sequential depth first approach (DFS) that we use is described in Algorithm 1, based on work by Liu et al. (2006). It attempts to expand an existing maximal biclique into a larger one by including additional vertices that qualify, and declares a biclique as maximal if it cannot be expanded any further. The algorithm takes the following inputs: (1) the graph $G = (V, E)$, (2) the current vertex set being processed, X, (3) T, the tail vertices of X, i.e. all vertices that come after X in lexicographical ordering and (4) s, the minimum size threshold below which a maximal biclique is not enumerated. s can be set to 1 so as to enumerate all maximal bicliques in the input graph. However, we can set s to a larger value to enumerate only large maximal bicliques such that for $B = \langle L, R \rangle$, we have $|L| \geq s$ and $|R| \geq s$. The size threshold s is provided as user input. The other inputs are initialized as follows: $X = \varnothing, T = V$.

The algorithm recursively searches for maximal bicliques. It increases the size of X by recursively adding vertices from the tail set T, and pruning away those vertices from T which cannot be added to X to expand the biclique. From the expanded X, the algorithm outputs the maximal biclique $\langle \Gamma(\Gamma(X)), \Gamma(X) \rangle$.

Algorithm 1: MineLMBC(G,X,T,s)

```
forall the vertex $v \in T$ do
    if $|\Gamma(X \cup \{v\})| < s$ then
        $T \leftarrow T \setminus \{v\}$ ;
if $|X| + |T| < s$ then
    return
Sort vertices in $T$ as per ascending order of $|\eta(X \cup \{v\})|$ ;
forall the vertex $v \in T$ do
    $T \leftarrow T \setminus \{v\}$ ;
    if $|X \cup \{v\}| + |T| \geq s$ then
        $N \leftarrow \Gamma(X \cup \{v\})$ ;
        $Y \leftarrow \Gamma(N)$ ;
        Biclique $B \leftarrow \langle Y, N \rangle$ ;
        if $(Y \setminus (X \cup \{v\})) \subseteq T$ then
            if $|Y| \geq s$ then
                Emit $B$ as a maximal biclique ;
            MineLMBC($G$, $Y$, $T \setminus Y$, $s$) ;
```

3.1.0.2 Consensus Algorithm

Alexe et al. (2004) present an iterative approach to MBE. This algorithm starts off with a set of simple "seed" bicliques. In each iteration, it performs a "consensus" operation, which involves performing a cross-product on the set of current candidates bicliques with the seed bicliques, to generate a new set of candidates, and the process continues until the set of candidates does not change anymore. After each stage, the newly found bicliques can be expanded to find new maximal bicliques. After each step, the duplicate maximal bicliques can be dropped. It is proved that these algorithms exactly enumerate the set of maximal bicliques in the input graph. Algorithm 2 shows the sequential consensus Algorithm. For further details, we refer the reader to Alexe et al. (2004).

The consensus approach has a good theoretical guarantee, since its runtime depends on the number of maximal cliques that are output. In particular, the runtime of the MICA version of the algorithm is proved to be bounded by $O\left(n^3 \cdot N\right)$ where n is the number of vertices and N total number of maximal bicliques in G. The consensus algorithm has been found to be adequate for many applications and is quite popular.

Algorithm 2: Sequential Consensus Algorithm

```
1  Load Graph $G = (V, E)$ ;
2  $R \leftarrow$ Collection of all Stars in $G$ // Biclique formed by a vertex and its
       neighbors
3  $S \leftarrow \varnothing$ ;
4  forall the $b \in R$ do
5      $m \leftarrow$ Extend $b$ ;
6      $S \leftarrow S \cup m$ ;
7  $O \leftarrow S$; // Add seed set to the output
8  $P \leftarrow S$; // Initialize set PREV with SEED
9  repeat
10     $T \leftarrow$ Consensus between all maximal bicliques in $S$ and $P$ ;
11     $C \leftarrow \varnothing$ ;
12     forall the $b \in T$ do
13         $m \leftarrow$ Extend biclique $b$ ;
14         if $m$ is not a duplicate then
15             $C \leftarrow C \cup m$ ;
16     $O \leftarrow O \cup C$ ;
17     $P \leftarrow C$ ;
18 until $N$ is Empty;
```

3.2 Parallel Algorithms for MBE

We describe our parallel algorithms for MBE, and give an outline of how these are implemented using MapReduce. We first present a basic clustering approach, *which can be used with any sequential algorithm for MBE*, followed by enhancements to the basic clustering approach.

3.2.1 Basic Clustering Approach

For each $v \in V$, let subgraph (cluster) $C(v)$ be defined as the induced subgraph on all vertices in $\eta^2(v)$ (the 2-neighborhood of v in G). We first note the following.

Lemma 1. *Each maximal biclique B in $G = (V, E)$ is a maximal biclique in $C(v)$ for every vertex v in B. Further, for any $v \in V$, each maximal biclique in $C(v)$ is also a maximal biclique in G.*

Proof. We first prove the first direction, i.e. each maximal biclique B in $G = (V, E)$ is a maximal biclique in $C(v)$ for every vertex v that is contained in B. Consider a maximal biclique $M = \langle L, R \rangle$ in G. Let v be a vertex in M and without loss of generality suppose that $v \in L$. By the definition of a biclique, for each $u \in R$, u is a neighbor of v. Similarly, every vertex $w \in L$ is a neighbor of v, and is hence in $\eta^2(v)$. Hence M is completely contained in $C(v)$. Note that M is also a maximal biclique in $C(v)$. To see this, note that if M is not maximal biclique in $C(v)$, then M is not maximal in G either.

Next we prove the other direction, that is we prove that every maximal biclique in each cluster $C(v)$ is also a maximal biclique in G. Consider a maximal biclique $M = \langle L, R \rangle$ contained in cluster $C(v)$. We prove by contradiction. We assume M is non–maximal in G. This implies that there exists a maximal biclique $M' = \langle L', R' \rangle$ in G such that M is completely contained in M'. Let $\hat{M} = \{L \cup R\}$ and $\hat{M}' = \{L' \cup R'\}$. Let us consider a vertex $a \in \{\hat{M}' \setminus \hat{M}\}$. As per our assumption, $a \notin C(v)$. Also, without loss in generality, assume $v \in L$. There are two cases that are possible. Either $a \in R$ or $a \in L$. Let us first consider the first case, i.e. vertex $a \in R$. Now vertex a extends $M = \langle L, R \rangle$ by adding vertex a to R. This means a is connected to each vertex $u \in L$. Remember that $v \in L$. For a to extend R, vertex a must be connected to all vertices in L. This implies that vertex a is connected to vertex v by edge (v, a). This is a contradiction as $a \notin C(v)$. Thus no vertex $a \notin C(v)$ can extend R. Let us now take the other case, i.e. $a \in L$. This implies that there exists an edge (a, b) between vertex a and any vertex $b \in \eta(v)$, i.e. $a \in \eta^2(v)$. Again, this is a contradiction as $a \notin C(v)$. Thus no vertex $a \notin C(v)$ can extend L. □

3.2.2 Algorithm CDFS – Suppressing Duplicates

With the above observation, a basic parallel algorithm for MBE first constructs the different clusters $\{C(v) | v \in V\}$, and then enumerates the maximal bicliques in the different clusters in parallel, using any sequential algorithm for MBE for enumerating the bicliques within each cluster.

While each maximal biclique in G is indeed enumerated by the above approach, the same biclique may be enumerated multiple times. To suppress duplicates, the following strategy is used: a maximal biclique B arising from cluster $C(v)$ is emitted only if v is the smallest vertex in B according to a lexicographic total order on the vertices. The basic clustering framework is generic and can be used with any sequential algorithm for MBE. We have used a variant of the DFS-based sequential algorithm due to Liu et al. (2006), as well as the sequential consensus algorithm due to Alexe et al. (2004). We call the above basic clustering algorithm using DFS-based sequential algorithm as “CDFS”.

Observation 4. *Algorithm CDFS enumerates every maximal biclique in graph $G = (V, E)$ exactly once.*

There are two significant problems with the CDFS algorithm described above. First is *redundant work*. Although each maximal biclique in G is emitted only once, through suppressing duplicate output, it will still be generated multiple times, in different clusters. This redundant work significantly adds to the runtime of the algorithm. Second is an *uneven distribution of load* among different subproblems. The load on subproblem $C(v)$ depends on two factors, the complexity of cluster $C(v)$ (i.e. the number and size of maximal bicliques within $C(v)$) and the position of v in the total order of the vertices. The earlier v appears in the total order, the greater is the likelihood that a maximum biclique in $C(v)$ has v has its smallest vertex, and hence the greater is the responsibility for emitting bicliques that are maximal within $C(v)$. A lexicographic ordering of the vertices will lead to a significantly increased workload for clusters $C(v)$ where v appears early in the total order and a correspondingly low workload for clusters $C(v)$ where v occurs earlier in the total order.

Table 3.1: Different versions of Parallel Algorithms based on Depth First Search (DFS)

Label	Algorithm
CDFS	Clustering based on Depth First Search (DFS)
CD0	CDFS + Reducing Redundant Work, without Load Balancing
CD1	CDFS + Reducing Redundant Work + Load Balancing using Degree
CD2	CDFS + Reducing Redundant Work + Load Balancing using Size of 2-neighborhood

3.2.3 Algorithm CD0 – Reducing Redundant Work

In order to reduce redundant work done at different clusters, we begin with the basic clustering approach and modify the sequential DFS algorithm for MBE that is executed at each reducer. We first observe that in cluster $C(v)$, the only maximal bicliques that matter are those with v as the smallest vertex; the remaining maximal bicliques in $C(v)$ will not be emitted by this reducer, and need not be searched for here. We use this to prune the search space of the sequential DFS algorithm used at the reducer. The algorithm at the reducer is presented in Algorithms 7 and 8.

The above algorithm, the “optimized DFS clustering algorithm”, or “CD0” for short, is described in Algorithm 3. This takes two rounds of MapReduce. The first round, described in Algorithms 4 (map) and 5 (reduce), is responsible for generating the 1-neighborhood for each vertex. The second round, described in Algorithms 6 (map) and 7 (reduce) first constructs the clusters $C(v)$ and runs the optimized sequential DFS algorithm at the reducer to enumerate local maximal bicliques. We assume that the graph is presented as a file in HDFS organized as a list of edges with each line in the file containing one edge.

All search paths in the algorithm which lead to a maximal biclique having a vertex less than v can be safely pruned away. Hence, before starting the DFS, we prune away all vertices in the Tail set that are less than v, as described in Algorithm 7. Also, in DFS Algorithm 8, we prune the search path in Line 12 if the generated neighborhood contains a vertex less than v – maximal bicliques along this search path will not have v as the smallest vertex. Finally in Line 19 of Algorithm 8, we emit a maximal biclique only if the smallest vertex is the same as the key of the reducer in Algorithm 7.

Algorithm 3: Algorithm CD0

```
  Input: Edge List of $G = (V, E)$
1 MapReduce Round One (Map) – Algorithm 4 ;
2 MapReduce Round One (Reduce) – Algorithm 5 ;
3 MapReduce Round Two (Map) – Algorithm 6 ;
4 MapReduce Round Two (Reduce) – DFS – Algorithm 7 ;
```

Algorithm 4: Algorithm CD0 Round One – Map

```
  Input: Edge $(x, y)$
1 // Generate Adjacency List for vertices x and y
2 Emit ($key \leftarrow x$,$value \leftarrow y$) ;
3 Emit ($key \leftarrow y$,$value \leftarrow x$) ;
```

Algorithm 5: Algorithm CD0 Round One – Reduce

```
  Input: $key = v$, $value = \{v_1, v_2, \cdots, v_d\}$
1 // Generate Adjacency List for vertex v
2 $N \leftarrow \varnothing$ ;
3 forall the $val \in value$ do
4     $N \leftarrow N \cup val$ ;
5     // Add the neighbors of key to N
6 Emit ($key \leftarrow v$,$value \leftarrow N$) ;
```

Algorithm 6: Algorithm CD0 Round Two – Map

```
  Input: $key = v, value = N$
1 // Create Two Neighborhood for vertex v
2 Emit ($key \leftarrow v, value \leftarrow N$) ;
3 forall the $y \in N$ do
4     Emit ($key \leftarrow y, value \leftarrow \langle v, N \rangle$) ;
```

Algorithm 7: Algorithm CD0 Round Two – Reduce

```
  Input: $key = v$, $value = \{\eta(v), \eta(v_1), \eta(v_2), \cdots, \eta(v_d)\}$
1 // Create Two Neighborhood for vertex v from the values received
2 $G' = (V', E') \leftarrow$ Induced subgraph on $\eta^2(v)$ ;
3 $X \leftarrow key$ ;
4 $T \leftarrow V' \setminus \{key\}$ ;
5 forall the vertex $t \in T$ do
6     if $t < key$ then
7         $T \leftarrow T \setminus \{t\}$ ;
8 CD0_Sequential($G'$, $X$, $T$, $key$, $s$) ;
```

Lemma 2. *Algorithm 3 generates all maximal bicliques in a graph.*

Proof. The correctness of this Lemma can be proved from Lemma 1. Algorithm 3 generates the 2-neighborhood induced sub–graph of each vertex in G. It then runs the optimized sequential DFS algorithm that enumerates for each C(v), all the maximal bicliques where v is the smallest vertex. □

Lemma 3. *The total work done by Algorithm 3 is equal to the work done by the sequential DFS Algorithm 1.*

Proof. Algorithm 3 calls Algorithm 8 once for each vertex $v \in V$ for the input graph $G = (V, E)$. Thus there is one instance of Algorithm 8 created in parallel for each vertex v with input $C(v)$. Before we prove this, note that the the sequential DFS Algorithm 1 can be represented as a tree as follows. Let each recursive call to the method be a node in the tree. Let the value of the node be the set of vertices in the working set X in Algorithm 1. Each recursive call establishes a parent–child relationship where the calling instance of the method becomes the parent. Now to prove this lemma, we show that the work done by parallel instance of Algorithm 8 for vertex v is same as work done by the subtree of the sequential Algorithm 1 that starts with $X = v$.

Algorithm 8: CD0_Sequential(G', X, T, key, s)

```
Input: $G'$,$X$,$T$,$key$,$s$
// The sequential Algorithm to be run independently on each
   cluster for parallel Algorithm CD0
if $X = \{ key \}$ then
    $N \leftarrow \Gamma(X)$ // Same as $\Gamma(key)$
    $Y \leftarrow \Gamma(N)$ ;
    if $Y = X$ then
        Biclique $B \leftarrow \langle Y, N \rangle$ ;
        if $|Y| \geq s \wedge |N| \geq s$ then
            $v_s \leftarrow$ Smallest vertex in $B$ ;
            if $v_s = key$ then
                // Maximal biclique found
                Emit ($key \leftarrow \varnothing$,$value \leftarrow B$) ;
        else
            return
forall the vertex $v \in T$ do
    if $|\Gamma(X \cup \{v\})| < s$ then
        $T \leftarrow T \setminus \{v\}$ ;
if $|X| + |T| < s$ then
    return
Sort vertices in $T$ as per ascending order of $|\Gamma(X \cup \{v\})|$ ;
forall the vertex $v \in T$ do
    $T \leftarrow T \setminus \{v\}$ ;
    if $|X \cup \{v\}| + |T| \geq s$ then
        $N \leftarrow \Gamma(X \cup \{v\})$ ;
        $Y \leftarrow \Gamma(N)$ ;
        if $Y$ contains vertices smaller than $key$ then
            continue ;
        Biclique $B \leftarrow \langle Y, N \rangle$ ;
        if $(Y \setminus (X \cup \{v\})) \subseteq T$ then
            if $|Y| \geq s$ then
                $v_s \leftarrow$ Smallest vertex in $B$ ;
                if $v_s = key$ then
                    // Maximal biclique found
                    Emit ($key \leftarrow \varnothing$,$value \leftarrow B$) ;
            CD0_Sequential($G'$, $Y$, $T \setminus Y$, $key$, $s$) ;
```

Consider the root of the search tree for the sequential Algorithm 1. At the root the working set X is $\emptyset$. Let us consider the root to be depth 0. Let us assume some predefined ordering strategy of the tail set "T". Also, let us label the vertices $v_1 \cdots v_n$ following the ordering. Then for each vertex, $v \in V$, we have a branch that comes out of the root. Thus for depth 1, we have $(X_1 \leftarrow 1, T_1 \leftarrow V \setminus \{1\})$, $(X_2 \leftarrow 2$, $T_2 \leftarrow V \setminus \{1, 2\})$ and so on. Thus for each $v \in V$, we have $(X_v \leftarrow v, T_1 \leftarrow V \setminus \{1, 2, \cdots, v-1\})$. Hence for depth 1, we have the above mentioned $|V|$ calls.

Now we show that each such branch corresponds to the instance of the parallel Algorithm 8 such that the reducer $key = v$. To prove this, we note the call made to Algorithm 8 with key v. Algorithm 8 is called with $X = key$ and $\forall t \in T$, $t > v$. Thus we prune T such that $T \leftarrow V \setminus \{1, 2, \cdots, v-1\}$. This call is same as the branch of the search tree of Algorithm 1 that starts with key. The input graph to the parallel algorithm is different from the sequential one. However, from Lemma 1, this doesn't make a difference to the output of the parallel Algorithm.

Now, Algorithm 8 is different from Algorithm 1 in Lines 1–12 of Algorithm 8. However, we note that these lines simulate the call made in Algorithm 8 with $X = v$. All further recursive calls that follow are identical in both Algorithms 8 and 1. $\square$

3.2.4 Algorithms CD1 and CD2 – Improving Load Balance

In Algorithm CD0, vertices were ordered using a lexicographic ordering, which is agnostic of the properties of the cluster $C(v)$. The way the optimized DFS algorithm works, the enumeration load on a cluster $C(v)$ depends on the number of maximal bicliques within this cluster as well as the position of v within the total order. The earlier that v is in the total order, the greater is the load on the reducer handling $C(v)$.

For improving load balance, our idea is to adjust the position of vertex v in the total order according to the properties of its cluster $C(v)$. Intuitively, the more complex cluster $C(v)$ is (i.e. more and larger the maximal bicliques), the higher should be position of v in the total order, so that the burden on the reducer handling $C(v)$ is reduced. While it is hard to compute (or even accurately estimate) the number of maximal bicliques in $C(v)$, we consider two properties of vertex v that are simpler to estimate, to determine the relative ordering of v in the total order: (1) Size of 1-neighborhood of v (Degree), and (2) Size of 2-neighborhood of v.

Intuitively, we can expect that vertices with higher degrees are potentially part of a denser part of the graph and are contained within a greater number of maximal bicliques. The size of the 2-neighborhood is also the number of vertices in $C(v)$ and may provide a better estimate of the complexity of handling $C(v)$, but this is more expensive to compute than the size of the 1-neighborhood of the vertex.

The discussion below is generic and holds for both approaches to load balancing. To run the load balanced version of DFS, the reducer running the sequential algorithm must now have the following information for the vertex (key of the reducer) : (1) 2-neighborhood induced subgraph, and (2) vertex property for every vertex in the 2-neighborhood induced subgraph, where "vertex property" is the property used to determine the total order, be it the degree of the vertex or the size of the 2-neighborhood. The second piece of information is required to compute the new vertex ordering. However, the reducer of the second round does not have this information for every vertex in $C(v)$, and a third round of MapReduce is needed to disseminate this information among all reducers. We call the Algorithm using the size of 1–neighborhood of a vertex v as the heuristic as CD1 and the one using the size of 2–neighborhood as CD2.

Lemma 4. *The total work done by parallel Algorithm 9 is equal to the work done by the sequential DFS Algorithm 1.*

Proof. Note that the only difference between Algorithm 9 and Algorithm 3 is how they order the vertices. Algorithm 3 uses lexicographical ordering of vertices where as Algorithm 9 uses either degree or size of 2–neighborhood. Hence, the proof follows from the proof of Lemma 3. This is because, the proof of Lemma 3 makes no assumption on the strategy used to order the vertices in the graph. □

Algorithm 9: Algorithms CD1 and CD2

```
  Input: Edge List of $G = (V, E)$
1 MapReduce Round One (Map) – Algorithm 4 ;
2 MapReduce Round One (Reduce) – Algorithm 5 ;
3 MapReduce Round Two (Map) – Algorithm 6 ;
4 MapReduce Round Two (Reduce) – Algorithm 10 ;
5 MapReduce Round Three (Map) – Algorithm 11 ;
6 MapReduce Round Three (Reduce) – Algorithm 12 ;
```

Algorithm 10: Algorithms CD1 and CD2 Round Two – Reduce

```
   Input: $key = v$, $value = \{\eta(v), \eta(v_1), \eta(v_2), \cdots, \eta(v_d)\}$
1  // Send vertex property of vertex v to required nodes
2  $S \leftarrow$ 2–neighbors of $v$ ;
3  $N \leftarrow$ Compute neighborhood of $v$ from $S$ ;
4  // Need to pass neighborhood for Round 3
5  Emit ($key \leftarrow v$,$value \leftarrow N$) ;
6  // Need to send vertex property to all 2--neighbors
7  $p \leftarrow$ Value of vertex property of $v$ from $S$ ;
8  forall the vertices $s \in S$ do
9      Emit($key \leftarrow s$,$value \leftarrow [v, p]$) ;
```

Algorithm 11: Algorithms CD1 and CD2 Round Three – Map

```
   Input: $key = v, value = N$ OR $key = s, value = v, p$
1  // Create Two Neighborhood along with vertex property
2  if $key = v$ then
3      Emit ($key \leftarrow v$,$value \leftarrow N$) ;
4      forall the $y \in N$ do
5          Emit ($key \leftarrow y$,$value \leftarrow \langle v, N\rangle$) ;
6  else
7      Emit ($key \leftarrow s$,$value \leftarrow [v, p]$) ;
```

Algorithm 12: Algorithms CD1 and CD2 Round Three – Reduce

```
   Input: $key = v$, $value = \{\eta^2(v)$ along with vertex properties$\}$
1  // Create Two Neighborhood along with vertex property
2  $G' = (V', E') \leftarrow$ Induced subgraph on $\eta^2(v)$ ;
3  $Map \leftarrow$ HashMap of vertex and vertex property created from $value$ required to compute the
   new ordering ;
4  $X \leftarrow key$ ;
5  $T \leftarrow V' \setminus \{key\}$ ;
6  forall the vertex $t \in T$ do
7      if $t < key$ in the new ordering then
8          $T \leftarrow T \setminus \{t\}$ ;
9  CDL_Sequential($G'$, $X$, $T$, $key$,$Map$) ;
```

Algorithm 13: CDL_Sequential(G', X, T, key, s,Map)

```
Input: $G'$,$X$,$T$,$key$,$Map$
// The sequential Algorithm to be run independently on each
   cluster for parallel Algorithms CD1 / CD2
if $X = \{ key \}$ then
    $N \leftarrow \Gamma(X)$ // Same as $\Gamma(key)$
    $Y \leftarrow \Gamma(N)$ ;
    if $Y = X$ then
        Biclique $B \leftarrow \langle Y, N \rangle$ ;
        if $|Y| \geq s \wedge |N| \geq s$ then
            $v_s \leftarrow$ Smallest vertex in $B$ in the new ordering ;
            if $v_s = key$ then
                // Maximal biclique found
                Emit ($key \leftarrow \varnothing$,$value \leftarrow B$) ;
        else
            return
forall the vertex $v \in T$ do
    if $|\Gamma(X \cup \{v\})| < 1$ then
        $T \leftarrow T \setminus \{v\}$ ;
if $|X| + |T| < 1$ then
    return
Sort vertices in $T$ as per ascending order of $|\Gamma(X \cup \{v\})|$ ;
forall the vertex $v \in T$ do
    $T \leftarrow T \setminus \{v\}$ ;
    if $|X \cup \{v\}| + |T| \geq 1$ then
        $N \leftarrow \Gamma(X \cup \{v\})$ ;
        $Y \leftarrow \Gamma(N)$ ;
        if $Y$ contains vertices smaller than key in the new ordering then
            continue ;
        Biclique $B \leftarrow \langle Y, N \rangle$ ;
        if $(Y \setminus (X \cup \{v\})) \subseteq T$ then
            if $|Y| \geq 1$ then
                $v_s \leftarrow$ Smallest vertex in $B$ in the new ordering ;
                if $v_s = key$ then
                    // Maximal biclique found
                    Emit ($key \leftarrow \varnothing$,$value \leftarrow B$) ;
            CDL_Sequential($G'$, $Y$, $T \setminus Y$, $key$,$Map$) ;
```

3.2.5 Communication Complexity

We consider the communication complexity of Algorithms CD0, CD1 and CD2. For input graph $G = (V, E)$, recall $n = |V|$ and $m = |E|$. Let Δ denote the largest degree among all vertices in the graph. Also, let β denote the output size, defined as the sum of the numbers of edges of all enumerated maximal bicliques.

Definition 8. *The communication complexity of a MapReduce algorithm is defined as the sum of the total number of bytes emitted by all mappers and the total number of bytes emitted by all the reducers across all rounds.*

Lemma 5. *The communication complexity of Algorithm CD0 is $O(m \cdot \Delta + \beta)$.*

Proof. Algorithm CD0 has two rounds of MapReduce. In the first round the Map method (Algorithm 4) emits each edge twice, resulting in a communication complexity of $O(m)$. Similarly, the reducer (Algorithm 5), emits each adjacency list once. This also results in a communication complexity of $O(m)$. Hence total communication complexity of the first round is $O(m)$.

Now let us consider the second round of MapReduce. The total communication between the Map and Reduce methods (Algorithms 6 and 7 respectively) can be computed by analyzing how much data is received by all Reducers. Each reducer receives the adjacency list of all the neighbors of the key. Let d_i be the degree of vertex v_i, for $v_i \in V, i = 1, .., n$. The adjacency list of vertex v is sent to all vertices in that list. The size of adjacency list is d_i. This list is sent to d_i vertices. Thus communication complexity for vertex v_i becomes ${d_i}^2$. Total communication is thus $\sum_{i=1}^{n} {d_i}^2 = O\left(\Delta \cdot \sum_{i=1}^{n} d_i\right)$. Since $\sum_{i=1}^{n} d_i = 2 \cdot m$, the total communication becomes $O(m \cdot \Delta)$. The output from the final Reducer (Algorithm 7) is the collection of all maximal bicliques and hence the resulting communication cost is $O(\beta)$. Combining two rounds, total communication complexity becomes $O(m + m \cdot \Delta + \beta). = O(m \cdot \Delta + \beta)$. □

Lemma 6. *The communication complexity of Algorithm CD1 as well as CD2 is $O(m \cdot \Delta + \beta)$.*

Proof. First, note that both Algorithms CD1 and CD2 have the same communication complexity and observe that the first round uses the same Map and Reduce methods as CD0. Thus communication for Round 1 is $O(m)$. Again, note that Map method for Round 2 is same as CD0 and hence by Lemma 5, communication for Round 2 is $O(m \cdot \Delta)$.

The Reducer (Algorithm 10) of Round 2 sends the vertex property information to all its 2–neighbors. Thus every reducer receives information about all of its 2–neighbors. This makes the total output size of Reducer to be $O\left(m \cdot \Delta\right)$. The Map method of Round 3 (Algorithm 11) sends out the 2–neighborhood information as well as the vertex information to all vertices in 2–neighborhood. Thus communication cost becomes $O\left(m \cdot \Delta\right)$. The Reducer (Algorithm 12) emits all maximal bicliques and hence the resulting communication cost is $O\left(\beta\right)$. Thus total communication cost for Algorithms CD1 and CD2 is is $O\left(m \cdot \Delta + \beta\right)$. □

3.2.6 Parallel Consensus

We briefly describe another approach that we devised, which directly parallelizes the consensus sequential algorithm of Alexe et al. (2004), in a manner different from the clustering approach. The motivation for this approach is as follows. The clustering approach has the following potential drawback: it requires each cluster $C(v)$ to have the entire 2-neighborhood of v. For dense graphs, the size of the 2-neighborhood of a vertex can be large, so that the complexity of each reduce task can be substantial.

Unlike the parallel DFS algorithm which works on subgraphs of G, the consensus algorithm is always directly dealing with bicliques within graph G. At a high level, it performs two operations repeatedly (1) a "consensus" operation, which creates new bicliques by considering the combination of existing bicliques, and (2) an "extension" operation, which extends existing bicliques to form new maximal bicliques. There is also a need for eliminating duplicates after each iteration, and also a step needed for detecting convergence, which happens when the set of maximal bicliques is stable and does not change further.

We developed a parallel version of each of these operations, by performing the consensus, extension and duplicate removal using MapReduce.

Our Algorithm is described in Algorithm 14. Algorithms 15 and 16 (map and reduce) describe the consensus operation using MapReduce. Note that in Algorithm 14, line 8 performs consensus between each pair of biclique in sets S (seed set) and P (set of bicliques from previous round of iteration). To perform consensus between the all bicliques from the sets S and P naively, it would require $|S| \star |P|$ consensus operations. However, we reduce the total number of consensus operations using the following

Algorithm 14: Parallel Consensus Algorithm – Driver Program

```
1  Load Graph G = (V,E) ;
2  $R \leftarrow$ Star bicliques from $G$ // Biclique formed by a vertex and its
       neighbors
3  $S \leftarrow$ Extend all bicliques in $R$ using MapReduce ;
4  Eliminate duplicates from $S$ using MapReduce ;
5  $O \leftarrow O \cup S$ ;
6  $P \leftarrow S$ ;
7  repeat
8      $T \leftarrow$ Consensus among all maximal bicliques in $S$ and $P$ using MapReduce ;
9      $C \leftarrow$ Extend all bicliques in $T$ using MapReduce ;
10     Eliminate duplicates from $C$ using MapReduce ;
11     $O \leftarrow O \cup C$ ;
12     $P \leftarrow C$ ;
13 until $N$ is $\varnothing$;
```

Algorithm 15: Parallel Consensus Algorithm – Consensus Map

```
1  forall the $i$ such that $i$ is an node in the left set of the biclique $H$ do
2      Emit $(i, H)$ ;
3  forall the $j$ such that $j$ is an node in the right set of the biclique $H$ do
4      Emit $(j, H)$ ;
```

Algorithm 16: Parallel Consensus Algorithm – Consensus Reduce

```
1  forall the $x$ such that $x$ is a seed biclique containing the key $k$ do
2      forall the $y$ such that $y$ is a biclique from previous round having the key $k$ do
3          if key = minimum common element of the bicliques x and y then
4              $C \leftarrow$ Potentially new maximal bicliques from consensus of $x$ and $y$ ;
5              forall the $c$ in $C$ do
6                  Extend the biclique $c$ to generate maximal biclique $H$ ;
7                  Emit $(\varnothing, H)$ ;
```

Algorithm 17: Parallel Consensus Algorithm – Extension Map

```
1  $B \leftarrow$ Input biclique ;
2  if $B$ is a star then
3      $x \leftarrow$ Main vertex ;
4      Emit ($x$,$B$) ;
5  if data is from consensus output then
6      forall the vertices $i$ such that $i$ is in $B$ do
7          Emit ($i$,$B$) ;
```

Algorithm 18: Parallel Consensus Algorithm – Extension Reduce

```
1  $S \leftarrow \varnothing$ ;
2  forall the $value$ for $key$ do
3      if $value$ is a neighborhood information then
4          $N \leftarrow$ Neighborhood of vertex $key$ ;
5      else
6          $S \leftarrow S \cup value$ ;
7  forall the bicliques $b$ in $S$ do
8      $h \leftarrow$ Hash value of biclique $b$ ;
9      Emit $(h,b)$ ;
10     Emit $(h,N)$ ;
```

observation: If there are no common vertices between two bicliques, in that case the consensus output between the concerned two bicliques is the NULL set. This is because the intersection operation in the consensus will result in NULL. This helps us to "group" the bicliques in $n = |V|$ sets, one for each vertex of the graph. A biclique is a part of the group for vertex v, if v is contained in the biclique. The map method helps to achieve this by "grouping" all bicliques having a particular vertex in common, thus eliminating the need of doing unnecessary consensus operations.

Finally we explain the extension operation. To reduce memory requirement, we required four rounds of MapReduce to perform the extension. The intention of the process is to bring together only those neighborhood information, which is required to extend a biclique. Algorithms 17 and 18 describe the map and reduce algorithms for the first round. Recall that the extension operation requires computation of 2-neighborhood of both the left and right set of the vertices in the biclique. The first two rounds of MapReduce are used to compute the 1-neighborhood of both the sets and then the same two rounds are run one more time to obtain the 2-neighborhood information.

Finally, the algorithm stops when no new maximal bicliques are found after completing an iteration. The Driver Algorithm 14 checks for the same and halts if no new maximal bicliques are found.

3.3 Experimental Results

We implemented our parallel algorithms on a Hadoop cluster, using both real-world and synthetic datasets. The cluster has 28 nodes, each with a quad-core AMD Opteron processor with 8GB of RAM.

All programs were written using Java version 1.5.0 with 2GB of heap space, and the Hadoop version used was 1.2.1.

We implemented the DFS based algorithms CDFS (clustering DFS with no optimizations), CD0 (clustering DFS with the pruning optimization), CD1 (clustering DFS with pruning and load balancing using degree), and CD2 (clustering DFS with pruning and load balancing using size of 2-neighborhood).

We also implemented the sequential DFS algorithm due to Liu et al. (2006), and the sequential consensus algorithm (MICA) due to Alexe et al. (2004). The sequential algorithms were not implemented on top of Hadoop and hence had no associated Hadoop overhead in their runtime. But on the real-world graphs that we considered, the sequential algorithms did not complete within 12 hours, except for the p2p-Gnutella09 graph. In addition, we implemented the parallel clustering algorithm using the consensus-based sequential algorithm, and we also implemented an alternate parallel implementation of the consensus algorithm that was not based on the clustering method.

Table 3.2: Various properties of the input graphs used, and runtime (in seconds) of different algorithms to enumerate all maximal bicliques within the graph using 100 reducers. DNF means that the algorithm did not finish in 12 hours. The size threshold was set as 1 to enumerate all maximal bicliques. Runtime includes overhead of all MapReduce rounds including graph clustering, i.e. formation of 2–neighborhood.

Input Graph	#vertices	#edges	#max–bicliques	Output Size	CDFS	CD0	CD1	CD2
p2p-Gnutella09	8114	26013	20332	203779	113	60	79	80
email-EuAll-0.6	125551	168087	292008	4580577	42023	4188	415	406
com-Amazon	334863	925872	706854	6369954	186	65	95	101
amazon0302	262111	1234877	886776	7276888	396	264	102	97
com-DBLP-0.6	251226	419573	1875185	41407481	1659	285	239	314
email-EuAll-0.4	175944	252075	2003426	55685463	DNF	33300	3140	2196
ego-Facebook-0.6	3928	35397	6597716	157777680	8657	2773	918	1847
loc-BrightKite-0.6	49142	171421	10075745	388709764	28585	6511	1381	1997
web-NotreDame-0.8	150615	300398	19941634	471150086	DNF	27827	1044	1577
ca-GrQc-0.4	5021	17409	16133368	1550607157	37279	4104	3728	4085
ER-50K	50000	275659	51756	558376	96	57	76	81
ER-60K	60000	330015	61821	667358	98	59	76	81
ER-70K	70000	393410	71962	794704	98	62	77	76
ER-80K	80000	448289	81983	904535	102	62	77	78
ER-90K	90000	526943	92214	1062772	109	63	78	77
ER-100K	100000	600038	102663	1210764	114	70	80	85
ER-250K	250000	1562707	252996	3137432	167	80	107	109
ER-500K	500000	3751823	506319	7528935	374	128	170	163
Bipartite-50K-100K	150000	1999002	306874	4628028	873	122	163	170
Bipartite-75K-150K	225000	11250524	27650168	136660625	DNF	8956	8351	8149

We used both synthetic and real-world graphs. A summary of all the graphs used is shown in Table 3.2. The real-world graphs were obtained from the SNAP collection of large networks (see Leskovec ()) and were drawn from social networks, collaboration networks, communication networks, product co-purchasing networks, and internet peer-to-peer networks. Some of the real world networks were so large and dense that no algorithm was able to process them. For such graphs, we thinned them down by deleting edges with a certain probability. This makes the graphs less dense, yet preserves some of the structure of the real-world graph. We show the edge deletion probability in the name of the network. For example, graph "ca-GrQc-0.4" is obtained from "ca-GrQc" by deleting each edge with probability 0.4. Synthetic graphs are either random graphs obtained by the Erdos-Renyi model (see Erdős and Rényi (1959)), or random bipartite graphs obtained using a similar model. To generate a bipartite graph with n_1 and n_2 vertices respectively in the two partitions, we randomly assign an edge between each vertex in the left partition to each vertex in the right partition. A random Erdos-Renyi graph on n vertices is named "ER-$\langle n \rangle$", and a random bipartite graph with n_1 and n_2 vertices in the bipartitions is called "Bipartite-$\langle n_1 \rangle$-$\langle n_2 \rangle$".

We seek to answer the following questions from the experiments: (1) What is the relative performance of the different methods for MBE? (2) How do these methods scale with increasing number of reducers? and (3) How does the runtime depend on the input size and the output size?

Figure 3.1 presents a summary of the runtime data for the algorithms in Table 3.2. All data used for these plots was generated with 100 reducers. *The runtime data given for the Parallel Algorithms include the time required to run all MapReduce rounds including time required to construct 2–neighborhood etc*.

3.3.1 Impact of the Pruning Optimization

From Figure 3.1, we can see that the optimizations to basic DFS clustering through eliminating redundant work make a significant impact to the runtime for all input graphs. For instance, in Figure 3.1d, on input graph email–EuAll–0.6 CD0, which incorporates these optimizations, runs 10 times faster than CDFS, the basic clustering approach. Also, we can see from Table 3.2 that the input graphs email–EuAll–0.4, web–NotreDame–0.8 and Bipartite–75K–150K could not be processed by Algorithm CDFS within 11 hours but could be processed by Algorithm CD0.

3.3.2 Impact of Load Balancing

From Figure 3.1, we observe that for graphs on which the algorithms do not finish very quickly (within 200 seconds), load balancing helps significantly. In Figure 3.1d, for graph email–EuAll–0.6, the Load Balancing approaches (CD1 and CD2) are 10 to 10.3 times faster than CD0, which incorporates the pruning optimization, without load balancing. In Figure 3.1b, we note that for input graph web–NotreDame–0.8, CD1 was 26.7 times faster than CD0 and CD2 was about 17.6 times faster. We can also observe the improvements in Load Balance from the reducer timings. For input graph email–EuAll–0.4, we observe that for CD0, most reducers finish in a few minutes. A very few took 2 hours. However, the last two reducers took 4.5 hours and 9.25 hours. By improving load balance in Algorithms CD1 / CD2, we redistribute this work load bringing the parallel runtime of both Algorithms to below one hour.

For most input graphs, the versions optimized through load balancing and pruning (Algorithms CD1 and CD2) worked the best overall, and both these optimizations helped significantly in reducing the runtime.

However, for graphs that completed quickly, load balancing performs slightly slower than Algorithm CD0 (see Figure 3.1a). This can be explained by the additional overhead of load balancing (an extra round of MapReduce), which does not payoff unless the work done at the DFS step is significant.

There are two different approaches to load balancing, one based on the vertex degree (Algorithm CD1) and the other on the size of the 2-neighborhood of the vertex (Algorithm CD2). From Figure 3.1 we observed that no one approach was consistently better than the other, and the performance of the two were close to each other. For some input graphs, like Email-EuAll-0.4, the 2-neighborhood approach (CD2) fared better than the degree approach (CD1), whereas for some other input graphs like web-NotreDame-0.8, the degree approach fared better.

To better understand the impact of load balancing, we calculated the mean and the standard deviation of the run time of each of the 100 reducers for the last round of MapReduce of Algorithms CD0, CD1 and CD2. We present results of this analysis for input graphs loc-BrightKite-0.6 and ego-Facebook-0.6 in Table 3.3. **The load balanced CD1 and CD2 have a much smaller standard deviation for reducer runtimes than CD0.**

We observe that random graphs have less variance in degree / size of 2–neighborhood than real world graphs. This leads to approximately balanced load on each node in the cluster, irrespective of how the work is distributed. Hence we don't get benefit out of the extra overhead involved in CD1 and CD2. Thus for randoms graphs, Algorithm CD0 performs better than Algorithms CD1 / CD2.

3.3.3 Scaling with Number of Reducers

In Figure 3.2 we plot the runtime of CD1 and CD2 with increasing number of reducers. In Figure 3.3, we also plot the speedup, defined as the ratio of the time taken with 1 reducer to the time taken with r reducers, as a function of the number of reducers r. We observe that the runtime decreases with increasing number of reducers, and further, the algorithms achieve near-linear speedup with the number of reducers. This data shows that the algorithms are scalable and may be used with larger clusters as well.

3.3.4 Relationship to Output Size

We observed the change in runtime of the algorithms with respect to the output size. We define the output size of the problem as the sum of the numbers of edges of all enumerated maximal bicliques. Figure 3.4 shows the runtime of algorithms CD0, CD1, and CD2 as a function of the output size. This data is only constructed for random graphs, where the different graphs considered are generated using the same model, and hence have very similar structure. We observe that **the runtime increases almost linearly with the output size for all three algorithms CD0, CD1, and CD2**.

With real world graphs, this comparison does not seem as appropriate, since the different real worlds graphs have completely different structures; however, we observed that the runtimes of Algorithms CD1 and CD2 are well correlated with the output size, even on real world graphs.

3.3.5 Large Maximal Bicliques

Next, we considered the variant where only large bicliques, whose total number of vertices is at least s, are required to be emitted. Figure 3.5 shows the runtime as the size threshold s varies from 1 to 5. We observe that the runtime decreases significantly as the threshold increases. Also, Algorithms CD1 and CD2 were not able to enumerate all maximal bicliques from input graph email-EuAll-0.2 even

Table 3.3: Mean and Standard Deviation computation of all 100 reducer runtimes for Algorithms CD0, CD1 and CD2. The analysis is done for the Reducer of the last MapReduce round as it performs the actual Depth First Search.

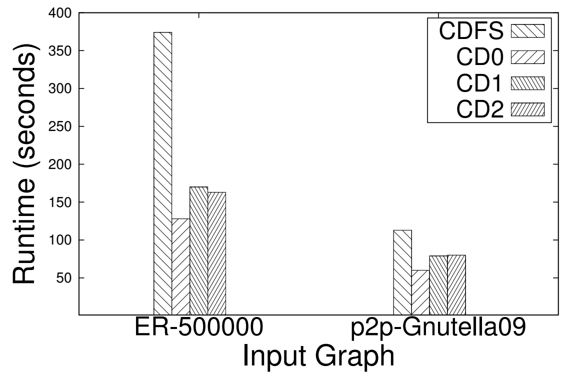
loc-BrightKite-0.6	CD0	CD1	CD2
Average	637.27	387.77	393.68
Variance	1259680.12	81447.47	111443.13
Standard Deviation	1122.35	285.39	333.83
ego-Facebook-0.6	CD0	CD1	CD2
Average	313.56	245.21	273.43
Variance	203661.36	29166.29	108260.19
Standard Deviation	451.29	170.78	329.03

after 12 hours. However, with size threshold 5, Algorithm CD1 took less than 6 hours to process this graph while Algorithm CD2 took about 3.5 hours.

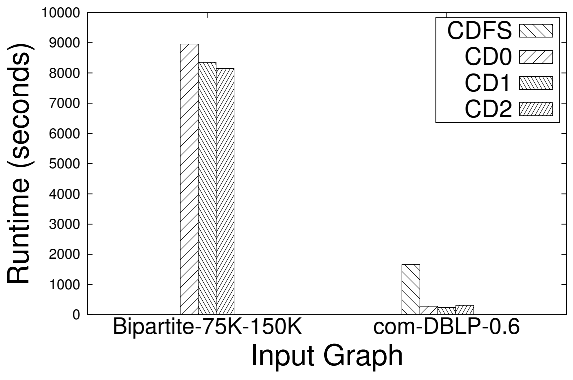
3.3.6 Consensus versus Depth First Search

Finally we compare the two sequential techniques used in this work. The basic clustering method was used with the consensus technique and compared with Algorithms CD1 and CD2. We note that the clustering approach using the consensus technique performed very poorly compared with the DFS based algorithm. For example for the input graph p2p–Gnutella09, CD1 and CD2 took 79 and 80 seconds respectively (using 100 reducers). This is in contrast with the implementation of the clustering method using the consensus technique which took 1469 seconds (again with 100 reducers). In all instances except for very small input graphs, clustering using consensus was 6 to 15 times slower than CD1 and CD2 or worse, and in many cases, clustering consensus did not finish within 12 hours while CD1 and CD2 finished within 1 - 2 hours.

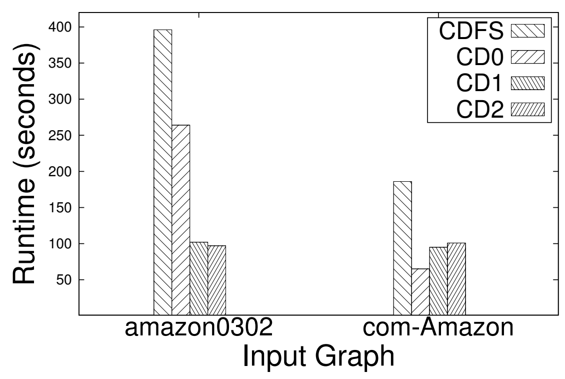
We also compared the runtime of the more direct parallel implementation of the consensus technique as described in Algorithm 14. The direct parallel consensus, which uses a different parallelization strategy was 13 to 100 times slower than clustering consensus. For example, for input graph ER–50K, Algorithm CD1 finished processing in 76 seconds, whereas Algorithm 14 took over an hour. Further, it could not process the p2p–Gnutella09 input graph within 12 hours.



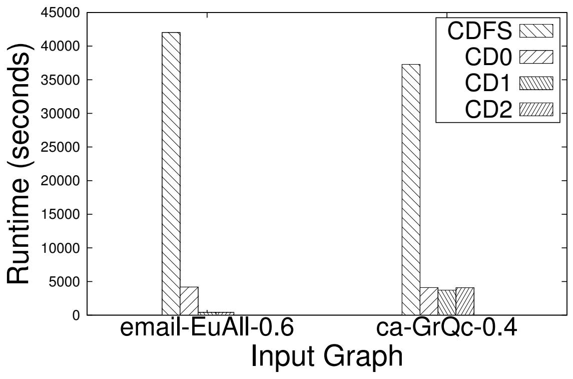
(a) ER-500K & p2p-Gnutella09



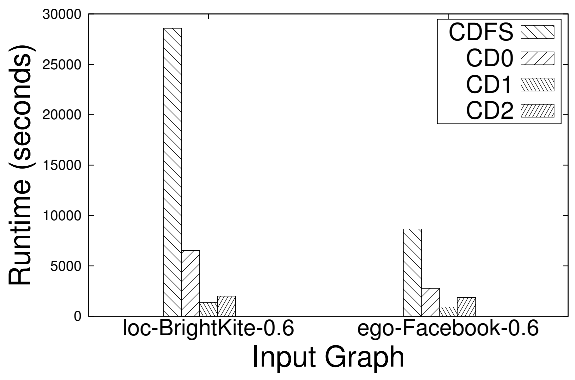
(b) web-NotreDame-0.8 & Bipartite-50K-100K



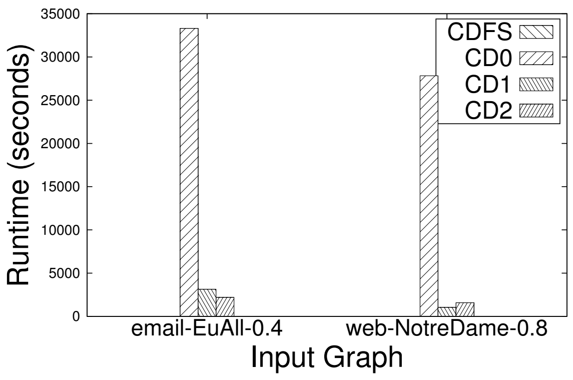
(c) amazon0302 & com-Amazon



(d) email-EuAll-0.6 & ca-GrQc-0.4

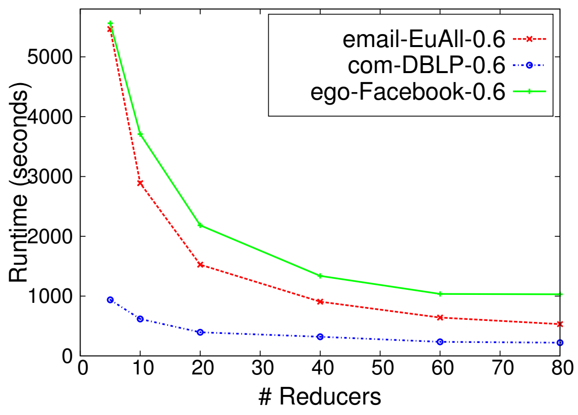


(e) loc-BrightKite-0.6 & ego-Facebook-0.6

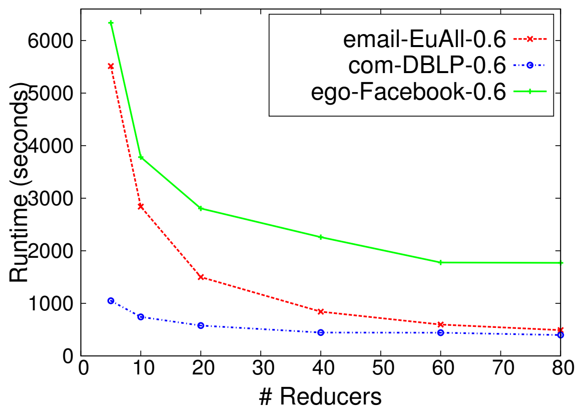


(f) email-EuAll-0.4 & com-DBLP-0.6

Figure 3.1: Runtime in seconds of parallel algorithms on real and random graphs. If an algorithm failed to complete in 12 hours the result is not shown. All algorithms were run using 100 reducers. Runtime includes overhead of all MapReduce rounds including graph clustering, i.e. formation of 2–neighborhood.

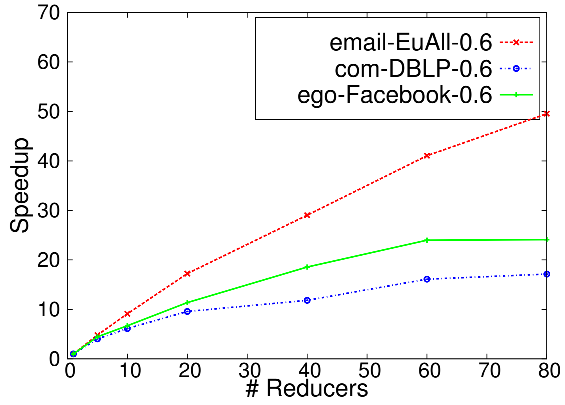


(a) Algorithm CD1

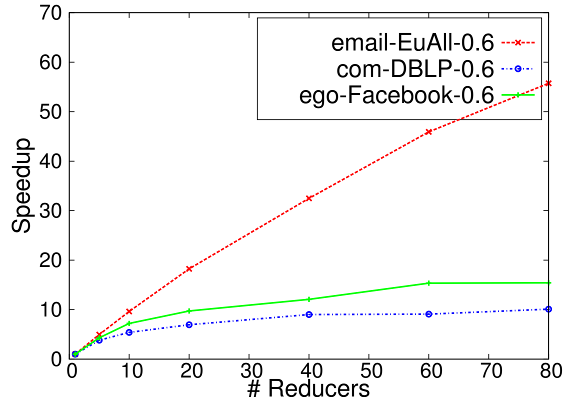


(b) Algorithm CD2

Figure 3.2: Runtime versus Number of Reducers.



(a) Algorithm CD1



(b) Algorithm CD2

Figure 3.3: Speedup versus Number of Reducers.

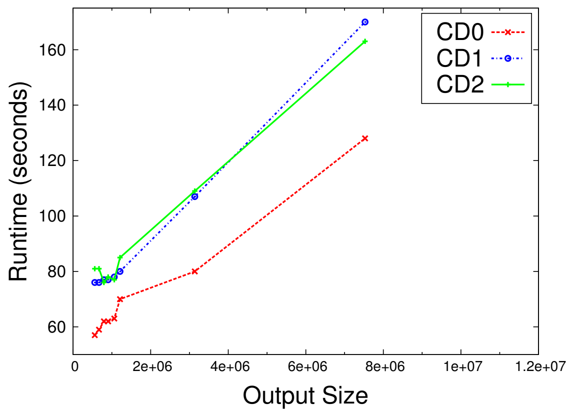
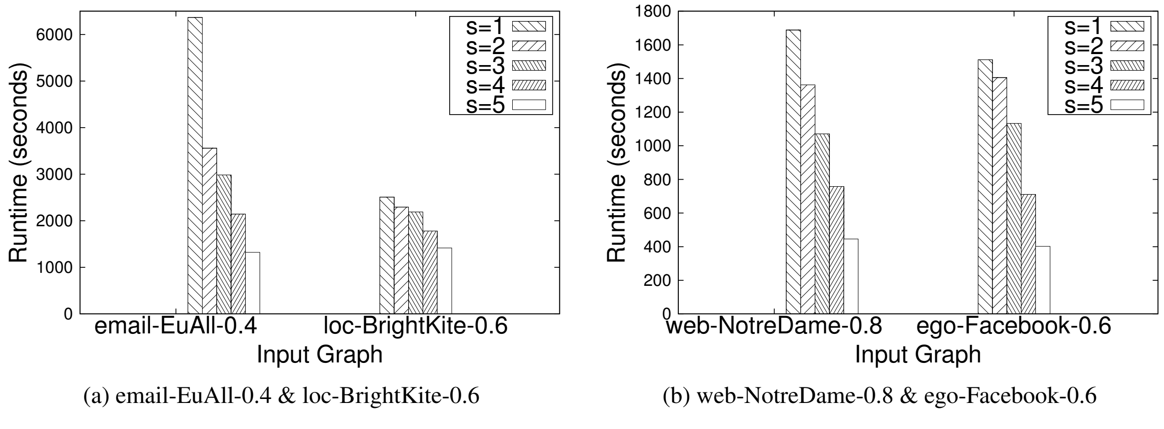


Figure 3.4: Runtime versus Output Size for random graphs. All Erdos-Renyi random graphs were used. Output size is defined as the number of edges summed over all maximal bicliques enumerated.



(a) email-EuAll-0.4 & loc-BrightKite-0.6

(b) web-NotreDame-0.8 & ego-Facebook-0.6

Figure 3.5: Runtime vs the size threshold for the emitted maximal bicliques. All experiments were performed using Algorithm CD1 and with 100 reducers.

CHAPTER 4. MINING ALL MAXIMAL CLIQUES

4.1 Algorithm

We first discuss a straightforward approach to parallel MCE using MapReduce. For $v \in V$, let $\Gamma(v)$ denote the neighbors of v in G, and let G_v denote the subgraph of G induced by $v \cup \Gamma(v)$. The following observation is easy to verify: each maximal clique $C \subseteq V$ is also a maximal clique in G_v for any vertex $v \in C$, and vice versa. A parallel algorithm works as follows: first construct (in parallel) the different subgraphs $\{G_v | v \in V\}$ and then separately enumerate maximal cliques in each of them using a sequential MCE algorithm, such as the ones in Bron and Kerbosch (1973); Tomita et al. (2006); Eppstein et al. (2010). The details are as follows.

As seen in the previous chapter, this naive approach as the following drawbacks:

I. The first is *duplication of cliques* in the results. A clique C with k vertices will be enumerated k times, once for each vertex $v \in C$. In earlier approaches Wu et al. (2009); Lu et al. (2010), this was handled using a post-processing step that eliminated duplicates. But a post-processing step has two problems; one is that it requires another communication-intensive round of MapReduce. The other is that size of intermediate output, with duplicate cliques, can be much larger than the size of the final output.

II. The second is *redundant work* in computing cliques. The different subgraphs G_v are explored by independent reduce tasks without communication among them, and the work done to enumerate a clique of size k is repeated k-fold. This is a major source of inefficiency in parallelization. Note that even if communication were allowed between different tasks exploring the subgraphs G_v, it is non-trivial to eliminate this redundant work in clique enumeration.

III. The third is *load balancing*. This problem arises since the subproblems for different vertices may vastly vary in size. For example, a vertex that is a part of many maximal cliques, or a part of maximal cliques of a relatively large size, will give rise to a more computationally intensive subproblem than a vertex that is part of only a few maximal cliques. Consequently, the distribution of work across subproblems is non-uniform, sometimes to an extreme degree.

4.1.1 Intuition

The key to our approach is an appropriately chosen *total order* among all vertices in V. Let `rank` define a function whose domain is V and which assigns an element from some totally ordered universe to each vertex in V. For $u, v \in V$ and $u \neq v$, either $\texttt{rank}(u) > \texttt{rank}(v)$ or $\texttt{rank}(v) > \texttt{rank}(u)$. The function `rank` implicitly defines a total order among all vertices in V.

Eliminating Duplicate Cliques Given the `rank` function, Problem I (duplicate cliques) is handled as follows. When a clique C is found by the reduce task for vertex v, C is output only if $\forall u \in C$, $\texttt{rank}(v) \leq \texttt{rank}(u)$. i.e., v has the smallest rank among all vertices in C. Otherwise C is simply discarded by v. Since only one vertex satisfies this condition for each clique C, each clique will be output exactly once. This removes the need for post-processing to eliminate duplicates.

Eliminating Redundant Work However, the above does not as easily solve Problem II (redundant work). Consider a clique C that has k vertices. While the above approach ensures that C is output only once, it is still computed by k different reduce tasks, and discarded by all but one of them. Eliminating this redundancy is more challenging, especially in a system such as MapReduce, since it is not possible for different reduce tasks to communicate and share state with each other. Our approach to this problem is to use the total ordering on vertices in conjunction with a modification to a sequential algorithm due to Tomita et al. (2006), which will allow us to ignore search paths that involve vertices with a smaller value of `rank`. We discuss this further in the following sections.

Improving Load Balance Let σ be a specific ranking function used to order vertices in G. For each vertex $v \in V$, there is a subproblem G_v, as defined above. Let ζ_v denote the set of all maximal cliques in G_v. With the above approach to reducing redundant work and avoiding duplicates, the reducer responsible for vertex v (which receives G_v as an input) is not required to enumerate all of ζ_v. Instead the reducer for v only has to enumerate those maximal cliques $C \in \zeta_v$ where v is the smallest vertex in

C according to the total order induced by σ. Let $\zeta_v(\sigma) \subseteq \zeta_v$ be the set of maximal cliques C such that v is the smallest vertex in C according to σ; $\zeta_v(\sigma)$ is the set of maximal cliques that are required to be enumerated from the subproblem G_v. A key observation is that we can tailor our sequential algorithm for subproblem G_v such that it is able to avoid the work done to enumerate cliques that are in ζ_v but not in $\zeta_v(\sigma)$.

As a result of this, the computational cost of subproblem G_v depends on two factors: the number and sizes of cliques in ζ_v, and the rank of v in the total order relative to other vertices in G_v. The higher is the rank of v in the total order, the fewer cliques in ζ_v it is responsible for.

In deciding the `rank` function, in order to keep the sizes of subproblems approximately balanced, the intuition is to assign a high value of `rank` for a vertex v for which $|\zeta_v|$ is large, and a small value of `rank` if $|\zeta_v|$ is small. Therefore, we define the "ideal" total order as follows: *If* $|\zeta_u| > |\zeta_v|$ *then* u *is ranked higher in the total order than* v. Overall, this increases the work done by vertices with a lower rank (for which the size of ζ_v is small) and decreases the work done by vertices with a larger rank (for which the size of ζ_v is large), resulting in a more even distribution of work. A difficulty with working with this ideal total order is that computing $|\zeta_v|$ is an expensive task in itself. It is not reasonable to spend too much effort in computing $|\zeta_v|$ exactly, since it is only used within an optimization. Instead, we base our ranking of vertices on metrics that are more easily computed, but provide some guidance on the number of cliques a vertex is a part of. We consider the following strategies for approximating the ordering described above.

- The Degree Ordering is defined through the following function. For vertex v, $\texttt{rank}(v) = (d, v)$, where d is the degree of v, and v the vertex identifier. Given two distinct vertices v_1 and v_2, and their ranks $\texttt{rank}(v_1) = (d_1, v_1)$ and $\texttt{rank}(v_2) = (d_2, v_2)$: $\texttt{rank}(v_1) > \texttt{rank}(v_2)$ if either $d_1 > d_2$, or if $d_1 = d_2$ and $v_1 > v_2$; otherwise, $\texttt{rank}(v_1) < \texttt{rank}(v_2)$. Given two vertices, it is easy to evaluate their relative position in the total order, since the degree of each vertex is readily available as the size of the neighbor list of the vertex. One can expect that the higher the degree of v, the larger is the size of ζ_v, though this may not always be true.

- The Triangle Ordering is defined as $\texttt{rank}(v) = (t, v)$, where t is the number of triangles (cliques of size 3) the vertex is a part of, and v is the vertex id. The relative ordering among tuples is defined the same way as in the degree ordering.

When compared with the degree ordering, the triangle ordering can be expected to be produce a total order that is closer to the ordering produced through the use of $|\zeta_v|$. Hence, it has the advantage that it can be expected to yield better load balance. However, it has the downside that it is an additional overhead to count the number of triangles that a vertex is a part of (another MapReduce task).

We also consider two other simple ordering strategies that are agnostic of the number of cliques a vertex is a part of.

- The Lexicographic Ordering is defined as `rank`$(v) = v$. It is assumed that the vertex ids themselves are unique and are chosen from a totally ordered set.

- The Random Ordering is defined as `rank`$(v) = (r, v)$, where r is a random number between 0 and 1, and v is the vertex id. Note that r is the most significant set of bits, with v only used as a tiebreaker in the event the r values of two vertices are equivalent.

4.1.2 Tomita *et al.* Sequential MCE Algorithm

PECO uses the sequential maximal clique enumeration algorithm by Tomita et al. (2006). The algorithm has a running time of $O(3^{\frac{n}{3}})$, which is worst case optimal, due to known lower bounds Moon and Moser (1965). Although only guaranteed to be optimal in the worst case, in practice, it is found to be one of the fastest on typical inputs. We present a brief description of this algorithm

Tomita et al. (2006) is based on the depth first search algorithm by Bron and Kerbosch (1973). Algorithm 19 shows the `Tomita` recursive function. The function takes as parameters a graph G and the sets K, `Cand`, and `Fini`. K is a clique (not necessarily maximal), which the function will extend to a larger clique if possible. `Cand` is the set $\{u \in V : u \in \Gamma(v), \forall v \in K\}$, or simply $u \in$ `Cand` must be a neighbor of every $v \in K$. Therefore, any vertex in `Cand` could be added to K to make a larger clique. `Fini` contains all the vertices which were previously in `Cand` and have already been used to extend the clique K.

The base case for the recursion occurs when `Cand` is empty. If `Fini` is also empty, then K is a maximal clique. If not, then a vertex from `Fini` could be added to K to form a larger clique. However, each vertex in `Fini` has already been explored, adding it would re-explore a previously searched path.

Algorithm 19: Tomita($G, K, \texttt{Cand}, \texttt{Fini}$)

```
Input:  $G$ - a graph
        $K$ - a non-maximal clique to extend
        Cand - the set of vertices that could be used to extend $K$
        Fini - the set of vertices previously used to extend $K$
1  if $(\texttt{Cand} = \emptyset)$ & $(\texttt{Fini} = \emptyset)$ then
2      report $K$ as maximal
3      return
4  pivot $\leftarrow u \in \texttt{Cand} \cup \texttt{Fini}$ that maximizes the intersection $\texttt{Cand} \cap \Gamma(u)$
5  Ext $\leftarrow \texttt{Cand} - \Gamma(\texttt{pivot})$
6  for $q \in \texttt{Ext}$ do
7      $K_q \leftarrow K \cup \{q\}$
8      $\texttt{Cand}_q \leftarrow \texttt{Cand} \cap \Gamma(q)$
9      $\texttt{Fini}_q \leftarrow \texttt{Fini} \cap \Gamma(q)$
10     Tomita($G, K_q, \texttt{Cand}_q, \texttt{Fini}_q$)
11     Cand $\leftarrow \texttt{Cand} - \{q\}$
12     Fini $\leftarrow \texttt{Fini} \cup \{q\}$
13     $K \leftarrow K - \{q\}$
```

Therefore, if Fini is non-empty, the function returns without reporting K as maximal. Otherwise, at each level of the recursion, a $u \in \texttt{Cand} \cup \texttt{Fini}$ with the property that it maximizes the size of $\Gamma(u) \cap \texttt{Cand}$ is selected to be the pivot vertex. The set Ext is formed by removing $\Gamma(\texttt{pivot})$ from Cand. Each $q \in \texttt{Ext}$ is used to extend the current clique K by adding q to K and updating the Cand and Fini sets. These updated sets are then used to recursively call the function. Upon returning, q is removed from Cand and K, and it is added to Fini. This is repeated for each $q \in \texttt{Ext}$.

Using the vertices from Ext instead of Cand to extend the clique prunes paths from the search tree that will not lead to new maximal cliques. The vertices in $\Gamma(\texttt{pivot})$ can be ignored at this level of recursion as they will be considered for extension when processing the recursive call for $K \cup \{\texttt{pivot}\}$ (for a proof see Tomita et al. (2006)).

One of the key points to note about the TTT algorithm is that *no cliques which contain a vertex in* Fini *will be enumerated by the function. PECO* uses this to avoid duplicate enumeration of cliques across reduce tasks.

4.1.3 *PECO*: Parallel Enumeration of Cliques using Ordering

We now provide details of our algorithm. It is assumed that the input is an undirected graph G stored as an adjacency list. For each vertex u, the adjacency list contains the list of vertices adjacent to u. If the input is instead presented as a list of edges, it can be converted into an adjacency list by a single, relatively inexpensive round of Map and Reduce.

Our algorithm consists of a single round of Map and Reduce. Algorithm 20 describes the `map` function of *PECO*. The function takes as input a single line of the adjacency list. Upon reading a vertex v and $\Gamma(v)$, it sends $\langle v, \Gamma(v)\rangle$ to each neighbor of v. This information is enough for the reducer for vertex v to construct the graph G_v.

Algorithm 20: PECO Map(`key`, `value`)

```
Input: key - line number of input file
       value - an adjacency list entry of the form $\langle v, \Gamma(v)\rangle$
1 $v \leftarrow$ first vertex in value
2 $\Gamma(v) \leftarrow$ remaining vertices in value
3 for $u \in \Gamma(v)$ do
4     emit($u, \langle v, \Gamma(v)\rangle$)
```

Algorithm 21 describes the `reduce` function of *PECO*. The reduce task for vertex v receives as input the adjacency list entry for each $u \in \Gamma(v)$, and constructs the induced subgraph G_v. Depending on the ordering selected, the total ordering among vertices in G_v is determined (note that in some cases, generating this total order may itself take an additional MapReduce computation, but this does not change the essence of the algorithm). The reduce task then creates the three sets needed to run `Tomita`: K, the current (not necessarily maximal) clique to extend, begins as $\{v\}$, since this task is only required to output cliques that contain v.

Let $L(v)$ denote the set $\{u \in \Gamma(v) | \texttt{rank}(u) < \texttt{rank}(v)\}$. Note that the reduce task for v should not output any maximal clique that contains a vertex from $L(v)$. One way to do this is to enumerate all maximal cliques in G_v, and filter out those that contain a vertex from $L(v)$. But this can be expensive, and leads to redundant work, as described in [II] above.

Our approach is to add the entire set of vertices in $L(v)$ to the `Fini` set, so that `Tomita` will not search for maximal cliques that contain a vertex from $L(v)$. A subtle point here is that it is not correct to simply delete the vertices $L(v)$ from G_v and search the residual graph, since this will lead to the enumeration of cliques that may not be maximal in G_v, and hence not maximal in G. These steps are described in lines 6-11 of the algorithm below.

```
Algorithm 21: PECO Reduce($v, list(\texttt{value})$)
   Input: $v$ - enumerate cliques containing this vertex
          list(value) - adjacency list entries for each $u \in \Gamma(v)$
1  $G_v \leftarrow$ induced subgraph on vertex set $v \cup \Gamma(v)$
2  rank $\leftarrow$ generated according to ordering selected
3  $K \leftarrow \{v\}$
4  Cand $\leftarrow \Gamma(v)$
5  Fini $\leftarrow \{\ \}$
6  for $u \in \Gamma(v)$ do
7      if rank($u$) $<$ rank($v$) then
8          Cand $\leftarrow$ Cand $- \{u\}$
9          Fini $\leftarrow$ Fini $\cup \{u\}$
10 Tomita($G_v, K,$ Cand, Fini)
```

First we prove the correctness of our Algorithm. We first note that it is easy to verify that the *PECO* map function (Algorithm 20) correctly sends G_v to the reduce task responsible for processing vertex v.

Claim 4.1.1. *The reduce function for vertex v (algorithm 21) enumerates every maximal clique C such that (1) v is contained in C and (2) For every vertex $u \in C$, $\texttt{rank}(v) \leq \texttt{rank}(u)$. Further, no other maximal clique is enumerated by the reduce function for v.*

Proof. Note that G_v and a consistent total order on vertices are correctly received as input by the reducer for v. Let C be a maximal clique that satisfies the above two conditions. Since $v \in C$, the subgraph G_v will contain C. The if statement in line 7 will never evaluate to true for $u \in C$ since v is the smallest vertex in C. Thus, every vertex in C will be in the Cand set when a call to Tomita is made. As a result, v will enumerate C, due to the correctness of the Tomita algorithm. Similarly, it is possible to show that no other maximal clique is output by this reduce function. □

Claim 4.1.2. *PECO (1) Outputs every maximal clique in G. (2) Does not output the same clique more than once. (3) Does not output a non-maximal clique.*

Proof. For (1) and (2). Let ζ be the set of all maximal cliques in G. Consider $C \in \zeta$. From Claim 4.1.1, C is output once by the reducer for vertex v such that V is ranked earliest in the total order among all vertices in C. Further, C is not output by the reducer for any other vertex.

For (3). The reduce task for v has G_v, the subgraph induced by $\{v\} \cup \Gamma(v)$, and outputs all maximal cliques in this subgraph. Consider one such output clique, say C; we claim that C is also maximal in G. The proof is by contradiction; suppose that C was not maximal in G. Then there is a vertex $w \notin C$ that is adjacent to every vertex in C. Then vertex $w \in \Gamma(v)$, and hence w is also present in G_v. This implies that C is not maximal in G_v, which is a contradiction. $\square$

Next, we analyze the communication and memory costs of the algorithm.

Communication The communication cost is equal to the amount of data output by the map tasks, since this data must be sent across the network to the corresponding reduce tasks. Examining the *PECO* map function, it is clear that the adjacency list entry for vertex v will be sent to each vertex in $\Gamma(v)$. Let $\deg(v)$ denote the degree of v. The communication cost due to transmitting the neighbor list of v is proportional to $(\deg(v))^2$. Hence the total communication cost is: $\Theta\left(\sum_{v \in V} (\deg(v))^2\right)$.

One way to reduce communication costs is to divide the graph into fewer subgraphs. In contrast with the current method, which makes as many subproblems as the number of vertices, it is possible to divide the graph into fewer overlapping subgraphs, and still apply a similar technique for each individual subgraph, involving ordering of vertices. This will lead to lesser communication; for instance, if there is only one subproblem, then the total communication is of the order of the number of edges in the graph. We tried this approach of having fewer subproblems. But there were two issues with this approach: (1) the load balance was worse, and (2) there is a higher overhead to construct the vertex ordering. Overall, it performed much worse than our current algorithm.

Memory The map function is trivial, and uses memory equal to the size of a single adjacency list entry, which is of the order of the maximum degree of a vertex in the graph. The reduce function for vertex v requires space equal to the size of the induced subgraph G_v. In the worst case, G_v can be as large as the input graph, if there is a single vertex that is connected to all other vertices. Fortunately, such cases seldom occur with large graphs, and in typical cases, G_v is much smaller.

4.2 Experiments

We ran experiments measuring the performance of *PECO* on a Hadoop cluster. The experiments used real-world graphs from the Stanford large graph database (see Leskovec ()), as well as synthetic random graphs generated according to the Erdös-Rényi model. The test graphs used are given in Table 4.1 along with some basic properties. The soc-sign-epinion Leskovec et al. (2010), soc-epinion Richardson et al. (2003), loc-gowalla Cho et al. (2011), and soc-slashdot0902 Leskovec et al. (2009) graphs are social networks, where vertices represent users and edges represent friendships. cit-patents Hall et al. (2001) is a citation graph for U.S. patents granted between 1975 and 1999. In the wiki-talk graph Leskovec et al. (2010) vertices represent users and edges represent edits to other users' talk pages. web-google Leskovec et al. (2009) is a web graph with pages represented by vertices and hyperlinks by edges. The as-skitter graph Leskovec et al. (2005) is an internet routing topology graph collected from a year of daily traceroutes. For the purpose of clique enumeration, these graphs are all treated as undirected graphs. The wiki-talk-3 and as-skitter-3 graphs are the wiki-talk and as-skitter graphs, respectively, with all vertices of degree less than or equal to 2 removed. Two random graphs are also used in the experiments. UG100k.003 is a random graph with $100,000$ vertices and a probability of 0.003 of an edge being present, while UG1k.3 has $1,000$ vertices and a probability of 0.3. Table 4.2 shows the maximum and average size of the enumerated cliques for every input graph.

The experiments were run on a Hadoop White (2009); Shvachko et al. (2010) cluster with 62 HP DL160 compute nodes each with dual quad core CPUs and 16GB of RAM. Hadoop was configured to use multiple cores so that multiple map or reduce tasks can run in parallel on a single compute node. Note that each reduce task only runs on a single core, so that when we say "10 reduce tasks", the total degree of parallelism (number of cores) in the reduce step is 10. The number of map / reduce tasks that can run on a single compute node can be configured by setting appropriate parameters.

4.2.1 Comparison of Ordering Strategies

In order to compare different ordering strategies, each was run on the set of test graphs. To limit the focus to the quality of the orderings produced, the runtimes of different reduce tasks are examined first, ignoring the map and shuffle phases. The different strategies do not vary in their `map` functions

Table 4.1: Statistics of Test Graphs

Graph	# vertices	# edges	# cliques	Max degree	Avg degree
soc-sign-epinion	131, 580	711, 210	22, 067, 495	3, 558	10.8
loc-Gowalla	196, 591	950, 327	1, 005, 048	14, 730	9.6
soc-slashdot0902	82, 168	504, 231	642, 132	2, 552	12.3
soc-Epinions	75, 879	405, 746	1, 681, 235	3, 044	10.7
web-google	875, 713	4, 322, 051	939, 059	6, 332	9.9
cit-Patents	3, 774, 768	16, 518, 947	6, 061, 991	793	8.8
wiki-talk	2, 294, 385	4, 659, 565	83, 355, 058	100, 029	3.9
wiki-talk-3	626, 749	2, 894, 276	83, 355, 058	46, 257	9.2
as-skitter	1, 696, 415	11, 095, 298	35, 102, 548	35, 455	13.1
as-skitter-3	1, 478, 016	10, 877, 499	35, 102, 548	35, 455	14.7
UG100k.003	100, 000	14, 997, 901	4, 488, 632	380	300
UG1k.30	1, 000	149, 851	15, 112, 753	349	299.7

Table 4.2: Clique Statistics of Test Graphs

Graph	Maximum size of a clique	Average size of a clique
soc-sign-epinion	94	22.7
loc-gowalla	29	7.4
soc-slashdot0902	27	12.3
soc-epinions	23	9
web-google	44	5.7
cit-patent	11	4.2
wiki-talk-3	26	13.8
as-skitter-3	67	21.1
UG100k.003	4	3
UG1k.30	10	5.8

and limiting the focus to only the reduce task will remove the impact of network traffic on the running times. Table 4.3 shows the completion time of the longest running reduce task for each graph and ordering strategy.

It is clear from Table 4.3 that the degree and triangle orderings are superior to the other two strategies, in their overall impact on the reduce times. This is particularly evident on the more challenging graphs such as soc-sign-epinions, wiki-talk-3, and as-skitter-3 where these orderings see a reduction in time of over 50% when compared to the random or lexicographic orderings.

Table 4.3: Completion time (seconds) of the longest reduce task for the combinations of graphs and ordering strategies. "Lex" stands for Lexicographic ordering.

Graph	Degree	Triangle	Random	Lex.
soc-sign-epinions	800	784	1843	1615
loc-gowalla	36	29	45	130
soc-slashdot0902	16	16	23	32
soc-epinions	25	21	32	41
web-google	70	65	86	79
cit-patent	64	59	64	63
wiki-talk-3	823	610	2999	7113
as-skitter-3	2091	2326	14009	> 37052
UG100k.003	226	223	238	269
UG1k.3	103	101	98	107

We note that for graphs where different vertex neighborhoods have a similar structure to each other, the ordering strategy does not matter much. For example, the UG1k.3 graph is a Erdos-Renyi random graph, where different vertices have similar neighborhoods. On such graphs, different subproblems are already of a similar size, and such a graph leads similar reducer runtimes, irrespective of the ordering used. However, on graphs where different neighborhoods are unbalanced, the advantage of the degree and triangle orderings are clear. For example, in the soc-sign-epinions graph, degree and triangle orderings perform much better than lexicographic and random orderings. This graph has different neighborhoods that are unbalanced; to see this, note that the maximum vertex degree is 3558 while the average degree is only 10.8. A similar behavior is observed with the loc-Gowalla graph.

Table 4.4 shows the total run time of the algorithm, (i.e. the total time from start to finish, including all map, shuffle, and reduce phases) for each ordering strategy.

When the pre-processing step is also considered, the triangle ordering no longer performs as well as the degree ordering. This is most evident in the wiki-talk-3 and as-skitter-3 completion times, where the map and shuffle phase contribute to a large portion of the total time. As a result, the degree ordering sees the lowest total running times.

Table 4.4: Total running times (seconds) for different combinations of graphs and ordering strategies. “Lex” stands for Lexicographic ordering.

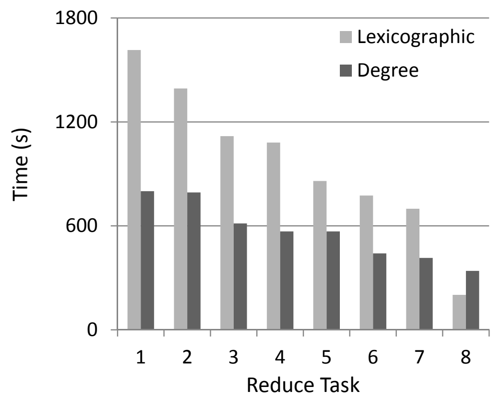
Graph	# Reducers	Degree	Triangle	Random	Lex.
soc-sign-epinions	8	840	828	1875	1646
loc-gowalla	8	122	112	211	280
soc-slashdot0902	8	45	50	52	64
soc-epinions	8	55	53	62	70
web-google	8	126	168	144	140
cit-patent	8	113	150	111	109
wiki-talk-3	32	10667	20465	12229	16647
as-skitter-3	32	8140	17659	20588	> 37052
UG100k.003	8	353	503	376	421
UG1k.3	16	135	129	135	136

4.2.2 Load balancing

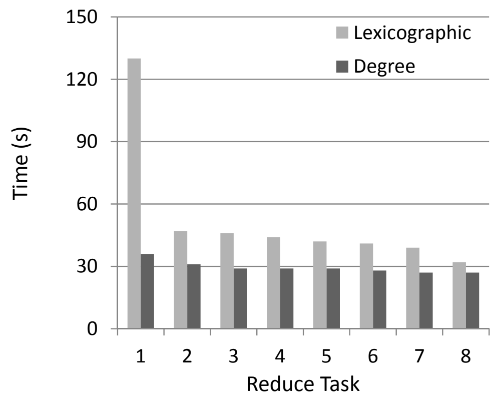
We now present results on the load balancing behavior of different ordering strategies. Figure 4.1 shows the completion times of different reduce tasks for the degree ordering and the lexicographic ordering on the soc-sign-epinion and loc-gowalla graphs, and Figure 4.2 shows the number of maximal cliques enumerated by different reducers, for the degree ordering and the lexicographic ordering strategies. For both sets of experiments, we used 8 reducers.

It is clear that the distribution of work has better load balance with the degree ordering than with lexicographic ordering. For instance, for the soc-sign-epinion graph (Figure 4.2a) we see that reducer 1 emits about 5 million maximal cliques for lexicographical ordering whereas reducer 8 emits less than 500 thousand maximal cliques, only one tenth of the number that reducer 1 emitted. Similarly, for the loc-gowalla graph (Figure 4.2b) with lexicographical ordering, we note that reducer 1 emits approximately 325 thousand maximal cliques whereas reducer 8 emits only about 70 thousand maximal cliques. Such large differences are not observed when degree ordering is used. A similar behavior is observed in Figure 4.1, which shows that the runtimes of different reducers varies widely for the lexicographic ordering strategy but it relatively even for the degree ordering strategy.

Interestingly, degree ordering also leads to a decrease in the total runtime when compared with lexicographic ordering. So the decrease in runtime is a result of two factors, better load balancing and reduced total work. To evaluate the impact of the two factors, we propose the following measures. The



(a) soc-sign-epinion

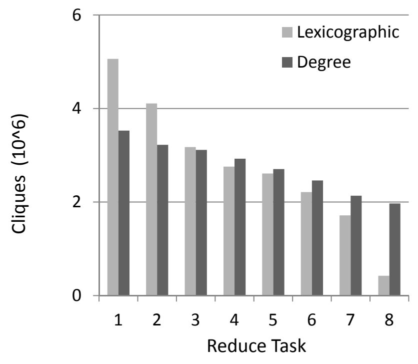


(b) loc-gowalla

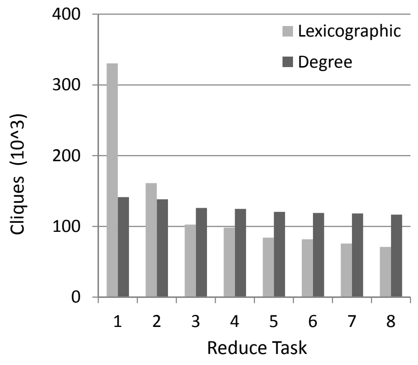
Figure 4.1: A comparison of reduce task completion times between the lexicographic ordering and degree ordering on the soc-sign-epinion and loc-gowalla graphs.

total work for ordering strategy `order` is defined as $T(\texttt{order}) = \sum_{i=1}^{\#\texttt{Tasks}} t_i$, where t_i is the time taken by reducer i.

To measure load balancing, the first step is to normalize the reduce task running times to determine the proportion of the overall work that each task is responsible for. For reduce task i, let $P_i(\texttt{order})$ represent the proportion of overall work i is responsible for when applying ordering `order`, i.e. $P_i(\texttt{order}) = \frac{t_i}{T(\texttt{order})}$, and further for each task i, we define $P(\texttt{order}) = \{P_i(\texttt{order})\}$. Then, one way to measure the load balance of an ordering is by the standard deviation of $P(\texttt{order})$. Let, $L(\texttt{order})$ be the load balance of an ordering, defined as: $L(\texttt{order}) = \texttt{stdev}(P(\texttt{order}))$. Thus, two orderings may have the same load balance but differ in total runtime, because they differ in total work. Alternatively, two orderings may have the same total work, but differ in total runtime, because they differ in load balance.



(a) soc-sign-epinion



(b) loc-gowalla

Figure 4.2: A comparison of the total number of maximal cliques emitted by each reducer for the lexicographic ordering and degree ordering on the soc-sign-epinion and loc-gowalla graphs.

Table 4.3 shows T and L for the degree and lexicographic orderings on the soc-sign-epinions graph. Comparing $T(\texttt{deg})$ and $T(\texttt{lex})$, it is evident that there is a reduction in enumeration time from lexicographic to degree. Similarly, the degree ordering has a smaller L value than lexicographic, indicating a better load balance. Overall, our finding was that on the soc-sign-epinions graph, the degree ordering significantly improves both load balance and enumeration time when compared to the lexicographic ordering. On graph UG1k.3 an improvement is seen in enumeration time, but not in load balancing.

4.2.3 Scalability with Number of Processors

We now present results from experiments on the scalability of *PECO* with increasing numbers of processors. *PECO* is run on several graphs using 1, 2, 4, 8, 16, 32, and 64 reduce tasks. Figures 4.4a and 4.4c show the speedup of the reduce step on the soc-sign-epinions and web-google graphs, respectively. Both graphs show good speedup, with the web-google graph achieving a speedup up of 42 on 64

Task	t_i(`deg`)	P_i(`deg`)
0	646	0.18
1	602	0.17
2	454	0.13
3	442	0.12
4	396	0.11
5	382	0.11
6	348	0.10
7	330	0.09
T, L	3600	0.03

(a) Degree Ordering

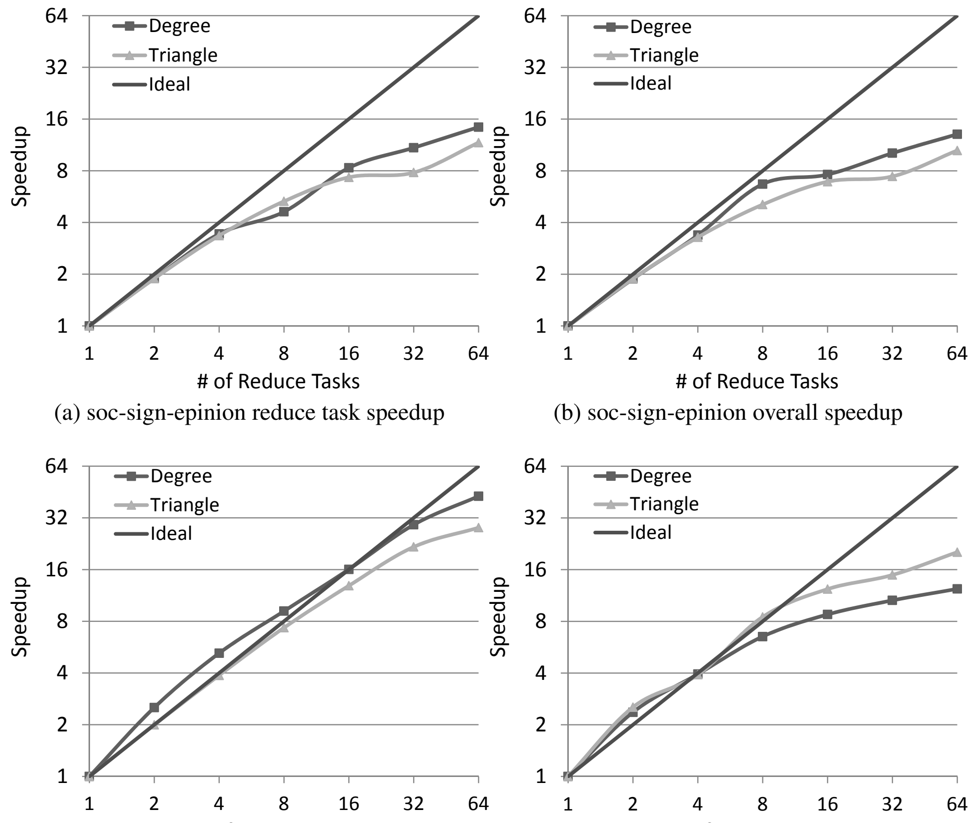
Task	t_i(`lex`)	P_i(`lex`)
0	1210	0.20
1	1081	0.17
2	1010	0.16
3	940	0.15
4	730	0.12
5	643	0.10
6	387	0.16
7	180	0.03
T, L	6181	0.05

(b) Lexicographic Ordering

Figure 4.3: Load balancing and time reduction statistics for the soc-sign-epinions graph.

reduce tasks. Figure 4.4b and 4.4d show the speedup for the two graphs when considering the entire job time. The soc-sign-epinions graph sees little change from the reduce task only graph, as communication costs contribute little to the overall running time. However, the web-google graph sees a larger impact, as the communication cost makes up a larger portion of the total run time.

The soc-sign-epinions and web-google graphs are relatively small when compared to the as-skitter graph. In order to better examine the scalability of both the communication and enumeration aspects of the algorithm, the speedup of the degree ordering is examined on the as-skitter-3 graph. Figure 4.5a shows the speedup of just the reduce tasks. A speedup of 22 with 64 reduce tasks is achieved. When the running time of the entire job is considered (Figure 4.5b), the speedup increases slightly to 24, demonstrating that the communication aspect of the algorithm scales well on large graphs. Figure 4.5c compares the completion times of the reduce phase to those of the overall running time.



(a) soc-sign-epinion reduce task speedup (b) soc-sign-epinion overall speedup

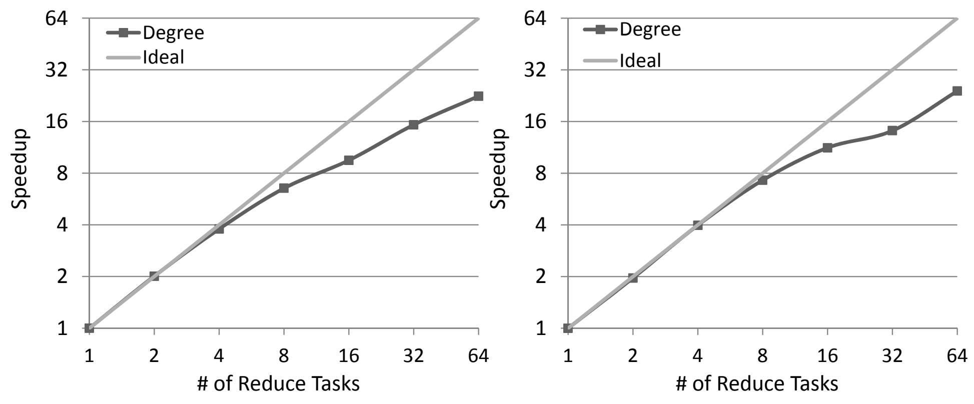
(c) web-google reduce task speedup (d) web-google overall speedup

Figure 4.4: *PECO* scalability for the degree and triangle ordering strategies.

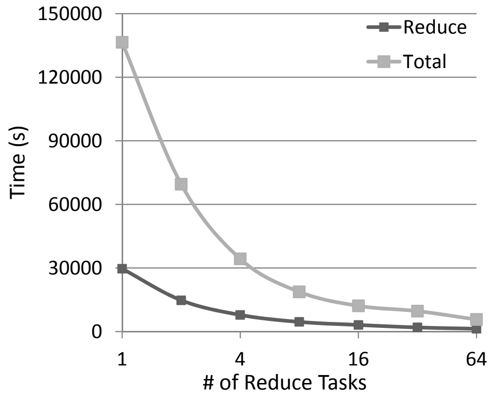
4.2.4 Comparison with Prior Work

We implemented the WYZW Algorithm, a parallel algorithm for MCE designed for MapReduce due to Wu et al. (2009), and compared its performance with that of *PECO*. The WYZW Algorithm divides the input graph into many subgraphs and independently processes each subgraph to enumerate cliques. This can emit non-maximal cliques which have to be filtered away by an additional post-processing step. For these experiments, we used a smaller Hadoop cluster with 50 compute nodes, each a Quad-Core AMD Opteron(tm) Processor 2354 and 8 GB of RAM.

Figure 4.6 shows the runtimes of *PECO* and WYZW when the number of reducers vary from 20 till 120. On the soc-epinions graph with 120 reducers, *PECO* runs approximately 127 times faster than WYZW, while it is about 42 times faster on 20 reducers, while using the same underlying sequential



(a) as-skitter reduce task speedup (b) as-skitter overall speedup

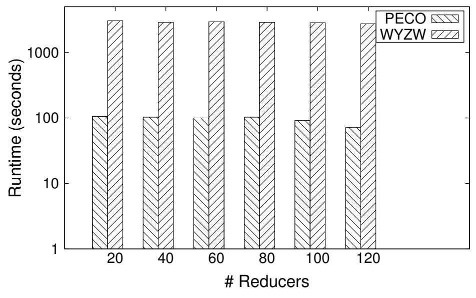


(c) as-skitter reduce and total running times

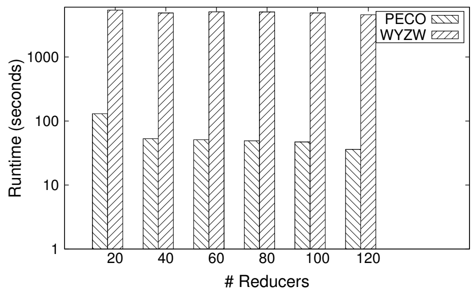
Figure 4.5: *PECO* large graph scalability

algorithm for both WYZW and *PECO*. The advantage of *PECO* over WYZW increases with the degree of parallelism (number of reducers). One of the reasons for this is the better load balancing of *PECO*, which allows it to use more processors more effectively. With WYZW, there were a few reducers that ran for a long time while the others finished quickly, and the runtime did not get significantly better as the number of reducers was increased.

A similar result was observed with the loc-gowalla graph (see Figure 4.6). Using 120 reducers, other input graphs that were processed by *PECO* include web-google (58 sec for *PECO*), wiki-talk-3 (1013 sec), and as-skitter3 (5930 sec). But none of these were completed by WYWZ within 6 hours.



(a) loc-gowalla



(b) soc-epinions

Figure 4.6: Runtime analysis of *PECO* vs *WYZW* for different reducer values on various input graphs. Note that the plot is in logarithmic scale.

CHAPTER 5. MINING MAXIMAL CLIQUES FROM UNCERTAIN GRAPHS

5.1 Number of Maximal Cliques

The maximum number of maximal cliques in a deterministic graph on n vertices is known exactly due to a result by Moon and Moser (1965). If $n \mod 3 = 0$, this number is $3^{\frac{n}{3}}$. If $n \mod 3 = 1$, then it is $4 \cdot 3^{\frac{n-4}{3}}$, and if $n \mod 3 = 2$, then it is $2 \cdot 3^{\frac{n-2}{3}}$. The graphs that have the maximum number of maximal cliques are known as Moon-Moser graphs.

For uncertain cliques, no such bound was known so far. In this section, we establish a bound on the maximum number of α-maximal cliques in an uncertain graph. For $0 < \alpha < 1$, let $f(n, \alpha)$ be the maximum number of α-maximal cliques in any uncertain graph with n nodes, without any assumption about the assignments of edge probabilities. The following theorem is the main result of this section.

Theorem 1. *Let $n \geq 2$, and $0 < \alpha < 1$. Then: $f(n, \alpha) = \binom{n}{\lfloor n/2 \rfloor}$*

Proof. We can easily verify that the theorem holds for $n = 2$. for $n \geq 3$, let $g(n) = \binom{n}{\lfloor n/2 \rfloor}$. We show $f(n, \alpha)$ is at least $g(n)$ in Lemma 7, and then show that $f(n, \alpha)$ is no more than $g(n)$ in Lemma 8. $\square$

Lemma 7. *For any $n \geq 3$, and any $\alpha, 0 < \alpha < 1$, there exists an uncertain graph $\mathcal{G} = (V, E, p)$ with n nodes which has $g(n)$ α-maximal cliques.*

Proof. First, we assume that n is even. Consider $\mathcal{G} = (V, E, p)$, where $E = V \times V$. Let $\kappa = \binom{n/2}{2}$. For each $e \in E$, let $p(e) = q$ where $q^{\kappa} = \alpha$. We have $0 < q < 1$ since $0 < \alpha < 1$. Let S be an arbitrary subset of V such that $|S| = n/2$. We can verify that S is an α-maximal clique since (1) the probability that S is a clique is $q^{\kappa} = \alpha$ and (2) for any set $S' \supsetneq S, S' \subseteq V$, the probability that S' is a clique is at most $qq^{\kappa} = q\alpha < \alpha$. We can also observe that for any subset $S \subseteq V$, S cannot be an α-maximal clique if $|S| < n/2$ or $|S| > n/2$. Thus we conclude that a subset $S \subseteq V$ is an α-maximal clique iff $|S| = n/2$ which implies that the total number of α-maximal cliques in $\mathcal{G}$ is $\binom{n}{n/2}$. A similar proof applies when n is odd. □

Note that our construction in the Lemma above employs the condition that $n \geq 3$ and $0 < \alpha < 1$. When $\alpha = 1$, the upper bound is from the result of Moon and Moser for deterministic graphs, and in this case $f(n, \alpha) = 3^{\frac{n}{3}}$ and is smaller than $g(n)$. Next we present a useful definition required for proving the next Lemma.

Definition 9. *A collection of sets $\mathcal{C}$ is said to be non-redundant if for any pair $S_1, S_2 \in \mathcal{C}$, $S_1 \neq S_2$, we have $S_1 \nsubseteq S_2$ and $S_2 \nsubseteq S_1$.*

Lemma 8. *$g(n)$ is an upper bound on $f(n, \alpha)$.*

Proof. Let $\mathcal{C}^\alpha(\mathcal{G})$ be the collection of all α-maximal cliques in $\mathcal{G}$. Note that by the definition of α-maximal cliques, any α-maximal clique S in $\mathcal{G}$ can not be a proper subset of any other α-maximal clique in $\mathcal{G}$. Thus from Definition 9, for any uncertain graph $\mathcal{G}$, $\mathcal{C}^\alpha(\mathcal{G})$ is a non-redundant collection. Hence, it is clear that the largest number of α-maximal cliques in $\mathcal{G}$ should be upper bounded by the size of a largest non-redundant collection of subsets of V.

Let $\mathcal{C}$ be the collection of all subsets of V. Based on $\mathcal{C}$, we construct such an undirected graph $\widehat{G} = (\mathcal{C}, \widehat{E})$ where for any two nodes $S_1 \in \mathcal{C}, S_2 \in \mathcal{C}$, there is an edge connecting S_1 and S_2 iff $S_1 \subseteq S_2$ or $S_2 \subseteq S_1$. It can be verified that a sub-collection $\mathcal{C}' \subseteq \mathcal{C}$ is a non-redundant iff $\mathcal{C}'$ is an independent set in $\widehat{G}$. In Lemma 9, we show that $g(n)$ is the size of a largest independent set of $\widehat{G}$, which implies that $g(n)$ is an upper bound for the number of α-maximal cliques in $\mathcal{G}$. □

Let $\mathcal{C}^*$ be a largest independent set in $\widehat{G}$. Also, let $\mathcal{C}_k \subseteq \mathcal{C}, 0 \leq k \leq n$ be the collection of subsets of V with the size of k. Observe that for each $0 \leq k \leq n$, $\mathcal{C}_k$ is an independent set of $\widehat{G}$. Also let $L(n)$ and $U(n)$ be respectively the minimum and maximum size of sets in $\mathcal{C}^*$. We can show that $L(n)$ and $U(n)$ can be bounded as shown in Lemma 10 and Lemma 11 respectively.

Lemma 9. *For any $n \geq 3$, $|\mathcal{C}^*| = g(n)$.*

Proof. We first consider the case when n is even. By Lemmas 10 and 11, we know $n/2 \leq L(n) \leq U(n) \leq n/2$. Thus we have $L(n) = U(n) = n/2$ which implies $\mathcal{C}^* = \mathcal{C}^*_{n/2}$. Recall that $\mathcal{C}_k \subseteq \mathcal{C}, 0 \leq k \leq n$ is the collection of subsets of V with the size of k.

We have (1) $\mathcal{C}^* = \mathcal{C}^*_{n/2} \subseteq \mathcal{C}_{n/2}$ and (2) $|\mathcal{C}^*| \geq |\mathcal{C}_{n/2}|$ since $\mathcal{C}^*$ is a largest independent set of $\widehat{G}$. Thus we conclude $\mathcal{C}^* = \mathcal{C}_{n/2}$ which has the size of $\binom{n}{(n/2)} = g(n)$.

We next consider the case when n is odd. From Lemmas 10 and 11, we know $(n-1)/2 \leq L(n) \leq U(n) \leq (n+1)/2$. Thus we have $\mathcal{C}^* = \mathcal{C}^*_{(n-1)/2} \bigcup \mathcal{C}^*_{(n+1)/2}$. For notation convenience, we set $n_1 = (n-1)/2, n_2 = (n+1)/2$. Let $\widehat{G}(\mathcal{C}_{n_1}, \mathcal{C}_{n_2})$ be the subgraph of $\widehat{G}$ induced by $\mathcal{C}_{n_1} \cup \mathcal{C}_{n_2}$. We can view $\widehat{G}(\mathcal{C}_{n_1}, \mathcal{C}_{n_2})$ as a bipartite graph with two disjoint vertex sets $\mathcal{C}_{n_1}$ and $\mathcal{C}_{n_2}$ respectively. Observe that $\mathcal{C}^*_{n_1} \subseteq \mathcal{C}_{n_1}$ and $\mathcal{C}^*_{n_2} \subseteq \mathcal{C}_{n_2}$. Let $\widehat{E}(\mathcal{C}^*_{n_1})$ be the set of edges induced by $\mathcal{C}^*_{n_1}$ in $\widehat{G}(\mathcal{C}_{n_1}, \mathcal{C}_{n_2})$. Since $\mathcal{C}^*$ is an independent set of $\widehat{G}$, none of the edges in $\widehat{E}(\mathcal{C}^*_{n_1})$ will have an end in a node of $\mathcal{C}^*_{n_2}$, i.e, all the edges of $\widehat{E}(\mathcal{C}^*_{n_1})$ should have an end falling in $\mathcal{C}_{n_2} \setminus \mathcal{C}^*_{n_2}$. Note that in $\widehat{G}(\mathcal{C}_{n_1}, \mathcal{C}_{n_2})$, all nodes have a degree of n_2. Thus we have:

$$|\widehat{E}(\mathcal{C}^*_{n_1})| = |\mathcal{C}^*_{n_1}| * n_2 \leq |\mathcal{C}_{n_2} \setminus \mathcal{C}^*_{n_2}| * n_2 = (|\mathcal{C}_{n_2}| - |\mathcal{C}^*_{n_2}|) * n_2$$

from which we obtain $|\mathcal{C}^*| = |\mathcal{C}^*_{n_1}| + |\mathcal{C}^*_{n_2}| \leq |\mathcal{C}_{n_2}| = \binom{n}{n_2}$. Note that $\mathcal{C}_{n_2}$ itself is an independent set of $\widehat{G}$ with size $\binom{n}{n_2}$. Thus we conclude that $|\mathcal{C}^*| = \binom{n}{n_2} = g(n)$. □

Lemma 10. $L(n) \geq \lfloor n/2 \rfloor$

Proof. Let us assume n is an even number. We prove by contradiction as follows. Suppose $L(n) = \ell \leq n/2 - 1$. Let $\mathcal{C}^*_k \subseteq \mathcal{C}^*, L(n) \leq k \leq U(n)$ be the collection of all sets in $\mathcal{C}^*$ which has the size of k, i.e, $\mathcal{C}^*_k = \{S \in \mathcal{C}^* || S| = k\}$. In the following we construct a new collection $\mathcal{C}_{new} \subseteq \mathcal{C}$ which proves to be an independent set in $\widehat{G}$ with the size being strictly larger than $\mathcal{C}^*$. For each $S \in \mathcal{C}^*_\ell$, we add to $\mathcal{C}^*$ all subsets of V which has the form as $S \cup \{i\}$ where $i \in V \setminus S$ and remove S from $\mathcal{C}^*$ meanwhile. Let $\mathcal{C}_{new}$ be the collection obtained after we process the same route for all $S \in \mathcal{C}^*_\ell$. Mathematically, we have: $\mathcal{C}_{new} = \mathcal{C}_1 \bigcup \mathcal{C}_2$ where $\mathcal{C}_1 = \bigcup_{S \in \mathcal{C}^*_\ell} \bigcup_{i \in V \setminus S} \{S \cup \{i\}\}, \mathcal{C}_2 = \mathcal{C}^* \setminus \mathcal{C}^*_\ell$. First we show $\mathcal{C}_{new}$ is an independent set of $\widehat{G}$. Arbitrarily choose two distinct sets, say $S_1 \in \mathcal{C}_{new}, S_2 \in \mathcal{C}_{new}, S_1 \neq S_2$. We check all the possible cases one by one:

- $S_1 \in \mathcal{C}_1, S_2 \in \mathcal{C}_1$. We observe that $|S_1| = |S_2| = \ell + 1$ and $S_1 \neq S_2$. Thus no inclusion relation could exist between S_1 and S_2.

- $S_1 \in \mathcal{C}_2, S_2 \in \mathcal{C}_2$. In this case no inclusion relation can exist between S_1 and S_2 since $\mathcal{C}_2$ is an independent set of $\widehat{G}$.

- $S_1 \in \mathcal{C}_1, S_2 \in \mathcal{C}_2$. Since $\mathcal{C}_\ell^*$ is the collection of sets in $\mathcal{C}^*$ which has the smallest size ℓ, we get that $|S_2| \geq \ell + 1 = |S_1|$. Therefore there is only one possible inclusion relation existing here, that is $S_1 \subset S_2$. Suppose $S_1 = S_1' \cup \{i_1\} \subset S_2$ for some $S_1' \in \mathcal{C}_\ell^*$. Thus we get that $S_1' \subset S_2$ which implies $\mathcal{C}^*$ is not an independent set of $\widehat{G}$. Hence we conclude that no inclusion relation could exist between S_1 and S_2.

Summarizing the analysis above, we get that no inclusion relation could exist between S_1 and S_2 which yields $\mathcal{C}_{new}$ is an independent set of $\widehat{G}$.

Now we prove that $|\mathcal{C}_{new}| > |\mathcal{C}^*|$. Observe that $\mathcal{C}_1$ and $\mathcal{C}_2$ are disjoint from each other; otherwise $\mathcal{C}^*$ is not an independent set. So we have $|\mathcal{C}_{new}| = |\mathcal{C}_1| + |\mathcal{C}_2|$. Note that $|\mathcal{C}^*| = |\mathcal{C}_\ell^*| + |\mathcal{C}_2|$ since $\mathcal{C}^*$ is the union of the two disjoint parts $\mathcal{C}_\ell^*$ and $\mathcal{C}_2$. Therefore $|\mathcal{C}_{new}| > |\mathcal{C}^*|$ is equivalent to $|\mathcal{C}_1| > |\mathcal{C}_\ell^*|$. Let $\widehat{G}(\mathcal{C}_\ell^*, \mathcal{C}_1)$ be the induced subgraph graph of $\widehat{G}$ by $\mathcal{C}_\ell^* \bigcup \mathcal{C}_1$. Note that $\widehat{G}(\mathcal{C}_\ell^*, \mathcal{C}_1)$ can be viewed as a bipartite graph where the two disjoint vertex sets are $\mathcal{C}_\ell^*$ and $\mathcal{C}_1$ respectively. In $\widehat{G}(\mathcal{C}_\ell^*, \mathcal{C}_1)$ we observe that (1) for each node $S_1 \in \mathcal{C}_\ell^*$, its degree $d(S_1) = n - \ell$; (2) for each node $S_2 \in \mathcal{C}_1$, its degree $d(S_2) \leq \ell + 1$. Thus we get that $|\widetilde{E}| = |\mathcal{C}_\ell^*|(n-\ell) \leq |\mathcal{C}_1|(\ell+1)$. According to our assumption we have $\ell \leq n/2 - 1$. Thus we have
$|\mathcal{C}_\ell^*|/|\mathcal{C}_1| \leq (\ell+1)/(n-\ell) \leq (n/2)/(n/2+1) < 1$, yielding $|\mathcal{C}_\ell^*| < |\mathcal{C}_1|$ which is equivalent to $|\mathcal{C}^*| < |\mathcal{C}_{new}|$.

So far we have successfully constructed a new collection $\mathcal{C}_{new} \subseteq \mathcal{C}$ such that (1) it is an independent set of $\widehat{G}$ and (2) $|\mathcal{C}_{new}| > |\mathcal{C}^*|$. That contradicts with the fact that $\mathcal{C}^*$ is a largest independent set of $\widehat{G}$. Thus our assumption $\ell \leq n/2 - 1$ does not hold, which yields $\ell \geq n/2$. For the case when n is odd, we can process essentially the same analysis as above and get $\ell \geq (n-1)/2$. □

Lemma 11. $U(n) \leq \lceil n/2 \rceil$

Proof. Let us assume n is an even number. Based on $\mathcal{C}^*$, we construct a dual collection $\mathcal{C}_{dual}^*$ as follows: Initialize $\mathcal{C}_{dual}^*$ as an empty collection. For each $S \in \mathcal{C}^*$, we add $V \setminus S$ into $\mathcal{C}_{dual}^*$. Mathematically, we have: $\mathcal{C}_{dual}^* = \bigcup_{S \in \mathcal{C}^*} \{V \setminus S\}$. First we show $\mathcal{C}_{dual}^*$ is an independent set of $\widehat{G}$. Arbitrarily choose two distinct sets, say $V \setminus S_1 \in \mathcal{C}_{dual}^*, V \setminus S_2 \in \mathcal{C}_{dual}^*$, where $S_1 \in \mathcal{C}^*, S_2 \in \mathcal{C}^*, S_1 \neq S_2$. Note that

$$V \setminus S_1 \subset V \setminus S_2 \Leftrightarrow S_1 \supset S_2, V \setminus S_2 \subset V \setminus S_1 \Leftrightarrow S_2 \supset S_1$$

Thus we have that no inclusion relation could exist between $V \setminus S_1$ and $V \setminus S_2$ since no inclusion relation exists between S_1 and S_2 resulting from the fact that $\mathcal{C}^*$ is an independent set of $\widehat{G}$. So we get $\mathcal{C}^*_{dual}$ is an independent set as well.

We can verify that $|\mathcal{C}^*_{dual}| = |\mathcal{C}^*|$. Therefore we can conclude $\mathcal{C}^*_{dual}$ is a largest independent set of $\widehat{G}$. By Lemma 10, we get to know the minimum size of sets in $\mathcal{C}^*_{dual}$ should be at least $n/2$, which yields the maximum size of of sets in $\mathcal{C}^*$ should be at most $n/2$. For the case when n is odd, we can analyze essentially the same as above. □

5.2 Enumeration Algorithm

In this section, we present **MULE** (Maximal Uncertain cLique Enumeration), an algorithm for enumerating all α-maximal cliques in an uncertain graph $\mathcal{G}$, followed by a proof of correctness and an analysis of the runtime. We assume that $\mathcal{G}$ has no edges e such that $p(e) < \alpha$. If there are any such edges, they can be pruned away without losing any α-maximal cliques, using Observation 3. Let the vertex identifiers in $\mathcal{G}$ be $1, 2, \ldots, n$. For clique C, let $\max(C)$ denote the largest vertex in C. For ease of notation, let $\max(\emptyset) = 0$, and let $clq(\emptyset, \mathcal{G}) = 1$.

Intuition We first describe a basic approach to enumeration using depth-first-search (DFS) with backtracking. The algorithm starts with a set of vertices C (initialized to an empty set) that is an α-clique and incrementally adds vertices to C, while retaining the property of C being an α-clique, until we can add no more vertices to C. At this point, we have an α-maximal clique. Upon finding a clique that is α-maximal, the algorithm backtracks to explore other possible vertices that can be used to extend C, until all possible search paths have been explored. To avoid exploring the same set C more than once, we add vertices in increasing order of the vertex id. For instance, if C was currently the vertex set $\{1, 3, 4\}$, we do not consider adding vertex 2 to C, since the resulting clique $\{1, 2, 3, 4\}$ will also be reached by the search path by adding vertices $1, 2, 3, 4$ in that order.

MULE improves over the above basic DFS approach in the following ways. First, given a current α-clique C, the set of vertices that can be added to extend C includes only those vertices that are already connected to every vertex within C. Instead of considering every vertex that is greater than $\max(C)$, it is more efficient to track these vertices as the recursive algorithm progresses – this will save the effort of needing to check if a new vertex v can actually be used to extend C. This leads us to incrementally track vertices that can still be used to extend C.

Second, note that not all vertices that extend C into a clique preserve the property of C being an α-clique. In particular, adding a new vertex v to C decreases the clique probability of C by a factor equal to the product of the edge probabilities between v and every vertex in C. So, in considering vertex v for addition to C, we need to compute the factor by which the clique probability will fall. This computation can itself take $\Theta(n)$ time since the size of C can be $\Theta(n)$, and there can be $\Theta(n)$ edges to consider in adding v. A key insight is to reduce this time to $O(1)$ by incrementally maintaining this factor for each vertex v still under consideration. The recursive subproblem contains, in addition to current clique C, a set I consisting of pairs (u, r) such that $u > \max(C)$, u can extend C into an α-clique, and adding u will multiply the clique probability of C by a factor of r. This set I is incrementally maintained and supplied to further recursive calls.

Finally, there is the cost of checking maximality. Suppose that at a juncture in the algorithm we found that I was empty, i.e. there are no more vertices greater than $\max(C)$ that can extend C into an α-clique. This does not yet mean that C is an α-maximal clique, since it is possible there are vertices less than $\max(C)$, but not in C, which can extend C to an α-maximal clique (note that such an α-maximal clique will be found through a different search path). This means that we have to run another check to see if C is an α-maximal clique. Note that even checking if a set of vertices C is an α-maximal clique can be a $\Theta(n^2)$ operation, since there can be as many as $\Theta(n)$ vertices to be potentially added to C, and $\Theta(n^2)$ edge interactions to be considered. We reduce the time for searching such vertices by maintaining the set X of vertices that can extend C, but will be explored in a different search path. By incrementally maintaining probabilities with vertices in I and X, we can reduce the time for checking maximality of C to $\Theta(n)$.

MULE incorporates the above ideas and is described in Algorithm 22.

5.2.1 Proof of Correctness

In this section we prove the correctness of MULE.

Theorem 2. *MULE (Algorithm 22) enumerates all α-maximal cliques from an input uncertain graph $\mathcal{G}$.*

Algorithm 22: MULE($\mathcal{G}, \alpha$)

```
Input: $\mathcal{G}$ is the input uncertain graph,
       $\alpha, 0 < \alpha < 1$ is the user provided probability threshold
1 $\hat{I} \leftarrow \emptyset$ ;
2 forall the $u \in V$ do
3     $\hat{I} \leftarrow \hat{I} \cup \{(u, 1)\}$
4 Enum-Uncertain-MC($\emptyset$, 1 ,$\hat{I}$, $\emptyset$) ;
```

Algorithm 23: Enum-Uncertain-MC(C, q, I, X)

```
Input: We assume $\mathcal{G}$ and $\alpha$ are available as immutable global variables
       $C$ is the current Uncertain Clique being processed
       $q = clq(C, \mathcal{G})$, maintained incrementally
       $I$ is a set of all tuples $(u, r)$, such that $\forall (u, r) \in I$, $u > max(C)$, and
       $clq(C \cup \{u\}, \mathcal{G}) = q \cdot r \geq \alpha$, i.e. $C \cup \{u\}$ is an $\alpha$-clique in $\mathcal{G}$
       $X$ is a set of all tuples $(v, s)$, such that $\forall (v, s) \in X$, $v \notin C$, $v < max(C)$, and
       $clq(C \cup \{v\}, \mathcal{G}) = q \cdot s \geq \alpha$ , i.e. $C \cup \{v\}$ is an $\alpha$-clique in $\mathcal{G}$
1 if $I = \emptyset$ and $X = \emptyset$ then
2     Output $C$ as $\alpha$-maximal clique ;
3     return
4 forall the $(u, r) \in I$ considered in increasing order of $u$ do
5     $C' \leftarrow C \cup \{u\}$ // Lemma 12:   $m = max(C') = u$
6     $q' \leftarrow q \cdot r$ //  $clq(C \cup \{v\}, \mathcal{G})$
7     $I' \leftarrow GenerateI(C', q', I)$ ;
8     $X' \leftarrow GenerateX(C', q', X)$ ;
9     Enum-Uncertain-MC($C', q', I', X'$) ;
10    $X \leftarrow X \cup \{(u, r)\}$
```

Proof. To prove the theorem we need to show the following. First, if C is a clique emitted by Algorithm 22, then C must be an α-maximal clique. Next, if C is an α-maximal clique, then it will be emitted by Algorithm 22. We prove them in Lemmas 14 and 15 respectively. $\square$

Before proving Lemmas 14 and 15, we prove some properties of Algorithm 23.

Lemma 12. *When Algorithm 23 is called with C' in line 9, I' is a set of all tuples $(u'r')$, where $u' \in V$ and $0 < r' \leq 1$, such that, $\forall (u', r') \in I'$, $u' > max(C')$, and $clq(C' \cup \{u'\}, \mathcal{G}) = q' \cdot r' \geq \alpha$, i.e. $C' \cup \{u'\}$ is an α-clique in $\mathcal{G}$.*

Proof. Let $u' \in V$ be a vertex such that (1) $u' > \max(C')$, and (2) $C' \cup \{u'\}$ is an α-clique in $\mathcal{G}$. We need to show that $(u', r') \in I'$ such that $clq(C' \cup \{u'\}, \mathcal{G}) = q' \cdot r'$.

Algorithm 24: GenerateI(C', q', I)

```
    Input: We assume $\mathcal{G}$ and $\alpha$ are available as immutable global variables
 1  $m \leftarrow max(C')$, $I' \leftarrow \emptyset$, $S \leftarrow \emptyset$ ;
 2  forall the $(u, r) \in I$ do
 3      $S \leftarrow S \cup \{u\}$
 4  $S \leftarrow S \cap \{\Gamma(m)\}$
 5  forall the $(u, r) \in I$ do
 6      if $u > m$ and $u \in S$ then
 7          $clq(C' \cup \{u\}, \mathcal{G}) \leftarrow q' \cdot r \cdot p(\{u, m\})$
 8          if $(clq(C' \cup \{u\}, \mathcal{G})) \geq \alpha$ then
 9              $u' \leftarrow u$ ;
10              $r' \leftarrow r \cdot p(\{u, m\})$ ;
11              $I' \leftarrow I' \cup \{(u', r')\}$
12  return I'
```

Let C' be a clique being called by Enum-Uncertain-MC with I'. Note that each call of the method adds one vertex $u \in I$ to the current clique C such $u > \max(C)$. Since the vertices are added in the lexicographical ordering, there is an unique sequence of calls to the method Enum-Uncertain-MC such that we reach a point in execution of Algorithm 23 where Enum-Uncertain-MC is called with C'. We call this sequence of calls as Call-0, Call-1, . . ., Call-$|C'|$. Also, let C_i be the clique used by method Enum-Uncertain-MC during Call-i.

We prove by induction. First consider the base case. For that consider the first call made to Algorithm 23, i.e. Call-0. We know that C is initialized as $\emptyset$. During the first call made, all vertices in V satisfy conditions (1) and (2). This is because, first $max(\emptyset) = 0$. Second any single vertex can be considered as a clique with probability 1. $\hat{I}$ is initialized such that all r in $\hat{I}$ are $1 \geq \alpha$. Thus for all u such that $(u, r) \in \hat{I}$, $u > \max(C)$. This proves the base case.

For the inductive step, consider a recursive call to the method Call-i which calls Call-$(i + 1)$. For every case expect initialization, I' is generated from I by line 7 of Algorithm 23 which in turn calls Algorithm 24. In Algorithm 24, only vertices in I that are greater than C' are added to I'. Thus all vertices in I that satisfy (1) are added to I'. Next every vertex in I is connected to C. We need to show that all vertices in I' are connected to C'. In line 4 of Algorithm 24, we prune out any vertex in I that is not connected to $m = \max(C')$. Assume that u' extends C such that $clq(C \cup \{u'\}, \mathcal{G}) = r$. Now let $c = \{C' \setminus C\}$. Note that c is a single vertex. Also, assume $u' > c$. From line 4, we know that $q' \cdot r' \geq \alpha$

Algorithm 25: GenerateX(C', q', X)

```
Input: We assume $\mathcal{G}$ and $\alpha$ are available as immutable global variables
1  $m \leftarrow max(C')$, $X' \leftarrow \emptyset$, $S \leftarrow \emptyset$ ;
2  forall the $(v, s) \in I$ do
3      $S \leftarrow S \cup \{v\}$
4  $S \leftarrow S \cap \{\Gamma(m)\}$
5  forall the $(v, s) \in X$ do
6      if $v \in S$ then
7          $clq(C' \cup \{v\}, \mathcal{G}) \leftarrow q' \cdot s \cdot p(\{v, m\})$
8          if $(clq(C' \cup \{v\}, \mathcal{G}) \geq \alpha$ then
9              $v' \leftarrow v$ ;
10             $s' \leftarrow s \cdot p(\{v, m\})$ ;
11             $X' \leftarrow X' \cup \{(v', s')\}$
12 return X'
```

Also from line 6 of Algorithm 24, $r' = r \cdot p(\{c, u'\})$. Now $clq(C' \cup \{u'\}, \mathcal{G}) = q' \cdot r \cdot p(\{c, u'\}) = q' \cdot r'$, Now in line 8 of Algorithm 24 we add u' to I' only if $r' \geq \alpha$ thus proving the inductive step. □

The following observation follows from Lemma 12.

Observation 5. *The input C to Algorithm 23 is an α-clique.*

Lemma 13. *When Algorithm 23 is called with C' in line 9, X' is a set of all tuples (v', s'), where $v' \in V$ and $0 < s' \leq 1$, such that, $\forall (v', s') \in X'$, we have $v' \notin C'$, $v' < \max(C')$, and $(clq(C' \cup \{v'\}, \mathcal{G}) = q' \cdot s') \geq \alpha$, i.e. $C' \cup \{v'\}$ is an α-clique in $\mathcal{G}$.*

Proof. Let $m = \max(C')$ and $C = C' \setminus \{m\}$. Since Algorithm 23 was called with C', it must have been called with C. This is because the working clique is always extended by adding vertices from I, and from Lemma 12, I only contains vertices that are greater than the maximum vertex in C. Let X be the corresponding set of tuples used when the call was made to Enum-Uncertain-MC with C. Let $u > \max(C)$ be a vertex such that $clq(C' \cup \{u\}, \mathcal{G}) \geq \alpha$ and $u < m$. Note that $u \notin C'$, $u < \max(C')$, and $C' \cup \{u\}$ is an α-clique in $\mathcal{G}$. This means u satisfies all conditions for $u \in X'$. We need to show that when Enum-Uncertain-MC is called with C', the generated X' which is passed in Enum-Uncertain-MC contains u.

Firstly, note that since $C' \cup \{u\}$ is α-clique in $\mathcal{G}$, we have $clq(C \cup \{u\}, \mathcal{G}) \geq \alpha$ (from Observation 2). Since $u > \max(C)$ and $clq(C \cup \{u\}, \mathcal{G}) \geq \alpha$, from Lemma 12, u will be used in line 4 to call Enum-

Uncertain-MC using $C \cup \{u\}$. Once this call is returned, u is added to X in line 10. Note that since the loop at line 4 add vertices in lexicographical order, m will be added to C after u. Thus u will be in X, when m is used to extend C. Next we show that if $u \in X$, after execution of line 8, $u \in X'$. We prove this as follows. Note that Algorithm 25 is used to generate X' from X. Note that X' is generated by Algorithm 25 by selectively adding vertices from X. A vertex is added to X' from X, only if $C' \cup \{u\}$ is α-clique in $\mathcal{G}$. From our initial assumptions, we know that u satisfies this condition and is hence added to X' and passed on to Enum-Uncertain-MC when it is called with C'.

Now let us consider v, such that v does not satisfy all the conditions for $v \in X'$. We need to show that $v \notin X'$. There are two cases. First, when $v \notin X$. This case is trivial as X' is constructed from X and hence if $v \notin X$, $v \notin X'$. For the second case, when $v \in X$, we need to show that v will not be added to X' in line 8 of Algorithm 23. Note that since $v \in X$, we know $v \notin C'$ and $v < \max(C')$. Thus, it must be that $C \cup \{m, v\}$ is not an α-clique in $\mathcal{G}$. Algorithm 25 will add v to X' only if $C \cup \{m, v\}$ is α-clique in $\mathcal{G}$. But from our previous discussion, we know that this condition doesn't hold. Hence, v will not be added to X'. Thus only vertices that satisfy all three conditions are in X'. □

Lemma 14. *Let C be a clique emitted by Algorithm 23. Then C is an α-maximal clique.*

Proof. Algorithm 23 emits C in Line 2. From Observation 5, we know that C is an α-clique. We need to show that C is α-maximal. We use proof by contradiction. Suppose C is non-maximal. This means that there exists a vertex $u \in V$, such that $C \cup \{u\}$ is an α-clique. We know that $I = \emptyset$ when C is emitted. From Lemma 12, we know that there exists no vertex $u \in V$ such that $u > max(C)$ that can extend C. Again, we know that $X = \emptyset$ when C is emitted. Thus from Lemma 13, we know that there exists no vertex $v \in V$ such that $v < \max(C)$ that can extend C. This is a contradiction and hence C is an α-maximal clique. □

Lemma 15. *Let C be an α-maximal clique in $\mathcal{G}$. Then C is emitted by Algorithm 23.*

Proof. We first show that a call to method Enum-Uncertain-MC with α-clique C enumerates all α-maximal cliques C' in $\mathcal{G}$, such that for all $c \in \{C' \setminus C\}$, $c > \max(C)$.

Without loss of generality, consider a α-maximal clique C' in $\mathcal{G}$ such that $\forall c \in \{C' \setminus C\}$, $c > \max(C)$. Note that C' will be emitted as an α-maximal clique by the method Enum-Uncertain-MC when called with C, if the following holds: (1) A call to method Enum-Uncertain-MC is made with C', (2) When this call is made, $I' = \emptyset$, and $X' = \emptyset$. Since C' is α-maximal clique in $\mathcal{G}$, the second point follows from Lemmas 12 and 13. Thus we need to show that a call to Enum-Uncertain-MC is made with C'.

We prove this by induction. Let $\hat{C} = \{C' \setminus C\}$. Let c_i represent the ith element in $\hat{C}$ in lexicographical order. Also let $C_i = C \cup \{c_1, c_2, \ldots, c_i\}$. For the base case, we show that if a call to Enum-Uncertain-MC is made with C, a call will be made with $C_1 = C \cup \{c_1\}$. This is because, line 4 of the method loops over every vertex $u \in I$ thus implying $u > \max(C)$ and $clq(C \cup \{u\}, \mathcal{G}) \geq \alpha$. Since C' is an α-maximal clique, c_1 will satisfy both these conditions and hence a call to Enum-Uncertain-MC is made with $C \cup \{c_1\}$. Now for the inductive step we show that if a call is made with clique C_i, then this call will in turn call the method with clique C_{i+1}. Again, c_{i+1} is greater than $\max(C_i)$ and $clq(C_i \cup \{c_{i+1}\}, \mathcal{G}) \geq \alpha$. Thus $c_{i+1} \in I$ when the call is made to Enum-Uncertain-MC with C_i. Hence using the previous argument, in line 4, c_{i+1} will be used as a vertex in the loop which would in turn make a call to Enum-Uncertain-MC with C_{i+1}.

Now without any loss of generality, consider an α-maximal clique in $\mathcal{G}$. We know that $C \supset \emptyset$. Thus the proof follows. □

5.2.2 Runtime Complexity

Theorem 3. *The runtime of MULE (Algorithm 22) on an input graph of n vertices is $O(n \cdot 2^n)$.*

Proof. MULE initializes variables and calls to Algorithm 23, hence we analyze the runtime of Algorithm 23. An execution of the recursive Algorithm 23 can be viewed as a search tree as follows. Each call to Enum-Uncertain-MC is a node of this search tree. The first call to the method is the root node. A node in this search tree is either an internal node that makes one or more recursive calls, or a leaf node that does not make further recursive calls. To analyze the runtime of Algorithm 23, we consider the time spent at internal nodes as well as leaf nodes.

The runtime at each leaf node is $O(1)$. For a leaf node, the parameter $I = \emptyset$, and there are no further recursive calls. This implies that either C is α-maximal ($X = \emptyset$) and is emitted in line 2 or it is non-maximal ($X \neq \emptyset$) but cannot be extended by the loop in line 4 as $I = \emptyset$. Checking the sizes of I and X takes constant time.

We next consider the time taken at each internal node. Instead of adding up the times at different internal nodes, we equivalently add up the cost of the different edges in the search tree. At each internal node, the cost of making a recursive call can be analyzed as follows. Line 5 takes $O(n)$ time as we add all vertices in C to C' and also u. Line 6 takes constant time. Lines 7 and 8 take $O(n)$ time (Lemmas 16 and 17 respectively). Note that lines 5 to 8 can get executed only once in between the two calls. Thus total runtime for each edge of the search tree is $O(n)$.

Note that the total number of calls made to the method method Enum-Uncertain-MC is no more than the possible number of unique subsets of V, which is $O(2^n)$. We see that for internal nodes, time complexity is $O(n)$ and for leaf nodes it is $O(1)$. Hence the total time complexity is $O(n \cdot 2^n)$. □

Thus now we need to prove that lines 7 and 8 take $O(n)$ time. This implies that time complexity of Algorithms 24 and 25 is $O(n)$. We prove the same in Lemmas 16 and 17 respectively.

Lemma 16. *The runtime of Algorithm 24 is $O(n)$.*

Proof. First note that lines 1-6 takes $O(n)$ time. This is because $|I| = O(n)$, and hence the loop at line 4 of Algorithm 24 can take $O(n)$ time. Further the set intersection at line 6 also takes $O(n)$ time. We need to show that the for loop in line 7 is $O(n)$, that is each iteration of the loop takes $O(1)$ time. Assume that it takes constant time to find out the probability of an edge. This is a valid assumption, as the edge probabilities can be stored as a HashMap and hence for an edge e, in constant time we can find out $p(e)$. With this assumption, it is easy to show that lines 8-13 takes constant time. This is because, they are either constant number of multiplications, or adding one element to a set. Thus total time complexity is $O(n)$. □

Lemma 17. *The runtime of Algorithm 25 is $O(n)$.*

We omit the proof of the above lemma since it is similar to the proof of Lemma 16.

Observation 6. *The worst-case runtime of any algorithm that can output all maximal cliques of an uncertain graph on n vertices is $\Omega(\sqrt{n} \cdot 2^n)$.*

Proof. From Theorem 1, we know that the number of maximal uncertain cliques can be as much as $\binom{n}{\lfloor n/2 \rfloor} = \Theta\left(\frac{2^n}{\sqrt{n}}\right)$. Since the size of each uncertain clique can be $\Theta(n)$, the total output size can be $\Omega(\sqrt{n} \cdot 2^n)$, which is a lower bound on the runtime of any algorithm. □

Lemma 18. *The worst-case runtime of MULE on an n vertex graph is within a $O(\sqrt{n})$ factor of the runtime of an optimal algorithm for Maximal Clique Enumeration on an uncertain graph.*

Proof. The proof follows from Theorem 3 and Observation 6. □

5.2.3 Enumerating Only Large Maximal Cliques

For a typical input graph, many maximal cliques are small, and may not be interesting to the user. Hence it is helpful to have an algorithm that can enumerate only large maximal cliques efficiently, rather than enumerate all maximal cliques. We now describe an algorithm that enumerates every α-maximal clique with more than t vertices, where t is an user provided parameter.

As a first step, we prune the input uncertain graph $\mathcal{G} = (V, E, p)$ by employing techniques described by Modani and Dey (2008). We apply the "Shared Neighborhood Filtering" where edges are recursively checked and removed as follows. First drop all edges $\{u, v\} \in E$, such that $|\Gamma(u) \cap \Gamma(v)| < (t-2)$. Next drop every vertex $v \in V$, that doesn't satisfy the following condition. For vertex $v \in V$, there must exist at least $(t-1)$ vertices in $\Gamma(v)$, such that for $u \in \Gamma(v)$, $|\Gamma(u) \cap \Gamma(v)| < (t-2)$. Let $\mathcal{G}'$ denote the graph resulting from $\mathcal{G}$ after the pruning step.

Algorithm 26 runs on the pruned uncertain graph $\mathcal{G}'$ to enumerate only large maximal cliques. The recursive method in Algorithm 27 differs from Algorithm 23 as follows. Before each recursive call to method Enum-Uncertain-MC-Large (Algorithm 27), the algorithm checks if the sum of the sizes of the current working clique C' and the candidate vertex set I' are greater than the size threshold t. If not, the recursive method is not called. This optimization leads to a substantial pruning of the search space and hence a reduction in runtime.

Algorithm 26: LARGE–MULE($\mathcal{G}, \alpha$,t)

```
Input: $\mathcal{G}'$ is the input uncertain graph post pruning
       $\alpha, 0 < \alpha < 1$ is the user provided probability threshold
       $t, t \geq 2$ is the user provided size threshold
1 $\hat{I} \leftarrow \emptyset$ ;
2 forall the $u \in V$ do
3     $\hat{I} \leftarrow \hat{I} \cup \{(u, 1)\}$
4 Enum-Uncertain-MC-Large($\emptyset$, 1 ,$\hat{I}$, $\emptyset$,$t$) ;
```

Algorithm 27: Enum-Uncertain-MC-Large(C, q, I, X,t)

```
Input: $C$ is the current Uncertain Clique being processed
       $q$ is pre-computed $clq(C, \mathcal{G})$
       $I$ is a set of tuples $(u, r)$, such that $\forall (u, r) \in I$, $u > \max(C)$, and
       $clq(C \cup \{u\}, \mathcal{G}) = q \cdot r \geq \alpha$, i.e. $C \cup \{u\}$ is an $\alpha$-clique in $\mathcal{G}$
       $X$ is a set of tuples $(v, s)$, such that $\forall (v, s) \in X$, $v \notin C$, $v < \max(C)$, and
       $clq(C \cup \{v\}, \mathcal{G}) = q \cdot s \geq \alpha$ , i.e. $C \cup \{v\}$ is an $\alpha$-clique in $\mathcal{G}$
       $t$ is the user provided size threshold
1  if $I = \emptyset$ and $X = \emptyset$ then
2      Output $C$ as $\alpha$-maximal clique ;
3      return
4  forall the $u, r \in I$ taken in lexicographical ordering of $u$ do
5      $C' \leftarrow C \cup \{u\}$ // Lemma 12:  $m = max(C') = u$
6      $q' \leftarrow q \cdot r$ //  $clq(C \cup \{v\}, \mathcal{G})$
7      $I' \leftarrow GenerateI(C', q', I)$ ;
8      if $|C'| + |I'| < t$ then
9          continue ;
10     $X' \leftarrow GenerateX(C', q', X)$ ;
11     Enum-Uncertain-MC-Large($C', q', I', X', t$) ;
12     $X \leftarrow X \cup \{(u, r)\}$
```

Lemma 19. *Given an input graph $\mathcal{G}$, LARGE–MULE (Algorithm 26) enumerates every α-maximal clique with more than t vertices.*

Proof. First we prove that no maximal clique of size less than t is enumerated by Algorithm 27. Consider an α-maximal clique C_1 in $\mathcal{G}$ with less than t vertices. Also let $m_1 = \max(C_1)$ and $C_1' = C_1 \setminus \{m_1\}$. Note that if C_1 is emitted by Algorithm 27, then a call must be made to Enum-Uncertain-MC-Large with C_1. Since the Algorithm adds vertices in lexicographical ordering, this implies that a call must be made to Enum-Uncertain-MC-Large with C_1' before the call is made with C_1. In the worst case, let us consider that the search tree reaches the execution point where Enum-Uncertain-MC-Large is called with C_1'. Consider the execution of the algorithm where m_1 is added to $C = C_1'$ to form $C' = C_1$. Since C_1 is an α-maximal clique, I' will become NULL which implies $|I'| = 0$. We know that $|C_1| < t$. Thus $|C_1 + I'|$ will also be less than t and the If condition (line 8) will succeed. This will result in the execution of the continue statement. Thus Enum-Uncertain-MC-Large will not be called with C_1 implying that C_1 is not enumerated.

Next we show that any maximal clique of size at least t is enumerated by Algorithm 27. Consider an α-maximal clique C_2 in $\mathcal{G}$ of size at least t. We note that the “If” condition in line 8 is never satisfied in the search path ending with C_2 and hence a call is made to the method with Enum-Uncertain-MC-Large with C_2. This is easy to see as whenever a call is made to Enum-Uncertain-MC-Large with any $C \subseteq C_2$, since C_2 is large, we always have $|C| + |I| \geq t$. $\square$

5.3 Experimental Results

We report the results of an experimental evaluation of our algorithm. We implemented the algorithm using Java. We ran all experiments on a system with a 3.19 GHz Intel(R) Core(TM) i5 processor and 4 GB of RAM, with heap space configured at 1.5GB.

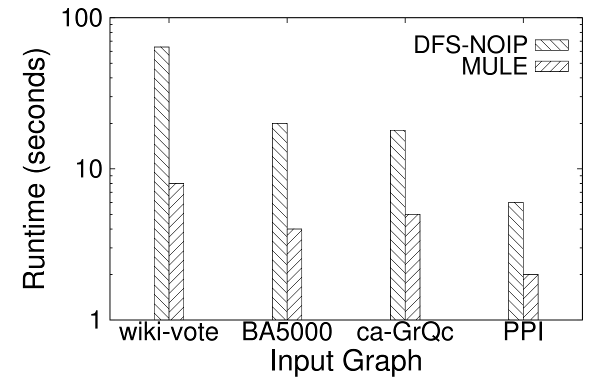
Table 5.1: Input Graphs

Input Graph	Category	# Vertices	# Edges
Fruit-Fly	Protein Protein Interaction network	3751	3692
DBLP10	Social network	2284991	
p2p-Gnutella08	Internet peer-to-peer networks	6301	20777
p2p-Gnutella04	Internet peer-to-peer networks	10879	39994
p2p-Gnutella09	Internet peer-to-peer networks	8114	26013
ca-GrQc	Collaboration networks	5242	28980
wiki-vote	Social networks	7118	103689
BA5000	Barabási–Albert random graphs	5000	50032
BA6000	Barabási–Albert random graphs	6000	60129
BA7000	Barabási–Albert random graphs	7000	70204
BA8000	Barabási–Albert random graphs	8000	80185
BA9000	Barabási–Albert random graphs	9000	90418
BA10000	Barabási–Albert random graphs	10000	99194

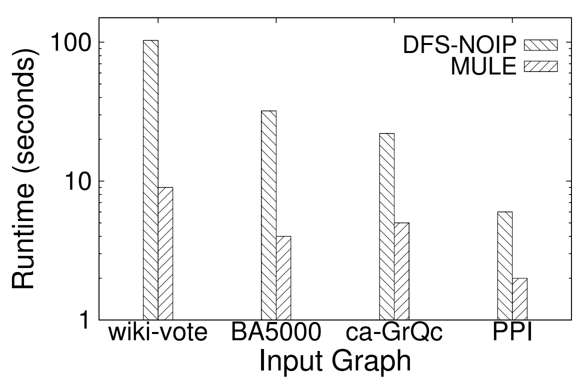
Input Data: Details of the input graphs that we used are shown in Table 5.1.

The first set of graphs consists of real world uncertain graphs shared by authors of Zou et al. (2010c) and Khan et al. (2014). These include a protein-protein interaction (PPI) network of a Fruit Fly obtained by integrating data from the BioGRID [1] database with that form the STRING [2] database, and the DBLP [3] dataset from authors of Khan et al. (2014), which is an uncertain network predicting

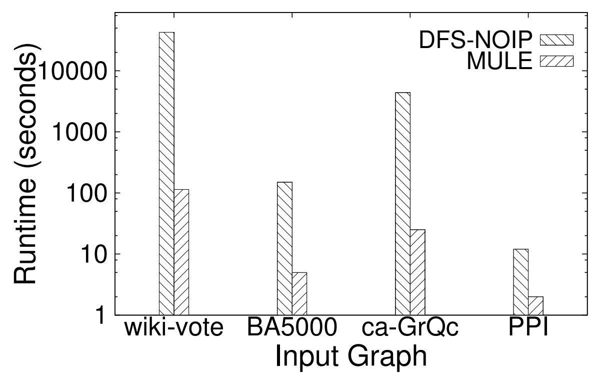
[1] http://thebiogrid.org/
[2] http://string-db.org/
[3] http://dblp.uni-trier.de/



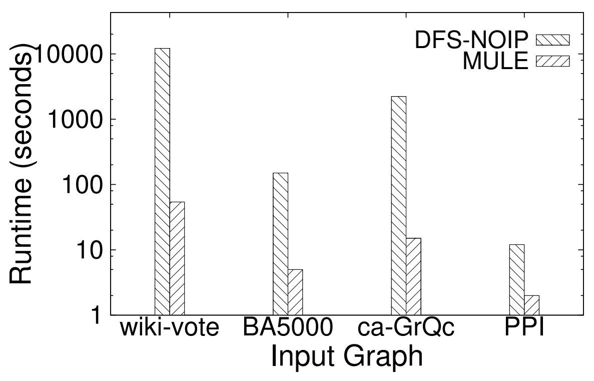
(a) $\alpha = 0.9$



(b) $\alpha = 0.8$



(c) $\alpha = 0.0001$

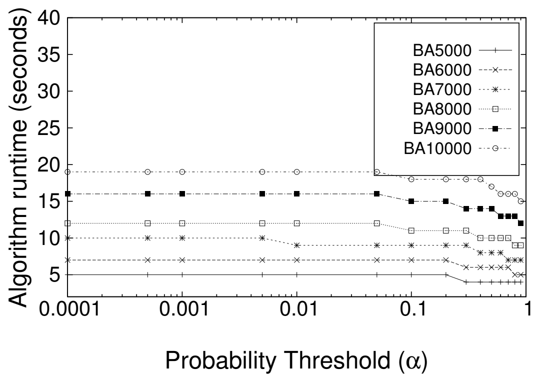


(d) $\alpha = 0.0005$

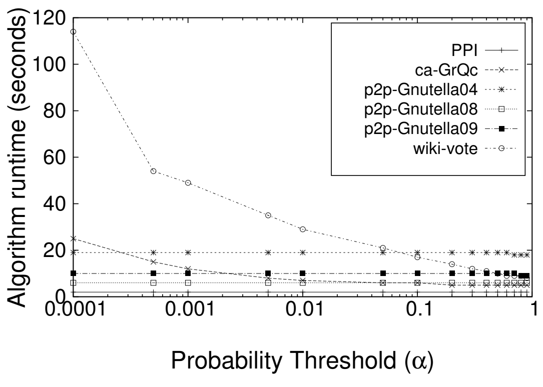
Figure 5.1: Comparison of Simple and Optimized Depth First Search approaches. The y–axis is in log–scale.

future co-authorship. The PPI network is an uncertain graph where each vertex represents a protein and two vertices are connected by an edge with a probability representing the likelihood of interaction between the the two proteins. The DBLP network represents co-authorship in academic articles. Each vertex in this network represents an author. Two vertices are connected by an edge with a probability that depends on the "strength" of their co-authorship, which is computed as $1 - e^{-c/10}$, where c is the number of papers co–authored.

The second set of graphs was obtained from the Stanford Large Network Collection Leskovec (), and includes graphs representing Internet p2p networks, collaboration networks, and an online social



(a) Random Graphs



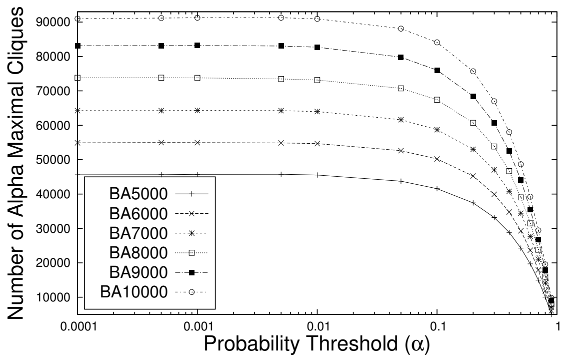
(b) Semi–synthetic and Real Graphs

Figure 5.2: Runtime vs Probability threshold (α). The x–axis is in log–scale

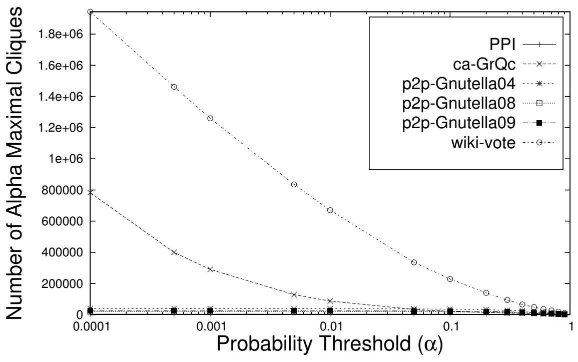
network. The p2p-Gnutella graphs represent peer to peer file sharing networks, where each vertex in the graph represents a computer and the edges represent the communication among them. The p2p-Gnutella04, p2p-Gnutella08 and p2p-Gnutella09 graphs represent communications occurring on 4th, 8th and 9th of August, 2002 respectively. The ca-GrQc graph represents the collaboration network among scientist working on General Relativity and Quantum Cosmology. Each vertex in the graph is a scientist and two vertices are connected by an edge if the corresponding scientists have co-authored a paper. Finally the wiki-vote graph represents the voting that occurs while selecting a new wikipedia administrator. Each vertex is either a wikipedia admin or wikipedia user and the edges represent the votes that each admin / user casts in favor of a candidate. The candidate is also a wikipedia user and hence is represented by a vertex in the graph. For all these graphs, the uncertain graphs were created from these deterministic graphs by assigning edge probabilities uniformly at random. Hence these can be considered as semi–synthetic uncertain graphs.

The third set of input graphs was synthetically generated using the Barabási–Albert model for random graphs Albert and Barabási (2002). Then the edges were assigned probabilities uniformly at random from $[0, 1]$.

Comparison with other approaches. We compare our algorithm with another algorithm based on depth-first-search, which we call DFS-NOIP (DFS with NO Incremental Probability Computation),



(a) Random Graphs



(b) Semi–synthetic and Real Graphs

Figure 5.3: No of α-maximal cliques vs Probability threshold (α). The x–axis is in log–scale

described in Algorithm 28. This algorithm also performs a depth first search to enumerate all α–maximal cliques but does not compute the probabilities incrementally like MULE does. Figure 5.1 compares the performance of MULE with DFS–NOIP. The results show that MULE performs much better than DFS–NOIP. For instance, for the graph wiki–vote with $\alpha = 0.9$ DFS–NOIP took 64 seconds while MULE took only 8 secs. The relative performance results hold true over a wide range of input graphs and values of α, including synthetic and real-world graphs, and small and large values of α. For $\alpha = 0.0001$, MULE took only 25 secs to enumerate all maximal cliques in ca-GrQc, while DFS–NOIP took over 4400 secs. On the wiki–vote input graph with probability threshold 0.9, MULE took 8 seconds while DFS–NOIP took 64 seconds. For the same graph, with probability threshold 0.0001, MULE took 114 secs, while DFS–NOIP took more than 11 hours.

Dependence on α. We measured the runtime of enumeration as well as the output size, (the number of α-maximal cliques that were output) for different values of α and for the various input graphs described above. The dependence of the runtime on α is shown in Figure 5.2, and the number of cliques as a function of α is shown in Figure 5.3. We note that as α increases, the number of maximal cliques, and the time of enumeration both drop sharply. The decrease in runtime is because with a larger value of α, the algorithm is able to prune search paths aggressively early in the enumeration. We note that the number of α-maximal cliques does not have to always decrease as α increases. Sometimes it is possible

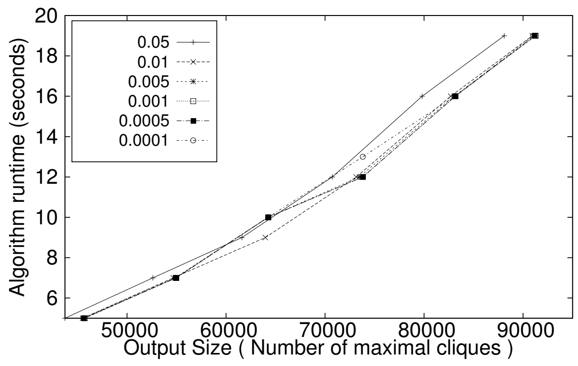
Algorithm 28: DFS with NO Incremental Probability Computation: DFS–NOIP(C,I)

```
1  $I_{copy} \leftarrow I$ ;
2  forall the $u \in I_{copy}$ do
3      if $u \leq max(C)$ OR $clq(C \cup \{u\}) < \alpha$ then
4          $I \leftarrow I \setminus \{u\}$
5  if $I = \emptyset$ then
6      if $C$ is an $\alpha$-maximal clique then
7          Output $C$ as $\alpha$-maximal clique ;
8          return;
9  forall the $v \in I$ do
10     $C' \leftarrow C \cup \{v\}$ ;
11     if $C'$ is an $\alpha$-maximal clique then
12         Output $C'$ as $\alpha$-maximal clique ;
13     else
14         $I' \leftarrow I \cap \Gamma(v)$ ;
15         DFS–NOIP($C'$,$I'$) ;
```

that the number of α-maximal cliques increases with α. This is because as α increases, a large maximal clique may split into many smaller maximal cliques. However, these differences are negligible, and are not visible in the plots.

Dependence on Size of Output. Figure 5.4 shows the change in runtime with respect to the number of α-maximal cliques enumerated, for the randomly generated graphs. It can be seen that the runtime of the algorithm is almost proportional to the number of maximal cliques in the output. This shows that the algorithm runtime scales well with the number of α-maximal cliques in output. This comparison was not done for real world or semi–synthetic graphs as these graphs have different structural properties, hence different sizes of maximal cliques and thus there is no meaningful way to interpret the results.

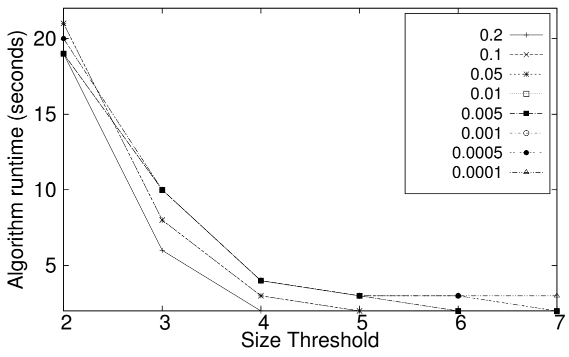
Enumerating Large Maximal Cliques. Figures 5.5 and 5.6 show the runtime of LARGE–MULE (Algorithm 26) and the output size respectively as a function of t, the minimum size of an α-maximal clique that is output. As t increases, both runtime and output size decrease substantially. For instance, MULE takes 76797 seconds to enumerate all uncertain maximal cliques from the DBLP dataset (for probability threshold 0.9). However, LARGE–MULE takes only 32 seconds when $t = 3$. Similarly, for input graph ca-GrQc and $\alpha = 0.0001$, MULE takes 125 seconds, while LARGE–MULE takes 10 seconds when $t = 6$ and 6 seconds when $t = 7$.



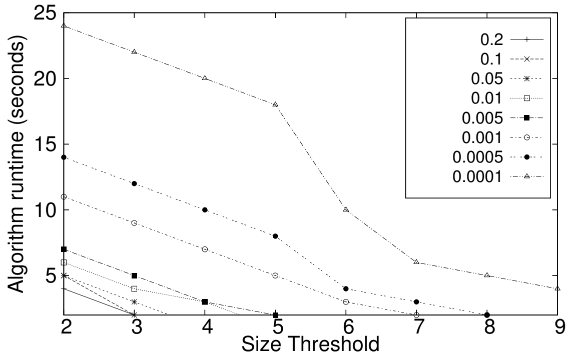
(a) Random Graphs

Figure 5.4: Runtime vs Output size

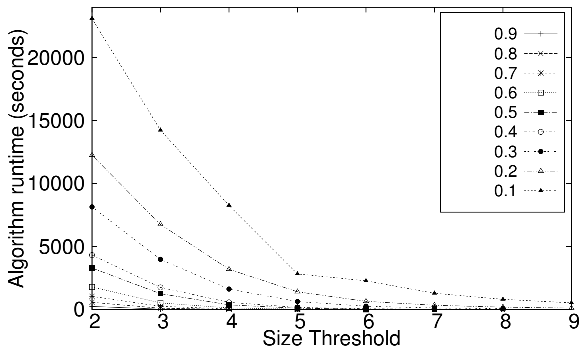
Runtime with change in Graph Uncertainty. Figure 5.7 shows the change in runtime with respect to the change in graph uncertainty. We use the ca–GrQc and DBLP10 networks for this experiment. We consider the same underlying network with difference in the number of uncertain edges. We use the following three cases: 1) when all edges are uncertain, 2) when two–third of all the edges are uncertain, and 3) when one–third of all the edges are uncertain. We can see that when we increase the number of uncertain edges in the graph, the runtime decreases. Considering that there can be many more uncertain maximal cliques than just maximal cliques, Figure 5.7 shows that our algorithm employs effective pruning techniques that help us to quickly identify all maximal uncertain cliques. For example, for the graph ca-GrQc and $\alpha = 0.5$, MULE took 6 and 7 seconds for all uncertain and two–third uncertain edges respectively. However, for one–third uncertain edges, it took 124 seconds. Again for the graph DBLP10 and $\alpha = 0.9$, MULE (with size threshold 4), could find all maximal cliques in 6 seconds with all uncertain edges. But with only two–third uncertain edges, the same graph took over 42 minutes with one–third uncertain edges, we could not process the graph within 4 hours. Thus, the graph processing time required by our method reduces as the uncertainty of the graph increases. Note that there are two ways in which uncertain cliques can be handled – first by using deterministic MCE and then finding embedded uncertain cliques, and second, by directly incorporating uncertainty in pruning, as we do. From Figure 5.7 we can see that for multiple values of α, as the number of uncertain edges increase, runtime of MULE decreases. This implies that when we directly model uncertainty and use



(a) BA10000



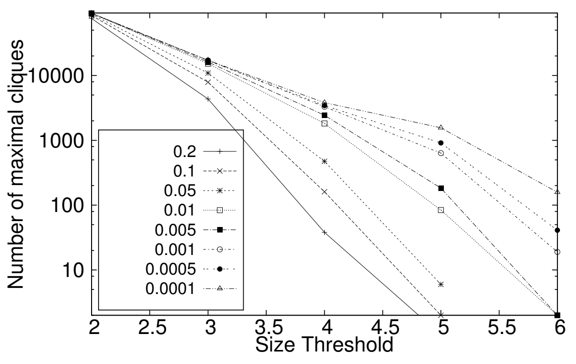
(b) ca-GrQc



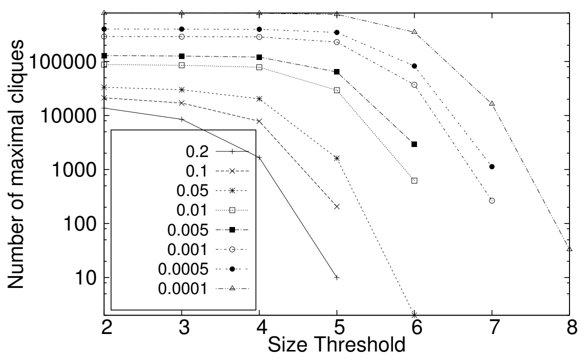
(c) DBLP

Figure 5.5: Runtime vs Size threshold of enumerated uncertain maximal cliques

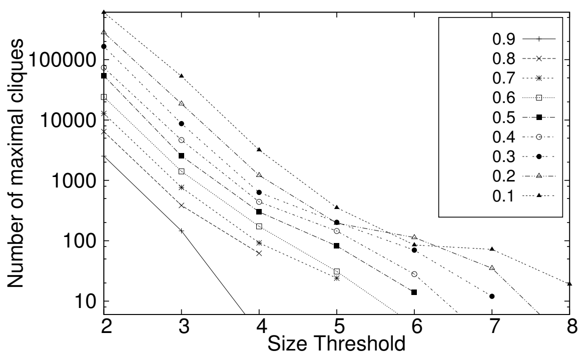
our Algorithm, we can find structures much faster than finding all maximal cliques followed by pruning on basis of probability to find maximal uncertain cliques.



(a) BA10000

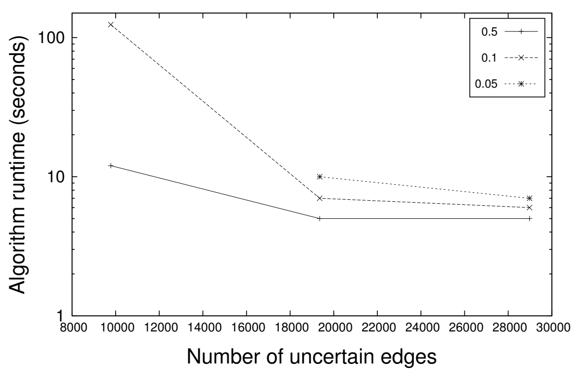


(b) ca-GrQc

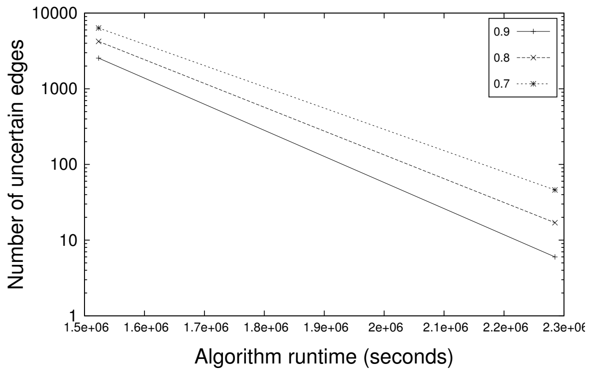


(c) DBLP

Figure 5.6: Number of α-maximal cliques vs Size threshold of enumerated uncertain maximal cliques



(a) ca–GrQc



(b) DBLP10 (s=4)

Figure 5.7: Runtime vs the number of uncertain edges in the graph. The y–axis is in log scale. Observe that the runtime decreases when the number of uncertain edges in the graph increases.

CHAPTER 6. SUMMARY AND DISCUSSION

As stated before, mining dense substructures in large graphs is an open field with many unanswered questions. In this thesis we try to answer some of the most fundamental and basic problems in this area. Both Maximal clique and Maximal biclique enumeration are fundamental tools in uncovering dense relationships within graphical data. We presented a scalable parallel method for mining maximal bicliques from a large graph. Our method uses a basic clustering framework for parallelizing the enumeration, followed by two optimizations, one for reducing redundant work, and another for improving load balance. Experimental results using MapReduce show that the algorithms are effective in handling large graphs, and scale with increasing number of reducers. To our knowledge, this is the first work to successfully enumerate bicliques from graphs of this size; previous reported results were mostly sequential methods that worked on much smaller graphs. We also performed experimental study among parallel algorithms for Maximal clique enumeration.

We also present a systematic study of the enumeration of maximal cliques from an uncertain graph, starting from a precise definition of the notion of an α-maximal clique, followed by a proof showing that the maximum number of α-maximal cliques in a graph on n vertices is exactly $\binom{n}{\lfloor n/2 \rfloor}$, for $0 < \alpha < 1$. We present a novel algorithm, MULE, for enumerating the set of all α-maximal cliques from a graph, and an analysis showing that the worst-case runtime of this algorithm is $O\left(n \cdot 2^n\right)$. We present an experimental evaluation of MULE showing its performance, and an extension for faster enumeration of large maximal cliques. An interesting open problem is to design an algorithm for enumerating maximal cliques from an uncertain graph whose time complexity is worst-case optimal, $O\left(\sqrt{n} \cdot 2^n\right)$.

Some future directions of this work can be enumerating uncertain maximal bicliques and finding quasi–dense substructures from a large graph..

Bibliography

Abello, J., Resende, M. G. C., and Sudarsky, S. (2002). Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics*, volume 2286 of *Lecture Notes in Computer Science*, pages 598–612. Springer Berlin Heidelberg.

Adar, E. and Re, C. (2007). Managing uncertainty in social networks. *IEEE Data Engineering Bulletin*, 30(2):15–22.

Agarwal, P. K., Alon, N., Aronov, B., and Suri, S. (1994). Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12(1):347–365.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97.

Alexe, G., Alexe, S., Crama, Y., Foldes, S., Hammer, P. L., and Simeone, B. (2004). Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11–21.

An, Y., Janssen, J., and Milios, E. E. (2004). Characterizing and mining the citation graph of the computer science literature. *Knowledge and Information Systems*, 6:664–678. 10.1007/s10115-003-0128-3.

Asthana, S., King, O. D., Gibbons, F. D., and Roth, F. P. (2004). Predicting protein complex membership using probabilistic network reliability. *Genome Research*, 14:1170–1175.

Bader, J., Chaudhuri, A., Rothberg, J., and Chant, J. (2004). Gaining confidence in high-throughput protein interaction networks. *Nature Biotechnology*, 22(1):78–85.

Biswas, S. and Morris, R. (2005). Exor: opportunistic multi-hop routing for wireless networks. *ACM SIGCOMM Computer Communication Review*, 35(4):133–144.

Boldi, P., Bonchi, F., Gionis, A., and Tassa, T. (2012). Injecting uncertainty in graphs for identity obfuscation. *Proceedings of the VLDB Endowment*, 5(11):1376–1387.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. (2000). Graph structure in the web. *Computer Networks*, 33(1):309 – 320.

Bron, C. and Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. *Communications of ACM*, 16(9):575–577.

Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G., and Chen, R. (2003). Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9):2443–2450.

Chen, Y. and Crippen, G. M. (2005). A novel approach to structural alignment using realistic structural and environmental information. *Protein Science*, 14(12):2935–2946.

Cho, E., Myers, S. A., and Leskovec, J. (2011). Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 1082–1090. ACM.

Colantonio, A., Pietro, R. D., Ocello, A., and Verde, N. V. (2010). Taming role mining complexity in rbac. *Computers & Security*, 29(5):548 – 564.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Opearting Systems Design & Implementation*, OSDI'04, pages 137–150, Berkeley, CA, USA. USENIX Association.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51:107–113.

Dias, V. M. F., Celina, M. M. H. d. F., and Szwarcfiter, J. L. (2005). Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337:240–248.

Driskell, A. C., Ané, C., Burleigh, J. G., McMahon, M. M., O'Meara, B. C., and Sanderson, M. J. (2004). Prospects for building the tree of life from large sequence databases. *Science*, 306(5699):1172–1174.

Du, N., Wu, B., Xu, L., Wang, B., and Pei, X. (2006). A parallel algorithm for enumerating all maximal cliques in complex network. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 320–324.

Eppstein, D. (1994). Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51:207–211.

Eppstein, D., Loffler, M., and Strash, D. (2010). Listing all maximal cliques in sparse graphs in near-optimal time. In *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer Berlin / Heidelberg.

Erdős, P. and Rényi, A. (1959). On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297.

Gaspers, S., Kratsch, D., and Liedloff, M. (2008). Graph-theoretic concepts in computer science. chapter On Independent Sets and Bicliques in Graphs, pages 171–182. Springer.

Gély, A., Nourine, L., and Sadi, B. (2009). Enumeration aspects of maximal cliques and bicliques. *Discrete Applied Mathematics*, 157(7):1447 – 1459.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA. ACM.

Ghosh, J., Ngo, H., Yoon, S., and Qiao, C. (2007). On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1721–1729.

Gibson, D., Kumar, R., and Tomkins, A. (2005). Discovering large dense subgraphs in massive graphs. PVLDB, pages 721–732. VLDB Endowment.

Grindley, H. M., Artymiuk, P. J., Rice, D. W., and Willett, P. (1993). Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm. *Journal of Molecular Biology*, 229(3):707–721.

Gu, Y. and Grossman, R. L. (2009). Sector and sphere: the design and implementation of a high-performance data cloud. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1897):2429–2445.

Guha, R., Kumar, R., Raghavan, P., and Tomkins, A. (2004). Propagation of trust and distrust. In *Proceedings of the 13th International conference on World Wide Web*, WWW '04, pages 403–412, New York, NY, USA. ACM.

Hall, B. H., Jaffe, A. B., and Trajtenberg, M. (2001). The nber patent citation data file: Lessons, insights and methodological tools. Nber working papers, National Bureau of Economic Research, Inc.

Harley, E. and Bonner, A. (2001). Uniform integration of genome mapping data using intersection graphs. *Bioinformatics*, 17(6):487–494.

Hattori, M., Okuno, Y., Goto, S., and Kanehisa, M. (2003). Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways. *Journal of the American Chemical Society*, 125(39):11853–11865.

Hintsanen, P. and Toivonen, H. (2008). Finding reliable subgraphs from large probabilistic graphs. *Data Mining and Knowledge Discovery*, 17(1):3–23.

Jermaine, C. (2005). Finding the most interesting correlations in a database: how hard can it be? *Information Systems*, 30(1):21 – 46.

Jiang, R., Tu, Z., Chen, T., and Sun, F. (2006). Network motif identification in stochastic networks. *Proceedings of the National Academy of Sciences*, 103(25):9404–9409.

Jin, R., Liu, L., and Aggarwal, C. C. (2011a). Discovering highly reliable subgraphs in uncertain graphs. In *Proceedings of the 17th ACM SIGKDD International conference on Knowledge discovery and data mining*, KDD '11, pages 992–1000, New York, NY, USA. ACM.

Jin, R., Liu, L., Ding, B., and Wang, H. (2011b). Distance-constraint reachability computation in uncertain graphs. *Proceedings of the VLDB Endowment*, 4(9):551–562.

Jonsson, P. F. and Bates, P. A. (2006). Global topological features of cancer proteins in the human interactome. *Bioinformatics*, 22(18):2291–2297.

Karloff, H., Suri, S., and Vassilvitskii, S. (2010). A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Kawahigashi, H., Terashima, Y., Miyauchi, N., and Nakakawaji, T. (2005). Modeling ad hoc sensor networks using random graph theory. In *Second IEEE Consumer Communications and Networking Conference*, pages 104–109.

Kempe, D., Kleinberg, J., and Tardos, E. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International conference on Knowledge discovery and data mining*, KDD '03, pages 137–146, New York, NY, USA. ACM.

Khan, A., Bonchi, F., Gionis, A., and Gullo, F. (2014). Fast reliability search in uncertain graphs. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '14, pages 535–546, New York, NY, USA. ACM.

Kollios, G., Potamias, M., and Terzi, E. (2013). Clustering large probabilistic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):325–336.

Kose, F., Weckwerth, W., Linke, T., and Fiehn, O. (2001). Visualizing plant metabolomic correlation networks using clique-metabolite matrices. *Bioinformatics*, 17(12):1198–1208.

Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999). Trawling the web for emerging cyber-communities. *Computer networks*, 31(11):1481–1493.

Kuter, U. and Golbeck, J. (2010). Using probabilistic confidence models for trust inference in web-based social networks. *ACM Transactions on Internet Technology*, 10(2):8:1–8:23.

Lee, V., Ruan, N., Jin, R., and Aggarwal, C. (2010). A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 303–336. Springer US.

Lehmann, S., Schwartz, M., and Hansen, L. K. (2008). Biclique communities. *Physical Review E*, 78:016108.

Leskovec, J. Stanford large network dataset collection. `http://snap.stanford.edu/data/index.html`.

Leskovec, J., Huttenlocher, D., and Kleinberg, J. (2010). Signed networks in social media. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1361–1370. ACM.

Leskovec, J., Kleinberg, J., and Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187. ACM.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. (2009). Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123.

Li, J., Liu, G., Li, H., and Wong, L. (2007). Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1625–1637.

Li, J. and Liu, Q. (2009). 'double water exclusion' : a hypothesis refining the o–ring theory for the hot spots at protein interfaces. *Bioinformatics*, 25(6):743–750.

Liben-Nowell, D. and Kleinberg, J. (2003). The link prediction problem for social networks. In *Proceedings of the 12th International conference on Information and knowledge management*, CIKM '03, pages 556–559, New York, NY, USA. ACM.

Liu, G., Sim, K., and Li, J. (2006). Efficient mining of large maximal bicliques. In *Data Warehousing and Knowledge Discovery*, volume 4081 of *Lecture Notes in Computer Science*, pages 437–448. Springer.

Liu, L., Jin, R., Aggarwal, C., and Shen, Y. (2012). Reliable clustering on uncertain graphs. In *IEEE 12th International Conference on Data Mining (ICDM)*, pages 459–468.

Lo, D., Surian, D., Zhang, K., and Lim, E.-P. (2011). Mining direct antagonistic communities in explicit trust networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1013–1018. ACM.

Lu, L., Gu, Y., and Grossman, R. (2010). dmaximalcliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1320–1327.

Makino, K. and Uno, T. (2004). New algorithms for enumerating all maximal cliques. In *Algorithm Theory-SWAT 2004*, pages 260–272. Springer.

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 135–146. ACM.

Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P., and Bhattacharjee, B. (2007). Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42. ACM.

Modani, N. and Dey, K. (2008). Large maximal cliques enumeration in sparse graphs. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 1377–1378.

Mohseni-Zadeh, S., Brzellec, P., and Risler, J.-L. (2004). Cluster-c, an algorithm for the large-scale clustering of protein sequences based on the extraction of maximal cliques. *Computational Biology and Chemistry*, 28(3):211–218.

Moon, J. and Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28.

Mouret, S., Grossmann, I. E., and Pestiaux, P. (2011). Time representations and mathematical models for process scheduling problems. *Computers & Chemical Engineering*, 35(6):1038 – 1063.

Mukherjee, A. P. and Tirthapura, S. (2014). Enumerating maximal bicliques from a large graph using mapreduce. In *2014 IEEE 3rd International Congress on Big Data*.

Mukherjee, A. P., Xu, P., and Tirthapura, S. (2015). Mining maximal cliques from an uncertain graph. In *2015 IEEE 31st International Conference on Data Engineering*.

Mushlin, R. A., Kershenbaum, A., Gallagher, S. T., and Rebbeck, T. R. (2007). A graph-theoretical approach for pattern discovery in epidemiological research. *IBM Systems Journal*, 46(1):135–149.

Nagarajan, N. and Kingsford, C. (2008). Uncovering genomic reassortments among influenza strains by enumerating maximal bicliques. In *IEEE International Conference on Bioinformatics and Biomedicine, 2008*, pages 223–230.

Nataraj, R. V. and Selvan, S. (2009). Parallel mining of large maximal bicliques using order preserving generators. *International Journal of Computing*, 8(3):105–113.

Newman, M. E. J., Watts, D. J., and Strogatz, S. H. (2002). Random graph models of social networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 1):2566–2572.

Palla, G., Dernyi, I., Farkas, I., and Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818.

Peeters, R. (2003). The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131:651–654.

Potamias, M., Bonchi, F., Gionis, A., and Kollios, G. (2010). k-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, 3(1-2):997–1008.

Rhodes, D. R., Tomlins, S. A., Varambally, S., Mahavisno, V., Barrette, T., Kalyana-Sundaram, S., Ghosh, D., Pandey, A., and Chinnaiyan, A. M. (2005). Probabilistic model of the human protein-protein interaction network. *Nature Biotechnology*, 23(8):951–959.

Richardson, M., Agrawal, R., and Domingos, P. (2003). Trust management for the semantic web. In *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 351–368. Springer Berlin / Heidelberg.

Rokhlenko, O., Wexler, Y., and Yakhini, Z. (2007). Similarities and differences of gene expression in yeast stress conditions. *Bioinformatics*, 23(2):e184–e190.

Rome, J. E. and Haralick, R. M. (2005). Towards a formal concept analysis approach to exploring communities on the world wide web. In *Formal Concept Analysis*, volume 3403 of *Lecture Notes in Computer Science*, pages 33–48. Springer.

Sanderson, M. J., Driskell, A. C., Ree, R. H., Eulenstein, O., and Langley, S. (2003). Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular Biology and Evolution*, 20(7):1036–1042.

Schmidt, M. C., Samatova, N. F., Thomas, K., and Park, B.-H. (2009). A scalable, parallel algorithm for maximal clique enumeration. *J. Parallel Distrib. Comput.*, 69:417–428.

Schweiger, R., Linial, M., and Linial, N. (2011). Generative probabilistic models for protein–protein interaction networks–the biclique perspective. *Bioinformatics*, 27(13):i142–i148.

Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., and Toivonen, H. (2006). Link discovery in graphs derived from biological databases. In *Proceedings of the 3rd International conference on Data Integration in the Life Sciences*, DILS'06, pages 35–49, Berlin, Heidelberg. Springer-Verlag.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1 –10.

Sim, K., Li, J., Gopalkrishnan, V., and Liu, G. (2006). Mining maximal quasi-bicliques to co-cluster stocks and financial ratios for value investment. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 1059–1063. IEEE.

Svendsen, M. (2012). Mining maximal cliques from a large graph using mapreduce. Master's thesis, Iowa State University.

Svendsen, M., Mukherjee, A. P., and Tirthapura, S. (2014). Mining maximal cliques from a large graph using mapreduce: Tackling highly uneven subproblem sizes. *Journal of Parallel and Distributed Computing, Special Issue: Scalable Systems for Big Data Management and Analytics*.

Tomita, E., Tanaka, A., and Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363:28–42.

Tsukiyama, S., Ide, M., Ariyoshi, H., and Shirakawa, I. (1977). A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517.

Uno, T., Kiyomi, M., and Arimura, H. (2004). Lcm ver.2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE International Conference Data Mining Workshop Frequent Itenset Miing Implementations (FIMI)*. IEEE.

Valiant, L. G. (1979). The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421.

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition.

Wodo, O., Tirthapura, S., Chaudhary, S., and Ganapathysubramanian, B. (2012). A graph-based formulation for computational characterization of bulk heterojunction morphology. *Organic Electronics*, 13(6):1105 – 1113.

Wu, B., Yang, S., Zhao, H., and Wang, B. (2009). A distributed algorithm to enumerate all maximal cliques in mapreduce. In *Frontier of Computer Science and Technology, 2009. FCST '09. Fourth International Conference on*, pages 45–51.

Xiang, Y., Payne, P. R. O., and Huang, K. (2012). Transactional database transformation and its application in prioritizing human disease genes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(1):294–304.

Yan, C., Burleigh, J. G., and Eulenstein, O. (2005). Identifying optimal incomplete phylogenetic data sets from sequence databases. *Molecular Phylogenetics and Evolution*, 35(3):528–535.

Yi, J. and Maghoul, F. (2009). Query clustering using click-through graph. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 1055–1056, New York, NY, USA. ACM.

Yoon, S. and Micheli, G. D. (2005). Prediction of regulatory modules comprising micrornas and target genes. *Bioinformatics*, 21(2):ii93–ii100.

Yoshinaka, R. (2011). Towards dual approaches for learning context-free grammars based on syntactic concept lattices. In *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 429–440. Springer Berlin Heidelberg.

Yuan, Y., Chen, L., and Wang, G. (2010). Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In *Database Systems for Advanced Applications*, volume 5981 of *Lecture Notes in Computer Science*, pages 155–170. Springer Berlin Heidelberg.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA. USENIX Association.

Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. (1997). New algorithms for fast discovery of association rules. In *In 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 283–286. AAAI Press.

Zhang, B., Park, B.-H., Karpinets, T., and Samatova, N. F. (2008a). From pull-down data to protein interaction networks and complexes with biological relevance. *Bioinformatics*, 24(7):979–986.

Zhang, Y., Abu-Khzam, F., Baldwin, N., Chesler, E., Langston, M., and Samatova, N. (2005). Genome-scale computational approaches to memory-intensive applications in systems biology. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 12–12.

Zhang, Y., Chesler, E. J., and Langston, M. A. (2008b). On finding bicliques in bipartite graphs: a novel algorithm with application to the integration of diverse biological data types. *Hawaii International Conference on System Sciences*, page 473.

Zou, Z., Gao, H., and Li, J. (2010a). Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *Proceedings of the 16th ACM SIGKDD International conference on Knowledge discovery and data mining*, KDD '10, pages 633–642, New York, NY, USA. ACM.

Zou, Z., Li, J., Gao, H., and Zhang, S. (2010b). Finding top-k maximal cliques in an uncertain graph. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 649–652.

Zou, Z., Li, J., Gao, H., and Zhang, S. (2010c). Mining frequent subgraph patterns from uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, 22(9):1203–1218.