Dense Subgraph Mining in Probabilistic Graphs

by

Fatemeh Esfahani
B.Sc., Iran University of Science and Technology, 2011
M.Sc., Iran University of Science and Technology, 2013

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

© Fatemeh Esfahani, 2021
University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.

Dense Subgraph Mining in Probabilistic Graphs

by

Fatemeh Esfahani
B.Sc., Iran University of Science and Technology, 2011
M.Sc., Iran University of Science and Technology, 2013

Supervisory Committee

Dr. Alex Thomo, Supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Supervisor
(Department of Computer Science)

Dr. Xuekui Zhang, Outside Member
(Department of Mathematics and Statistics)

ABSTRACT

In this dissertation we consider the problem of mining cohesive (dense) subgraphs in probabilistic graphs, where each edge has a probability of existence. Mining probabilistic graphs has become the focus of interest in analyzing many real-world datasets, such as social, trust, communication, and biological networks due to the intrinsic uncertainty present in them.

Studying cohesive subgraphs can reveal important information about connectivity, centrality, and robustness of the network, with applications in areas such as bioinformatics and social networks. In deterministic graphs, there exists various definitions of cohesive substructures, including cliques, quasi-cliques, k-cores and k-trusses. In this regard, k-core and k-truss decompositions are popular tools for finding cohesive subgraphs. In deterministic graphs, a k-core is the largest subgraph in which each vertex has at least k neighbors, and a k-truss is the largest subgraph whose edges are contained in at least k triangles (or $k-2$ triangles depending on the definition). The k-core and k-truss decomposition in deterministic graphs have been thoroughly studied in the literature. However, in the probabilistic context, the computation is challenging and state-of-art approaches are not scalable to large graphs. The main challenge is efficient computation of the tail probabilities of vertex degrees and triangle count of edges in probabilistic graphs. We employ a special version of central limit theorem (CLT) to obtain the tail probabilities efficiently. Based on our CLT approach we propose peeling algorithms for core and truss decomposition of a probabilistic graph that scales to very large graphs and offers significant improvement over state-of-the-art approaches. Moreover, we propose a second algorithm for probabilistic core decomposition that can handle graphs not fitting in memory by processing them sequentially one vertex at a time. In terms of truss decomposition, we design a second method which is based on progressive tightening of the estimate of the truss value of each edge based on h-index computation and novel use of dynamic programming. We provide extensive experimental results to show the efficiency of the proposed algorithms.

Another contribution of this thesis is mining cohesive subgraphs using the recent notion of nucleus decomposition introduced by Sarıyüce et al. Nucleus decomposition is based on higher order structures such as cliques nested in other cliques. Nucleus decomposition can reveal interesting subgraphs that can be missed by core and truss decompositions. In this dissertation, we present nucleus decomposition for probabilis-

tic graphs. The major questions we address are: How to define meaningfully nucleus decomposition in probabilistic graphs? How hard is computing nucleus decomposition in probabilistic graphs? Can we devise efficient algorithms for exact or approximate nucleus decomposition in large graphs?

We present three natural definitions of nucleus decomposition in probabilistic graphs: local, global, and weakly-global. We show that the local version is in PTIME, whereas global and weakly-global are #P-hard and NP-hard, respectively. We present an efficient and exact dynamic programming approach for the local case. Further, we present statistical approximations that can scale to bigger datasets without much loss of accuracy. For global and weakly-global decompositions we complement our intractability results by proposing efficient algorithms that give approximate solutions based on search space pruning and Monte-Carlo sampling. Extensive experiments show the scalability and efficiency of our algorithms. Compared to probabilistic core and truss decompositions, nucleus decomposition significantly outperforms in terms of density and clustering metrics.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

There are a number of people to whom I am greatly indebted. I would like to express my deep gratitude to my supervisors, Prof. Alex Thomo, and Prof. Venkatesh Srinivasan, for their generous support and great encouragement to conduct this research as well as their valuable comments to enhance the quality of the dissertation. They are the greatest and nicest supervisors in the world. Also, I am very grateful to Prof, Kui Wu, for his help, support, and guidance throughout my research studies. Moreover, I have a sincere gratitude to all of those with whom I have had the pleasure to work during my PhD program. I owe my deepest gratitude to my family for their endless love, support, and always believing in me. Finally, I am heartily grateful to my wonderful husband, Alireza, whose love and support carried me through the roughest times.

DEDICATION

This dissertation is dedicated to my dearest husband, my loving parents, and my close friends which have always supported me.

Chapter 1

Introduction

Probabilistic graphs are graphs in which each edge has a probability to exist in the graph (cf. [9, 10, 12, 35, 37, 74]). Many real-world graph networks such as social, trust, communication, and biological networks feature uncertainty. For instance, influence between users (cf. [29, 10, 37]) in a social network can be modeled as a probabilistic graph with probabilities on the edges representing the likelihood that some action of one user will be adopted by another. In terms of trust inference, probabilistic models with trust values as edge probabilities can be used to compute trust associated with a social relationship [42, 44]. In biological networks of protein-protein interactions (cf. [13]) an edge can be assigned a probability value representing the strength of prediction that a pair of proteins will interact in a living organism [19, 20, 72].

Mining dense subgraphs and discovering hierarchical relations between them is a fundamental problem in graph analysis tasks [46, 83]. For instance, it can be used in visualizing complex networks [93], finding correlated genes and motifs in biological networks [88, 28], detecting communities in social and web graphs [16, 27, 47], summarizing text [1], and revealing new research subjects in citation networks [64].

Core and truss decompositions are popular notions of dense subgraphs which can be computed in polynomial time. In deterministic graphs, the k-core of a graph G is defined as the largest subgraph in which each vertex has at least k neighbors within that subgraph, and the k-truss of G is the largest subgraph in which each edge is contained in at least k triangles (or $k-2$ in some works). The set of all k-cores and k-trusses of a graph, for various k, forms core and truss decomposition of that graph, respectively. A recent notion of dense subgraphs is *nucleus* introduced by Sarıyüce et al. [66, 67]. Nucleus decomposition is a generalization of core and truss decompositions that uses higher-order structures to detect dense regions. It can

reveal interesting subgraphs that can be missed by core and truss decompositions. In a nutshell, a k-(r, s)-nucleus is a maximal subgraph whose r-cliques are contained in at least k of s-cliques, where $s > r$. Finding all k-(r, s)-nuclei of a graph G for different values of k, forms nucleus decomposition of G.

Due to the intrinsic uncertainty in many networks such as social, biological,and communication networks, it is of great importance to study these three notions of dense subgraphs in a probabilistic context. However, in probabilistic graphs the computation is challenging and has received much less attention. State-of-art in core and truss decomposition for probabilistic graphs does not scale to very large datasets. On the other hand, nucleus decomposition for probabilistic graphs has not been studied yet. Thus, the main contribution of this dissertation is proposing efficient algorithms for finding core, truss and nucleus decomposition in probabilistic graphs.

1.1 Core Decomposition in Probabilistic Graphs

Core decomposition is a well-known concept for finding cohesive subgraphs which has a wide range of applications. For instance, it can be used in measuring structural diversity in social contagion [76], describing biological functions of proteins in protein-protein interaction networks [48], analyzing network structure's properties to explore collaboration in software teams [80], and also as a metric for sentence selection in text summarization [1]. Probabilistic core decomposition naturally extends all the applications of deterministic core decomposition to probabilistic graphs. Other applications which are showcased in [9] include facilitating influence maximization and task-driven team formation in probabilistic graphs.

In k-core computation the goal is to find the maximal (largest) subgraph of a graph in which each vertex has degree at least k within that subgraph. The set of all k-cores forms core decomposition of the graph [71], where k can be from 0 up to maximum degree of any vertex in the graph. Core decomposition in deterministic graphs has been thoroughly studied in literature [3, 17, 40, 54], and can be computed in $O(m)$ time, where m is the number of the edges in the input graph. However, in the probabilistic context, computing core decomposition is much more challenging.

Here we use the probabilistic (k, η)-core notion introduced by Bonchi, Gullo, Kaltenbrunner, and Volkovich in [9]. In (k, η)-core computation the goal is to find the maximal subgraph in which each vertex has at least k neighbors within that subgraph

with probability no less than $\eta \in [0, 1]$. Threshold η is given by user and defines the desired level of certainty of the output cores. A fundamental notion needed to compute the (k, η)-core is the η-degree of a vertex v. It is the maximum degree such that the probability for v to have that degree is no less than η.

1.1.1 Challenges and Contributions

A significant initial challenge is computing η-degrees of graph vertices. In [9], the η-degree of each vertex v is computed using dynamic programming (DP) which has a complexity of $O(d_v^2)$, where d_v is the number of edges incident to v. Unfortunately, in many real social and web networks, d_v can be in the millions and a quadratic algorithm such as DP is impractical.

Our first contribution is the design of an efficient method for computing η-degrees. Our method is based on Lyapunov's special version of the Central Limit Theorem [41, 51] and we show its output to be virtually indistinguishable from exact computation for vertices with a high number of incident edges.

While solving the challenge of computing η-degrees is an important step forward, we still need efficient algorithms to compute core decomposition for large probabilistic graphs. We propose two efficient algorithms to solve this problem.

The first one, which we call the "peeling algorithm" (PA), recursively deletes (peels-off) the vertex of the smallest degree. Our contribution here is in designing efficient arrays for storing important bookkeeping information. Handling these arrays becomes challenging because, differently from the deterministic case, the process of keeping the vertices sorted based on their changing η-degrees is more complicated and we need to shuffle information carefully in order to keep the arrays up to date. Notably, our PA algorithm scales to datasets two orders of magnitude bigger than those that the state-of-the-art algorithm [9] can handle.

For the case when the input graph does not fit in memory, we propose a sequential algorithm (SA) based on the vertex-centric model of computation. The main idea of the SA algorithm is to maintain an upper-bound, called vertex value, on the core number of each vertex. This upper-bound is initialized to be the η-degree of each vertex, and after each iteration of the algorithm it is tightened further until it reaches the exact core value. While being moderately slower than PA, there are two notable advantages associated with this algorithm. First, the SA algorithm has a memory footprint of $O(n)$ as opposed to $O(m)$ for PA, where n and m are the number of

vertices and edges, respectively. This amounts to SA requiring 30 to 40 times lower memory for social and web network graphs in practice. Second, as shown in our experiments, after only a fraction of iterations of the algorithm, we can obtain an approximation very close to the true core numbers of vertices.

In summary, our contributions are as follows.

- We introduce an efficient approach to compute η-degrees using Lyapunov's central limit theorem which gives very accurate approximations on the probability that a vertex can have a certain degree. We prove the accuracy of the approach and show that this method of computing probabilistic degree is numerically stable.
- We propose a peeling algorithm (PA) based on recursive vertex deletions which, by using carefully engineered array structures, is able to scale to graphs two orders of magnitude larger than what the state-of-the-art algorithm can handle.
- For the case when the input graph does not fit into memory, we propose a sequential algorithm (SA) to produce the core decomposition in probabilistic graphs with a low memory footprint. This algorithm can also produce accurate approximations after only a fraction of total iterations, a useful feature to have in applications when exact core numbers are not necessary.

1.2 Truss Decomposition in Probabilistic Graphs

In addition to core decomposition, the notion of *truss* is particularly suited to extracting a hierarchical structure of cohesive subgraphs [78]. Finding truss subgraphs can be useful in many application areas [34]. For instance, since k-trusses are dense subgraphs, they can be used to represent cohesive groups or communities in social and science networks. Moreover, k-truss can be used to efficiently compute maximum k-cliques. Other applications include visualization of complex networks [93] and community modeling [33].

Truss decomposition in deterministic graphs has been broadly studied in the literature [78, 33, 90]. In probabilistic graphs, computing truss decomposition has received much less attention due to its computational challenges. In this dissertation, we present efficient algorithms for finding truss decomposition in probabilistic context. We consider the notion of (k, η)-truss introduced in [34]. Specifically, we aim to

compute the largest subgraph in which each edge is contained in at least k triangles within that subgraph with probability no less than a user specified threshold η. The highest value of k for which an edge is part of a (k, η)-truss is called truss value of that edge. In the following we discuss the challenges associated with truss decomposition in probabilistic graphs as well as our proposed algorithms.

1.2.1 Challenges and Contributions

The standard approach to computing k-truss decomposition is the edge peeling process, which is based on continuously removing edges supported by less than k triangles (cf. [34]). This process is repeated after incrementing k until no edges remain [65], which results in finding all k-trusses for different values of k. In deterministic graphs, counting the number of triangles which contains an edge is straightforward. But, in probabilistic graphs, the triangles in which an edge might participate have combinatorial nature [34]. That is, each edge should have enough probability to participate in at least k triangles in the input graph. This probability is called support probability of the edge. As a result, the most difficult task, in the edge peeling process, is computing edge support probabilities efficiently. This becomes particularly important when the input graph is huge. Given an edge $e = (u, v)$, support probability of the edge can be computed using dynamic programming (DP) as proposed in [34]. The time complexity of computation using DP is $O((\min\{d(u), d(v)\})^2)$, where $d(u)$ and $d(v)$ is deterministic degree of u and v, respectively. However, this way of computation is not scalable to large probabilistic graphs.

Realizing the fact that each triangle in probabilistic graph can be defined as a Bernoulli random variable with an existence probability, we design a novel approach based on Lyapunov's special version of the Central Limit Theorem [41] to approximate probability distribution of the support of an edge. We show that the proposed approximation is accurate for our problem when the number of triangles is big. In addition, we derive an error bound on the approximation to ensure that the output probabilities are very close to the values obtained through exact computation.

We design a peeling algorithm for probabilistic k-truss decomposition. Our algorithm takes advantage of the fast calculation of edge support probabilities in time $O(\min\{d(u), d(v)\})$ using Central Limit Theorem. It also uses optimized array-based data structures for storing edge information of the graph.

While our first approach employs a statistical technique to approximate the sup-

port probability of the edges, it is a peeling algorithm, nevertheless. Edge peeling is associated with a major drawback: the edges have to be kept sorted by their current triangle support (count) at all times which requires maintaining global information of the graph at each step of the algorithm. This affects the scalability of the algorithm considerably. Furthermore, if we want to avoid statistical approximations, edge peeling becomes even more challenging, since the exact computation of the support probability of an edge e is done using dynamic programming. The process is repeated each time the edge loses a neighbor during the peeling approach. This approach does not scale as it involves many recomputations, especially when there are edges with many neighbors. This leads us to ask whether there is an *exact* and *scalable* approach to truss decomposition in probabilistic graphs.

We answer the above question positively by introducing an algorithm which extends the iterative h-index computation, recently introduced for deterministic graphs in [65], to probabilistic graphs.

In deterministic graphs, the triangle supports of the edges are obtained at the beginning, and each edge computes the h-index value for the list of its neighbors' triangle supports. Neighbors of an edge are those edges which form a triangle with the given edge. This process is repeated on these values until convergence to truss values occurs. Upon termination, the final h-index value of each edge equals its truss value. The authors in [65] prove that convergence of support values to the truss values is guaranteed.

Unfortunately, this idea does not work for probabilistic graphs, since it does not consider uncertainty in such graphs, thus resulting in wrong truss values. In this dissertation, we introduce an h-index updating algorithm that works for probabilistic graphs. In particular, we design a procedure which considers properties of truss subgraphs in probabilistic graphs and maintains proper upper-bounds on truss value of edges until convergence to true truss values.

In the following, we summarize all the contributions for computing truss decomposition in probabilistic graphs.

- We use statistical approximation based on Lyapunov central limit theorem to compute the triangle support probabilities of the edges in the input graph. We provide a theoretical analysis and obtain a tight error bound on the approximation, which shows that the higher the number of triangles supporting an edge, the higher the accuracy of the approximation results.

- We develop a peeling algorithm based on recursive edge deletions, which, by utilizing the central limit theorem is able to calculate truss decomposition in big probabilistic graphs not possible with the pure DP approach.

- We propose a second algorithm based on h-index updating which works for probabilistic graphs. Our proposed algorithm is exact with respect to final result, but also progressive allowing the user to see near-results along the way, and it works by processing only one edge and its immediate neighbors at a time, resulting in smaller memory required by the algorithm.

- While proving the correctness of the algorithm, we obtain an upper-bound on the number of iterations that the algorithm needs for convergence. It shows that the convergence to truss values can be obtained after a finite number of iterations.

- We evaluate the performance of our approaches on a wide range of datasets. Our experimental results confirm the scalability and efficiency of our exact algorithm, significantly outperforming the exact algorithm in [34] for large datasets. Furthermore, comparisons with the proposed approximate algorithm show that the running time of our exact algorithm is very close to that of the approximate algorithm. It is indeed surprising that we can achieve efficiency without sacrificing the exactness of the solution.

1.3 Nucleus Decomposition in Probabilistic Graphs

Recently, Sarıyüce et al. introduced nucleus decomposition, which is based on r-cliques contained in s-cliques, where $s > r$. In deterministic graphs, a k-(r, s)-nucleus is a maximal subgraph in which each r-clique is part of at least k number of 4-cliques. For $r = 1, s = 2$ and $r = 2, s = 3$ we obtain the notions of k-core and k-truss, respectively. For $r = 3, s = 4$, r-cliques are *triangles*, s-cliques are 4-*cliques*, and k-$(3, 4)$-nucleus is strictly stronger than k-truss and k-core. Sarıyüce et al. in [66, 67] observed that, in practice, k-$(3, 4)$-nucleus is the most interesting in terms of the quality of subgraphs produced for a large variety of graphs. As such, in this dissertation we also focus on this decomposition.

As pointed out by [66, 67], nucleus decomposition can uncover a finer grained structure of dense groups not possible using other dense subgraph mining methods;

as such, nucleus decomposition can be beneficial for a large variety of applications, e.g. community structure discovery [70], mining dense regions in internet of things [95], financial fraud detection [89], extracting brain connectome subgraph hierarchy [84], detection of complexes in biological networks [52], etc. All these applications of nucleus decomposition extend naturally to the probabilistic networks. Ignoring probabilities and using deterministic methods amounts to setting all probabilities to 1, which not only misses salient information, but could prove detrimental in applications such as finding cohesive subnetworks of proteins from probabilistic PPI networks which has valuable implications to disease diagnosis [19]. Last but not the least, computing probabilistic nucleus is highly beneficial for task driven team formation in probabilistic social networks, demonstrated later in our case study in Chapter 5 using a DBLP network. Also, we show how probabilistic nucleus performs in detecting the proteins that interact with COVID-19 in Chapter 5.

1.3.1 Challenges and Contributions

We are the first to study nucleus decomposition in probabilistic graphs. The major questions we address are: How to define meaningfully nucleus decomposition in probabilistic graphs? How hard is computing nucleus decomposition in probabilistic graphs? Can we devise efficient algorithms for exact or approximate nucleus decomposition in large graphs?

Definitions. We start by introducing three natural notions of probabilistic nucleus decomposition. They are based on the concept of *possible worlds* (PW's), which are instantiations of a probabilistic graph obtained by flipping a biased coin for each edge independently, according to its probability. We define *local*, *global*, and *weakly-global* notions of nucleus as a maximal probabilistic subgraph $\mathcal{H}$ satisfying different structural conditions for each case.

In the local case, we require a good number of PW's of $\mathcal{H}$ to satisfy a high level of density around each triangle (in terms of 4-cliques containing it) in $\mathcal{H}$. This is local in nature because the triangles are considered independently of each other. To contrast this, we introduce the global notion, where we request the PW's themselves be deterministic nuclei. This way, not only do we achieve density around each triangle but also ensure the same is achieved for all the triangles of $\mathcal{H}$ simultaneously. Finally, we relax this strict requirement for the weakly-global case by requiring that PW's only contain a deterministic nucleus that includes the triangles of $\mathcal{H}$.

Global and Weakly-Global Cases. We show that computing global and weakly-global decompositions are intractable, namely #P-hard and NP-hard, respectively. We complement these results with efficient algorithms for these two cases that give approximate solutions based on search space pruning combined with Monte-Carlo sampling.

Local Case. We show that local nucleus decomposition is in PTIME. The main challenge is to compute the probability of each triangle to be contained in k 4-cliques. We present a dynamic programming (DP) solution for this task, which combined with a triangle peeling approach, solves the problem of local nucleus decomposition efficiently. While this is welcome result, we further propose statistical methods to speed-up the computation. Namely, we provide a framework where well-known distributions, such as Poisson, Normal, and Binomial, can be employed to approximate the DP results. We provide detailed conditions under which the approximations can be used reliably, otherwise DP is used as fallback. This hybrid approach speeds-up the computation significantly and is able to handle datasets, which DP alone cannot.

Experiments. We present extensive experiments which show that our DP method for local nucleus decomposition is efficient and can handle large datasets; when combined with our statistical approximations, the process is significantly sped-up and can handle much larger datasets. We demonstrate the importance of nucleus decomposition by comparing it to probabilistic core and truss decomposition using density and clustering metrics. The results show that nucleus decomposition significantly outperforms core and truss decompositions in terms of these metrics.

1.4 Selected Publications

In the following, I have listed my publications which show the outcome of my research and collaboration, as the main author, with my supervisors and lab mates.

- [26] **F. Esfahani**, V. Srinivasan, A. Thomo, and K. Wu,"Nucleus Decomposition in Probabilistic Graphs: Hardness and Algorithms", *to appear in* ICDE, 2022.

- [23] **F. Esfahani**, M. Daneshmand, V. Srinivasan, A. Thomo, and K. Wu, "Truss decomposition on large probabilistic networks using h-index", In 33rd International Conference on Scientific and Statistical Database Management (SSDBM), pages 145–156, 2021.

- [24] **F. Esfahani**, V. Srinivasan, A. Thomo, and K. Wu, "Efficient Computation of Probabilistic Core Decomposition at Web-Scale", In 22nd International Conference on Extending Database Technology (EDBT), pages 325–336, 2019.
- [25] **F. Esfahani**, J. Wu, V. Srinivasan, A. Thomo, and K. Wu, "Fast Truss Decomposition in Large-scale Probabilistic Graphs", In 22nd International Conference on Extending Database Technology (EDBT), pages 722–725, 2019.

1.5 Dissertation Outline

The outline of this dissertation is as follows.

Chapter 1 contains introduction to three notions of dense sugraphs; core, truss, and nucleus as well as contributions in this dissertation followed by the selected publications and an overview of the structure of the dissertation.

Chapter 2 puts focus on core decomposition in probabilistic graphs. In this chapter we introduce the problem formulation, provide an overview of the existing work in the literature, and solve the problem using our propose algorithms. This chapter reflects my research published in [24].

Chapter 3 focuses on truss decomposition in probabilistic graphs using an approximation method that we propose. We introduce truss in probabilistic graphs, and discuss about our approach for solving truss decomposition in probabilistic context. This chapter represents my work in [25].

Chapter 4 addresses probabilistic truss decomposition using h-index iterative computation. We explain our proposed h-index approach in probabilistic graphs. Formal proofs are given to show the accuracy of the algorithm and convergence to true truss values of edges in an input graph. This chapter contains my research published in [23].

Chapter 5 contains formulation of three notions of probabilistic nucleus decomposition, namely *local*, *global*, and *weakly-global*. We prove the hardness result for global and weakly-global decompositions, and propose efficient algorithms based on Monte-Carlo sampling. Moreover, we show that local nucleus decomposition is in PTIME and propose two methods for finding all local nuclei. This chapter includes the work under submission in [26].

Chapter 6 concludes the dissertation and discusses the potential future work.

Chapter 2

Efficient Computation of Probabilistic Core Decomposition at Web-Scale

This chapter starts with the overview of literature work on core decomposition. Next, we introduce the problem statement and the proposed methods, followed by our experimental results and conclusions.

2.1 Related Work

Among different notions of cohesive subgraphs, k-core is one of the most popular (cf. [1, 40, 48, 76, 80]. In deterministic graphs, the computation of k-core has been well studied. Batagelj and Zaversnik [3] give an efficient peeling algorithm for deterministic core decomposition. Montresor et al. [54] give a distributed algorithm for deterministic core decomposition and introduce the concept of locality-based bound tightening. Wen et al. [79] propose I/O efficient core decomposition algorithms which only allow node information to be loaded in memory. Khaouid et al. [40] consider deterministic core decomposition of large networks on a single PC. Sarıyüce et al. [63] propose incremental k-core decomposition algorithms for dynamic graph data, in which edges are added/deleted on a regular basis. In a similar setting, a distributed k-core decomposition and maintenance algorithms are proposed in [2]. Core decomposition in large temporal graphs is addressed in [81].

For probabilistic graphs, the generalization of k-core is the notion of (k, η)-core

introduced by Bonchi et al. [9], which we discuss in detail throughout this chapter.

2.2 Background

Cores of deterministic graphs. Let $G = (V, E)$ be an undirected graph, where V is a set of n vertices, and E is a set of m edges. For vertex $v \in V$, let $N_G(v)$ be the set of v's neighbors: $N_G(v) = \{u : (u, v) \in E\}$. The (deterministic) degree of v in G, is equal to $|N_G(v)|$.

Given $V' \subseteq V$, and $E_{V'} = \{(u, v) \in E : u, v \in V'\}$, graph $H = (V', E_{V'})$ is called the subgraph of G induced by V'. Let $k \in [0, d_{\max}(G)]$, where $d_{\max}(G)$ is the maximum vertex degree in G. The k-core of G is defined as the maximal induced subgraph $C_k(G) = (V', E_{V'})$ in which each vertex $v \in V'$ has degree of at least k. The set of all k-cores forms the core decomposition of G. Core decomposition of G is unique and it satisfies the following relation [71]: $G = C_0(G) \supseteq \cdots \supseteq C_{d_{\max}(G)-1} \supseteq C_{d_{\max}(G)}$. The coreness (or core number) of a vertex is defined as the maximum value of k such that the corresponding $C_k(G)$ contains v.

Probabilistic graphs. A probabilistic graph is a triple $\mathcal{G} = (V, E, p)$, where V and E are as before and $p : E \to (0, 1]$ is a function that maps each edge $e \in E$ to its existence probability p_e. For each vertex $v \in V$, the set of edges incident to v is denoted by $\mathcal{N}_v$. $d_v = |\mathcal{N}_v|$ is the number of all edges incident to v which is equal to the deterministic degree of v. In the most common probabilistic graph model (cf. [9, 35, 38]), the existence probability of each edge is assumed to be independent of other edges.

In order to analyze probabilistic graphs, we use the concept of *possible worlds* that are deterministic graph instances of $\mathcal{G}$ in which only a subset of edges appears. The probability of a possible world $G \sqsubseteq \mathcal{G}$ is obtained as follows:

$$\Pr(G) = \prod_{e \in E_G} p_e \prod_{e \in E \setminus E_G} (1 - p_e). \tag{2.1}$$

In a probabilistic graph $\mathcal{G}$, the notion of η-*degree*, where $\eta \in [0, 1]$, denoted by η-$deg(v)$, of a vertex v is defined in [9] as the maximum k for which $\sum_{G \sqsubseteq \mathcal{G}} \Pr[d_G(v) \geq k] \geq \eta$, where $k = 0, \ldots, d_v$, and the probability is taken over all the possible worlds $G \sqsubseteq \mathcal{G}$.

For instance, consider Figure 2.1. When $\eta = 0.5$, vertex 6 has degree at least

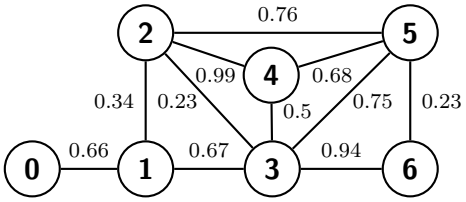


Figure 2.1: A probabilistic graph.

2 with probability $0.94 \cdot 0.23 = 0.2162$ (product of probabilities that each of the two edges is in a possible world), and it has degree at least 1 with the probability $1-((1-0.94)\cdot(1-0.23)) = 0.9538$ (complementary probability that none of the two edges is in a possible world) which is greater than the threshold. Thus, the η-degree of vertex 6 is 1.

In the rest of the dissertation, we use $\Pr[d(v) \geq k]$ to denote $\sum_{G \sqsubseteq \mathcal{G}} \Pr[d_G(v) \geq k]$. The value of $\Pr[d(v) \geq k]$ decreases with the increase of k. Note that d_v is different from $d(v)$; the former is constant, whereas the latter is a random variable.

Cores of probabilistic graphs. In order to extend k-core decomposition to probabilistic graphs, the notion of (k,η)-*core* is defined in [9]:

Definition 1. *Given a probabilistic graph $\mathcal{G} = (V, E, p)$, and a threshold $\eta \in [0,1]$, (k,η)-core is the maximal induced subgraph $C_{(k,\eta)}(\mathcal{G}) = (V', E_{V'}, p)$ in which the η-degree of each vertex $v \in V'$ is at least k. The set of all (k,η)-cores forms the core decomposition of $\mathcal{G}$.*

The core decomposition in probabilistic graphs is unique, and the (k,η)-cores are nested into each other similar to the deterministic case. The highest value of k for which v belongs to a (k,η)-core is called η-*core* number or probabilistic coreness of v.

Computing η-degrees using Dynamic Programming. We have that $\Pr[d(v) \geq k] = 1 - \sum_{i=0}^{k-1} \Pr[d(v) = i]$. One way of computing $\Pr[d(v) = i]$ is to use dynamic programming as proposed in [9]. However, this method of computing the η-degree has complexity of $O(d_v^2)$ for a vertex v of deterministic degree d_v. This is not practical when d_v is big, say over 20 thousand, which occurs often in all our datasets. In fact, DP cannot finish in reasonable time even for one such vertex. In addition, web-scale graphs normally have millions of nodes with moderate-high degree (e.g., a thousand or more), and if DP is applied to every such node, the total processing time increases considerably. In the next section we introduce an alternative way for fast computation of the η-degree of a vertex v using Lyapunov central limit theorem.

2.3 Computing η-Degrees using Central Limit Theorem

In this section, we first show how a special version of Central Limit Theorem (CLT) can be applied to accurately estimate $\Pr\big[d(v) \geq k\big]$. Then, we show theoretical bounds on the accuracy of this approximation. Specifically, we show that CLT can produce a very accurate approximation to tail probabilities of the vertex degree.

CLT, one of the most important theorems in statistics, states that given a set of random variables, their properly scaled sum converges to a normal distribution under certain conditions. There are different versions of CLT, with the most common one focusing on independent identically distributed (i.i.d.) random variables. In this dissertation, we consider a variant called Lyapunov CLT [50, 51] that can be applied when random variables are independent, but not necessarily identically distributed. The Lyapunov's condition imposes a limit on the growth of the third absolute central moment of each random variable in the given sequence ensuring the convergence of the normalized sum of that sequence to standard normal distribution. Formally, we have:

Theorem 1. ***Lyapunov CLT [41].*** *Let $\xi_1, \xi_2, \cdots, \xi_n$ be a sequence of independent, but non-identically distributed random variables, each with finite expected value μ_k and variance σ_k. Let*

$$s_n^2 = \sum_{k=1}^{n} \sigma_k^2, \tag{2.2}$$

Lyapunov CLT states that if

$$\lim_{n\to\infty} \frac{1}{s_n^{2+\delta}} \sum_{k=1}^{n} \mathrm{E}[|\xi_k - \mu_k|^{2+\delta}] = 0, \tag{2.3}$$

for some $\delta > 0$, then $\frac{1}{s_n}\sum_{k=1}^{n}(\xi_k - \mu_k)$ converges in distribution to a standard normal random variable.

Equation (2.3) is called Lyapunov's condition which in practice is usually tested for the special case $\delta = 1$. The proof for this theorem can be found in [5, 18].

Computing η-degrees using Lyapunov CLT. In what follows, we show how Lyapunov CLT can help compute $\Pr\big[d(v) \geq k\big]$, for each vertex v of the input probabilistic graph $\mathcal{G}$.

Recall that for each vertex v we have a set of edges incident to v denoted by $\mathcal{N}_v$. Each e_i in $\mathcal{N}_v$ has existence probability p_i, which is independent of the other edge probabilities in $\mathcal{G}$. Corresponding to each edge e_i in $\mathcal{N}_v$, we define a Bernoulli random variable X_i which takes on 1 with probability p_i, and 0 with probability $(1 - p_i)$. Formally,

$$X_i = \begin{cases} 1 & \text{if edge } e_i \text{ incident to } v \text{ exists in the graph} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Since X_i is a Bernoulli random variable, we know that $\mathrm{E}[X_i] = \mu_i = p_i$ and $\mathrm{Var}[X_i] = p_i(1 - p_i)$. Using the fact that $d(v) = \sum_{i=1}^{d_v} X_i$, we have:

$$\Pr[d(v) \geq k] = \Pr\left[\sum_{i=1}^{d_v} X_i \geq k\right]. \tag{2.5}$$

According to Equation (2.5), finding the probability that a vertex v is of the degree at least k is equivalent to computing the probability that the sum of X_i's for v is at least k. In addition, Bernoulli random variables X_i's are independent, but not identically distributed. Thus, if condition (2.3) is satisfied and if d_v is large enough, we can conclude that $\frac{1}{s_{d_v}} \sum_{i=1}^{d_v} (X_i - \mu_i)$ has standard normal distribution, where $s_{d_v} = \sqrt{\sum_{i=1}^{d_v} p_i(1 - p_i)}$. To compute $\Pr[\sum_{i=1}^{d_v} X_i \geq k]$, we can subtract $\sum_{i=1}^{d_v} \mu_i$ from both sides of the inequality, and then divide by s_{d_v} which results in:

$$\Pr\Big[\sum_{i=1}^{d_v} X_i \geq k\Big] = \Pr\Big[\frac{1}{s_{d_v}} \sum_{i=1}^{d_v} (X_i - \mu_i) \geq \frac{1}{s_{d_v}} (k - \sum_{i=1}^{d_v} \mu_i)\Big]. \tag{2.6}$$

Using Lyapunov CLT, and setting

$$Z = \frac{1}{s_{d_v}} \sum_{i=1}^{d_v} (X_i - \mu_i), \tag{2.7}$$

we can say that Z has standard normal distribution. Thus

$$\Pr\big[d(v) \geq k\big] \cong \Pr\big[Z \geq z\big], \tag{2.8}$$

where $z = \frac{1}{s_{d_v}} (k - \sum_{i=1}^{d_v} \mu_i)$. In fact, since Z in Equation (2.7) has standard normal distribution, using the complementary cumulative distribution function [100], we can efficiently evaluate the right hand side of Equation (2.8). To find the η-degree we

start with $k = 1$, and approximate $\Pr[d(v) \geq k]$ using Lyapunov CLT, finding the maximum k for which the probability is above threshold η.

We can apply Theorem 1 provided that Lyapunov's condition is satisfied. By setting $\delta = 1$ in Equation (2.3) we show that this condition holds for a sequence of non-identically distributed Bernoulli random variables.

Theorem 2. *Given a sequence of random variables $X_i \sim$ Bernoulli(p_i), for $i \in [1, n]$, the Lyapunov's condition (2.3) for $\delta = 1$ is satisfied whenever $s_n^2 = \sum_{k=1}^{n} p_k(1-p_k) \to \infty$.*

Proof. For each Bernoulli random variable X_i we know that $\sigma_i^2 = p_i(1 - p_i)$, and $\mu_i = p_i$. Therefore, $s_n^2 = \sum_{k=1}^{n} p_k(1 - p_k)$ according to the equation (2.2). On the other hand, when $\delta = 1$, $\mathrm{E}[|X_k - \mu_k|^3]$ is computed as follows:

$$\begin{aligned} \mathrm{E}[|X_k - \mu_k|^3] &= p_k(1 - p_k)^3 + (1 - p_k)p_k^3, \\ &= p_k(1 - p_k)[(1 - p_k)^2 + p_k^2] \leq \sigma_k^2, \end{aligned} \tag{2.9}$$

where in inequality (2.9) we have used the fact that $(1 - p_k)^2 + p_k^2 \leq 1$. Thus, $\sum_{k=1}^{n} \mathrm{E}[|X_k - \mu_k|^3] \leq s_n^2$. Substituting this in the Lyapunov's condition (2.3) for $\delta = 1$, we conclude that the condition is satisfied whenever $s_n^2/s_n^3 \to 0$ as $n \to \infty$ which means that s_n should go to infinity as n increases. This is equivalent to $s_n^2 = \sum_{k=1}^{n} p_k(1 - p_k) \to \infty$, and as a result the theorem follows. In other words, since we have

$$s_n^2 = \sum_{k=1}^{n} p_k(1 - p_k) = \sum_{k=1}^{n} \sigma_k^2 \geq n\sigma_{\min}^2, \tag{2.10}$$

for large n and as long as $\sigma_{\min}^2 = \min_k\{\sigma_k^2\}$ is away from zero, $n\sigma_{\min}^2 \to \infty$ which results in $s_n^2 \to \infty$ as n approaches infinity. □

Accuracy of the Approximation. In order to show the accuracy of the approximation, we refer to the Berry–Esseen theorem [101]: For a given sequence $Y_1, Y_2, ...$ of non identically distributed and independent random variables with $E(Y_i) = 0$, $E(Y_i^2) = \sigma_i^2$, and $E(|Y_i^3|) = \rho_i < \infty$, there exists a constant C_0 such that the following inequality is satisfied for all n:

$$\mathsf{sup}_{x \in \mathbb{R}} \, |F_n(x) - \Phi(x)| \leq C_0 \cdot \psi_0, \tag{2.11}$$

where F_n is the cumulative distribution of $S_n = \frac{Y_1 + Y_2 + \cdots + Y_n}{\sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2}}$, which is the sum of Y_i's standardized by the variances, and Φ is the cumulative distribution of the standard

normal distribution. In the above inequality ψ_0 is a function given by

$$\psi_0 = \psi_0(\overrightarrow{\sigma}, \overrightarrow{\rho}) = \left(\sum_{i=1}^{n} \sigma_i^2\right)^{-3/2} \cdot \sum_{i=1}^{n} \rho_i. \tag{2.12}$$

where $\overrightarrow{\sigma} = (\sigma_1, \cdots, \sigma_n)$, and $\overrightarrow{\rho} = (\rho_1, \cdots, \rho_n)$ are the vectors of σ_i's and ρ_i's respectively. It should be noted that the best upper-bound obtained so far for C_0 is 0.56 [73].

Based on the Berry–Esseen theorem, the more the number of edges incident to a vertex, the better the accuracy of the results. In the following corollary, we show how to obtain an upper-bound on the maximal error while approximating the true distribution of the sum of X_i's with the normal distribution.

Corollary 1. *For each vertex v in the probabilistic graph $\mathcal{G}$ with X_i's being Bernoulli random variables as defined in (2.4), where $i = 1, \ldots, d_v$, the error bound on the approximation of the right-hand side of Equation (2.8) to the standard normal distribution is given as follows:*

$$\mathit{sup}_{x\in\mathbb{R}} |F_{d_v}(x) - \Phi(x)| \leq \frac{0.56}{\sqrt{p_1(1-p_1) + \cdots + p_{d_v}(1-p_{d_v})}} \tag{2.13}$$

Proof. Setting $Y_i = X_i - p_i$ in equation (2.7), we can apply the Berry–Essen theorem for random variables $Y_1, Y_2, \cdots, Y_{d_v}$, since for each Y_i, $\mathrm{E}[Y_i] = 0$. In addition,

$$\begin{aligned} \mathrm{E}[Y_i^2] &= \mathrm{E}[(X_i - p_i)^2] = \mathrm{Var}[X_i] = \sigma_i^2 = p_i(1-p_i), \\ \mathrm{E}[\left|Y_i^3\right|] &= \mathrm{E}[|X_i - p_i|^3] = \rho_i = (1-p_i)^3 p_i + p_i^3(1-p_i) \\ &= p_i(1-p_i)[(1-p_i)^2 + p_i^2] < \infty. \end{aligned} \tag{2.14}$$

It should be noted that the random variable

$$S_{d_v} = \frac{(X_1 - p_1) + (X_2 - p_2) + \cdots + (X_{d_v} - p_{d_v})}{\sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_{d_v}^2}} \tag{2.15}$$

in the Berry–Essen theorem is the same as the random variable Z in Equation (2.7).

Substituting σ_i^2 and ρ_i in Equation (2.12), we obtain:

$$\begin{aligned}\psi_0 &= \psi_0(\overrightarrow{\sigma}, \overrightarrow{\rho}) \\ &= \left(\sum_{i=1}^{d_v} p_i(1-p_i)\right)^{-3/2} \cdot \left(\sum_{i=1}^{d_v} p_i(1-p_i)[(1-p_i)^2 + p_i^2]\right), \end{aligned} \tag{2.16}$$

Using the fact that $1 = (1-p_i+p_i)^2 = (1-p_i)^2 + p_i^2 + 2p_i(1-p_i) \geq (1-p_i)^2 + p_i^2$, we can simplify (2.16) to have:

$$\begin{aligned}\psi_0 &\leq \left(\sum_{i=1}^{d_v} p_i(1-p_i)\right)^{-3/2} \cdot \left(\sum_{i=1}^{d_v} p_i(1-p_i)\right) \\ &= \left(\sum_{i=1}^{d_v} p_i(1-p_i)\right)^{-1/2} = \frac{1}{\sqrt{p_1(1-p_1) + \cdots + p_{d_v}(1-p_{d_v})}}, \end{aligned} \tag{2.17}$$

Substituting (2.17) in (2.11) the stated claim follows. □

2.4 Peeling Algorithm (PA)

In this section we propose a graph peeling algorithm (PA). Graph peeling, that is, (1) recursively deleting the vertex v of smallest degree (2) setting v's coreness to be equal to its degree at the time of deletion, and (3) updating the degrees of v's neighbors while keeping them sorted, is a general idea that has been used broadly in core decomposition of deterministic graphs (cf. [3, 40]). However, it requires substantial algorithmic engineering in order to achieve high scalability when applied to probabilistic graphs. This is because when a vertex v is deleted in the peeling process, updating the η-degrees of v's neighbors and maintaining the data structures up to date are non-trivial. In order to tackle these challenges, we use efficient *array structures* and *lazy updates*, which delay updating the η-degrees of v's neighbors for as long as possible.

Computing and updating η-degrees. An expensive step in the peel-off process is computing initial η-degrees and updating them for those vertices that lose neighbors in a peel-off step. We will depart from the feature of the deterministic case of having vertices sorted at all times based on their current degrees and allow instead the vertices to not be on their precise order as long as at the time of their removal we can fix things up using the key functions and data structures that we define.

More specifically, the computation and update of η-degrees is delayed as much as possible by using easy to compute lower-bounds on η-degrees instead of exact values. It is only when a vertex is candidate for removal that we compute its exact η-degree. At that time the remaining graph is typically much smaller and the computation becomes significantly faster.

Based on Corollary 1, we know that Lyapunov CLT gives a very good approximation on tail probabilities of vertex degrees. We use the values produced by CLT (decremented by a small epsilon) in order to obtain lower-bounds on η-degrees. For simplicity of exposition, we refer to the lower-bound values simply as *values*.

One important difference between the core decomposition of probabilistic and deterministic graphs is that the η-degree of a vertex can decrease by *at most one* when a neighbor is removed (according to Lemma 2 in [9]) as opposed to *exactly one* in deterministic graphs. An array $\mathbf{A}$ stores, for each value in the input graph, the set of vertices with that value. In the PA algorithm at each iteration, we decrease the value of a vertex v by one if a v's neighbor is removed. When v's turn comes to be removed and processed, we compute its η-degree and if it is the same as its current value we remove v. Otherwise, we repeatedly swap v to the proper place in $\mathbf{A}$. There could be several neighbor removals that did not change the η-degree of v, thus the proper place of v in $\mathbf{A}$ could be far from the next block of vertices, and therefore we might need to perform several swaps.

PA algorithm description. The PA algorithm is given in Algorithm 1. There we have arrays $\mathbf{d}$, $\mathbf{b}$, $\mathbf{p}$, and $\mathbf{A}$. For an example see Figure 2.1 and Table 2.1. Vertices in the PA algorithm are assumed to be labeled by numbers 0 to $n-1$. Array $\mathbf{d}$ initially stores for each vertex the lower-bound on the η-degree of that vertex. For instance, vertex 1 has a lower-bound of 1 in Table 2.1, so $\mathbf{d}[1] = 1$. Initially, vertices are stored in $\mathbf{A}$ in ascending order of their lower-bounds. We have colored $\mathbf{A}$ in shades of green in Table 2.1. The first block in $\mathbf{A}$ contains all the vertices with a lower-bound equal to 0; the second block contains vertices with lower-bound equal to 1, and so on. In order to determine the index boundaries of such blocks in $\mathbf{A}$, we define array $\mathbf{b}$ which stores the index boundaries of the vertex blocks in $\mathbf{A}$. In Table 2.1 we have for instance $\mathbf{b}[1] = 2$ and $\mathbf{b}[2] = 6$. In order to handle swapping efficiently we define an array $\mathbf{p}$ which stores the position of each vertex in $\mathbf{A}$. For instance, vertex 6 is at position 2 in $\mathbf{A}$, therefore $\mathbf{p}[6] = 2$.

There are two additional arrays as well; **gone** and **valid**. Since we do not remove vertices physically from the graph, we use array **gone** to keep track of the removed

Index	0	1	2	3	4	5	6
d	0	1	1	2	1	1	0
A	0	6	1	2	4	5	3
p	1	3	4	7	5	6	2
b	0	2	6				
valid	false	false	false	false	false	false	false
gone	false	false	false	false	false	false	false

*Table 2.1: Arrays **d**, **b**, **A**, **p**, **valid**, and **gone** in the PA algorithm for the graph in Figure 2.1.*

Index	0	1	2	3	4	5	6
d	1	1	1	2	1	1	0
A	6	0	1	2	4	5	3
p	2	3	4	7	5	6	1
b	0	1	6				
valid	true	false	false	false	false	false	false
gone	false	false	false	false	false	false	false

Table 2.2: First step of the PA algorithm (after swapping vertex 0) for the graph in Figure 2.1.

vertices at each step of the algorithm. Array **valid** tells for each vertex v if the η-degree of v is the same as the value $\mathbf{d}[v]$. The array **gone** is initially set to false for all the vertices, because none of the vertices has been removed yet. Array **valid** is set to all-false vector because all the vertices are on their lower-bound at the beginning of the algorithm. For instance, in Table 2.1, we have $\mathbf{valid}[4] = false$ which indicates that vertex 4 is on its lower-bound and its η-degree should eventually be computed.

We illustrate a few steps of the PA algorithm on the graph in Figure 2.1. We set $\eta = 0.5$. The PA algorithm starts with the first vertex v in array **A**, checking whether v is on its lower-bound or its η-degree is available. As can be seen in Table 2.1, vertex 0 is the first vertex in **A**. Since $\mathbf{valid}[0] = false$, the vertex 0 is on its lower-bound, and its η-degree might be higher. Therefore, it might not be its turn to be removed. As such, the compute_swap function is called (line 19, Algorithm 1) to compute the η-degree, and then do the required number of swaps to the right, using Algorithm 3, to place the vertex in the proper block of **A**. *Thus, unlike deterministic case we do need to perform several swaps to the right.* The compute_swap function (Algorithm 4) computes the η-degree of vertex 0, which turns out to be 1. Thus, the vertex only needs one swap to the right in **A** to be placed in the block containing all the vertices

Index	0	1	2	3	4	5	6
d	1	1	1	2	1	1	1
A	6	0	1	2	4	5	3
p	2	3	4	7	5	6	1
b	0	0	6				
valid	true	false	false	false	false	false	true
gone	false	false	false	false	false	false	false

Table 2.3: Second step of the PA algorithm executed on the graph in Figure 2.1.

with degree 1. In order to do each swap in constant time (using Algorithm 3), we swap vertex 0 with the last vertex in the same block which is 6. As a result, the positions of vertices 0 and 6 should be swapped as well: $\mathbf{p}[0] = 2, \mathbf{p}[6] = 1$. Vertex 0 becomes the last element of its current block (the block of vertices with degree 0). In order to make vertex 0 become an element of the next block, the index of the block of vertices with degree 1, $\mathbf{b}[1]$, is decremented by one to include vertex 0 as its first element. We have $\mathbf{b}[1] = 1$, and vertex 0 becomes the first element of the block of vertices with degree 1.

Table 2.2 shows the current status of arrays after swapping vertex 0 with 6. The updated values are shown in red color. As can be seen, $\mathbf{valid}[0] = true$ because now the η-degree of vertex 0 is available. Now, vertex 6 is the first vertex in $\mathbf{A}$ and the algorithm processes it. The summary of the results is shown in Table 2.3. Since $\mathbf{valid}[6] = true$ we can safely remove vertex 6. The corresponding index in array **gone** is set to true, and the coreness of vertex 6 is set to $\mathbf{d}[6]$, which is 1.

When a vertex v is removed, we process those neighbors u of v with a higher degree (lower-bound or exact) than v's (see lines 14-16), and decrement their degrees by one. Therefore, these neighbors should be moved one block to the left in $\mathbf{A}$. This is done in constant time using the swap_left function (line 15) which is shown in Algorithm 2. For instance, vertex 3 is a neighbor of vertex 6 with degree 2, which is decremented by one, from 2 to 1, when vertex 6 is removed. Therefore, vertex 3 should be swapped to the block in the left in $\mathbf{A}$, which contains vertices of degree 1. The process is similar to swapping to the right. However, when swapping to the left, vertex u is swapped with the first vertex, w in the same block in $\mathbf{A}$. In addition, the positions of u and w are swapped in $\mathbf{p}$. Then, the block index in $\mathbf{b}$ is incremented by one (line 7, Algorithm 2), making u the last element of the previous block.

Correctness of the algorithm. For every $v \in V$, and $C \subseteq V$, a vertex property

Algorithm 1 PA k-core computation function

```
1: function K_CoreCompute(Graph 𝒢, η)
2:     initialize(d, b, p, A, 𝒢)
3:     gone ← False                                   ▷ all-false vector
4:     valid ← False                                  ▷ all-false vector
5:     i ← 1
6:     while i < n do
7:         v ← A[i]
8:         if valid[v] = true then
9:             gone[v] ← true
10:            for all u : (u, v) ∈ 𝒩_v do
11:                if d[u] = d[v] then
12:                    if valid[u] = false then
13:                        compute_swap(d, p, b, A, u, valid)
14:                if d[u] > d[v] then
15:                    swap_left(d, p, b, A, u)
16:                    valid[u] ← false
17:            i + +
18:        else
19:            compute_swap(d, p, b, A, v, valid)
20:    return d
```

Algorithm 2 PA swap to left function

```
1: function SWAP_LEFT(d, p, b, A, u)
2:     du ← d[u], pu ← p[u]
3:     pw ← b[du] , w ← A[pw]
4:     if u ≠ w then
5:         A[pu] ← w, A[pw] ← u
6:         p[u] ← pw, p[w] ← pu
7:     b[du] + +, d[u] − −
```

Algorithm 3 PA swap to right function

```
1: function SWAP_RIGHT(d, p, b, A, u)
2:     du ← d[u], pu ← p[u]
3:     pw ← b[du + 1] − 1 , w ← A[pw]
4:     if u ≠ w then
5:         A[pu] ← w, A[pw] ← u
6:         p[u] ← pw, p[w] ← pu
7:     b[du + 1] − −, d[u] + +
```

Algorithm 4 η-degree computation and swap function

```
1: function COMPUTE_SWAP(d, p, b, A, u, valid)
2:     η_deg ← compute η-deg(u)
3:     valid[u] ← true
4:     diff ← η_deg − d[u]
5:     for j ← 1 to diff do
6:         swap_right(d, p, b, A, u)
```

function [3] is a function $\phi(v, C) : V \times 2^V \to \mathbb{R}$, and it is monotonic if $\forall C_1, C_2 \subseteq V$: $C_1 \subseteq C_2$ implies that $\phi(v, C_1) \leq \phi(v, C_2)$. According to the result by Batagelj and Zaversnik [3], for a monotonic vertex property function $\phi(v, C)$, the algorithm that repeatedly removes the vertex with smallest ϕ value gives the core decomposition. Since η-degree of a vertex is a monotonic vertex property function [9], then our peeling algorithm, which removes the vertex with smallest η-degree at each iteration of the algorithm, computes the desired core decomposition. Also, it should be noted that, the algorithm scans the next vertex in array $\mathbf{A}$ only when the η-degree of the previous vertex has been set. In addition, the lower-bounds never surpass the value of η-degrees. Thus, we conclude that the PA algorithm computes the desired core decomposition in probabilistic graphs.

Running time of the PA algorithm. The compute_swap function is dominated by two parts: **(1)** computing η-degrees takes $O(\eta\text{-deg}(u))$ for each vertex u **(2)** swapping a vertex to the proper block; each swap is done in constant time, and the maximum number of swaps required for a vertex u is $O(\eta\text{-deg}(u))$. However, as reported above, initially the difference between the lower-bounds obtained by Lyapunov CLT and the η-degrees is no more than one. Hence, either one or no swap is required initially. Thus, the compute_swap function takes in total

$$O\left(\sum_{v \in V} \sum_{u:(u,v) \in \mathcal{N}_v} \eta\text{-deg}(u)\right) = O\left(\sum_{v \in V} d_v \delta'\right) = O(m\delta'),$$

where m is the number of edges, and δ' is the maximum η-degree over all vertices at the time of their removal. The swap_left function swaps each vertex one block to the left in $\mathbf{A}$ in constant time. Therefore, the main cycle (Algorithm 1, line 6-19) takes $O(m\delta')$. In conclusion, the running time of the PA algorithm is $O(m\delta')$.

2.5 Sequential Algorithm (SA)

In this section we present a sequential algorithm (SA) which processes the vertices one-by-one, and as such, does not require the graph to be fully loaded into memory but rather one vertex at a time. Furthermore, as shown in Section 2.6, after only a fraction of iterations, SA is able to produce high quality results.

SA maintains bookkeeping information for each vertex and has a memory footprint of $O(n)$ as opposed to $O(m)$ for PA. More specifically, SA adopts the "semi-external" model of computation, which assumes that for each vertex we can fit a small constant amount of information in main memory while the edges of the graph are stored on disk. As other works have shown, this model is practical for a large number of real-world, web-scale graphs, and widely adopted to handle other graph problems [49, 79, 91, 92].

Algorithms that are sequential and semi-external do exist for deterministic core-decomposition (see [40, 54, 79]). They are based on the idea of maintaining an upper-bound on each vertex's coreness. This upper-bound is initialized to the degree of each vertex, and after each iteration of the algorithm it is tightened further using a simple *locality property* until it reaches the exact core value. The locality property is as follows. The coreness of a vertex v can at most be the largest value k such that v has at least k neighbors with a value greater or equal to k. Locality-based tightening (LBT) lowers the bound of a vertex to be the number k described above.

Unfortunately, this idea does not work for probabilistic core decomposition. LBT does not necessarily converge to true core values of vertices. For an example, consider Figure 2.1, $\eta = 0.5$, and Table 2.4. We initialize the upper-bounds to the η-degree of each vertex. Then, we execute a round of LBT. The bound for vertex 3 is tightened to 2. This is because the largest k for which vertex 3 has at least k neighbors with a value at least k is 2. These neighbors are 1, 2, 4, and 5 with a bound equal to 2. Running one more iteration of LBT does not produce any further tightening. However, the true core number of vertex 3 is 1 not 2 (see Table 2.4, last column). In the following, we tackle the problem with a new procedure we call *probabilistic bound tightening* (PBT). PBT takes into consideration the edge probability values and uses an optimized version of dynamic programming to gradually tighten the upper bounds of vertices to the true coreness. While PBT by itself can be used to compute coreness, we combine PBT with LBT in order to speed up the convergence, since LBT is faster than PBT. First we show that the obtained values at the end of LBT are always an upper-bound to the true coreness values.

v	η-degree	LBT-Round 1	PBT-Round 1	Coreness
0	1	1	1	1
1	2	2	1	1
2	2	2	2	2
3	3	2	1	1
4	2	2	2	2
5	2	2	2	2
6	1	1	1	1

Table 2.4: Upper-bounds obtained at LBT and PBT phase of SA. $\eta = 0.5$ for the example. LBT and PBT correspond to locality-based and probability bound tightening, respectively.

Proposition 1. *Let $\mathcal{G} = (V, E, p)$ be a probabilistic graph. Also, for each vertex $v \in V$, let k_v be the true coreness of v, and $\bar{k}_v$ be the value assigned to v at the end of the LBT phase. Then, $\bar{k}_v \geq k_v$.*

Proof. Once a LBT phase terminates, and the coreness of v is fixed to $\bar{k}_v$; then there should be a subset $V_{\bar{k}_v}$ of neighbors of v with the size at least $\bar{k}_v$, and $\forall u \in V_{\bar{k}_v}$: $\bar{k}(u) \geq \bar{k}_v$, where $\bar{k}(u)$ is the coreness assigned to u by LBT. However, considering the existence probability of edges incident to v, the value for $\Pr[d_{\mathcal{G}(V_{\bar{k}_v})}(v) \geq \bar{k}_v]$ can be less than η, where $\mathcal{G}(V_{\bar{k}_v})$ is the induced subgraph by $V_{\bar{k}_v}$. On the other hand, since $\Pr[d(v) \geq k]$ is monotonically non-increasing with the value of k, the true coreness of v should be in the interval $[0, \bar{k}_v]$ such that $\Pr[d_{\mathcal{G}(V_{\bar{k}_v})}(v) \geq k] \geq \eta > \Pr[d_{\mathcal{G}(V_{\bar{k}_v})}(v) \geq \bar{k}_v]$. Thus, $\bar{k}_v \geq k_v$. □

Once a LBT phase terminates, PBT starts to check whether the obtained value for each vertex is the true coreness of the vertex or not. If not, the gap is tightened further. We run LBT and PBT repeatedly, one after the other, until the core value for each vertex reaches a fixed point.

Before formally giving the algorithm, we present an example to illustrate how the SA algorithm works. We consider the probabilistic graph in Figure 2.1 and $\eta = 0.5$. LBT for this example was discussed earlier and the vertex values at the end of it are given in Table 2.4, third column. As can be seen, there are some vertices whose bounds are different from their true coreness (e.g. vertices 1 and 3). Therefore, the SA algorithm starts PBT to close this gap by checking for exactness of each bound to the true coreness of the vertex. For instance, consider vertex 1. We should check whether the coreness of it can be equal to 2 or not. In probabilistic core decomposition, a vertex v has coreness k if $\Pr[d_{\mathcal{H}}(v) \geq k] \geq \eta$, where $\mathcal{H}$ is a maximal subgraph in

which each vertex has degree of at least k. Checking $\Pr[d_{\mathcal{H}}(1) \geq 2]$, we see that this probability does not pass threshold $\eta = 0.5$, and therefore vertex 1 cannot have coreness of 2 in subgraph $\mathcal{H}$ which contains the neighbors 2 and 3. We perform the check of $\Pr[d_{\mathcal{H}}(v) \geq k]$ using an optimized version of dynamic programming explained later. Vertex 0 cannot belong to the subgraph because its bound is less than 2. For vertex 1, the maximum value, for which the probability passes the threshold, is equal to 1. As a result, the bound on the coreness of vertex 1 is updated to 1. The same process is done for vertex 3. The values obtained at the end of PBT are shown in the fourth column of Table 2.4, which contains exactly the true core value of each vertex.

SA algorithm description. The main cycle of SA is given in Algorithm 5. We define two Boolean variables, *PBT_phase* and *etadegree_change*. Variable *PBT_phase* is used to determine whether a PBT phase has been started or not. The variable *etadegree_change* is used to record whether there is some vertex (in PBT phase) with its η-degree changed or not. Initially both of them are set to *false*. Then, we invoke LBT. Once this process terminates (line 4), the vertex value (upper-bound) of all the vertices should be checked for the possibility of gap between the current vertex value and its true coreness. In fact, for each vertex v it should be investigated whether the degree of v can be greater than or equal to its current vertex value with probability no less than η or not. If so, the current vertex value should be the same as its η-deg(v). Checking whether a vertex should be processed (in PBT) or not is done using the Boolean array *check* which contains a flag for each vertex. If the flag is set to *true*, its vertex value needs to be checked, using the *PBT* function (see line 10). Then, the algorithm enters a PBT phase (lines 8-10). We make a clone, *checkNow*, of *check*; then we reinitialize *check* to the all-false vector (lines 8-9). The clone is needed to be used in the *for* loop in the *PBT* function (line 3, Algorithm 6). If after a PBT phase terminates there is some vertex with its η-degree not equal to its current vertex value, a new LBT phase is started again (line 15) and this process continues until a fixed point is obtained for all the vertices (lines 11-12).

PBT is given in Algorithm 6. The implementation iterates over each vertex v, and checks if there is a gap between v's value and its true coreness; if so, that vertex should be processed. In this process, only the neighbors of v whose current values are greater or equal to v's value are considered because it is only them that can contribute to v's coreness. We compute $\Pr[d(v) \geq C[v]]$ and check whether it passes threshold η or not. If $\Pr[d(v) \geq C[v]] < \eta$, then, since we know that $\Pr[d(v) \geq k]$ is non-increasing with k, the maximum k for which the probability passes the threshold

Algorithm 5 SA k-core computation function

```
1: function SA_CORE_COMPUTATION(Graph 𝒢)
2:     PBT_phase ← false
3:     C ← 0                                          ▷ array of core values
4:     LBT()
5:     check ← True                                   ▷ all-true vector
6:     etadegree_change ← false
7:     while true do
8:         checkNow ← check.clone()
9:         check ← False                              ▷ all-false vector
10:        PBT(𝒢, checkNow)
11:        if etadegree_change = false then
12:            break
13:        else
14:            PBT_phase = true
15:            LBT()
16:            etadegree_change ← false
17:    return cores
```

is returned and stored in variable *localValue* (line 5). In this case, all the values from $(C[v]-1)$ down to *localValue* should be checked as candidate values for the coreness of v and the maximum value (variable *max* in line 16), is chosen as the new value of v. This way we make sure that the algorithm does not go below the real coreness value at each PBT step (lines 8-15). For each value j in the *for* loop (line 8) we check whether $\Pr[d(v) \geq j] \geq \eta$ (line 9). If so, *max* is set to j (line 10) and the vertex value is updated to *max* (line 16). Otherwise, the value for which the probability passes the threshold is stored in variable *value* (line 13), and if it is greater than *max*, the latter is updated to the former (lines 14-15).

It should be noted that in the *for* loop, similar to what we do in lines 4-5 of Algorithm 6, we should compute $\Pr[d(v) \geq j]$ and check if it passes the threshold or not. For this, we use an optimized dynamic programming process as follows. Variable j starts from $C[v]-1$ and goes down to *localValue*. In order to avoid computing these probabilities from scratch, once we compute $\Pr[d_{\mathcal{H}}(v) \geq C[v]]$, we cache the following probabilities $\Pr[d_{\mathcal{H}}(v) = 0], \cdots, \Pr[d_{\mathcal{H}}(v) = C[v]]$, where $\mathcal{H}$ contains all the neighbors of v whose upper-bound is at least $C[v]$. Then, since for $j = C[v]-1$, $\mathcal{H}'$ contains all the neighbors whose upper-bound is exactly equal to j (we denote this set by V_j) and higher, we can have $\mathcal{H}' = V_j \cup \mathcal{H}$. Therefore, to compute $\Pr[d_{\mathcal{H}'}(v) \geq j]$ we use the computed probabilities in $\mathcal{H}$ and only consider vertices in V_j. This way we

Algorithm 6 SA probabilistic bound tightening function

```
1: function PBT(G, checkNow)
2:     for all v ∈ V do
3:         if checkNow[v] = true then
4:             if Pr[d_H(v) ≥ C[v]] < η then
5:                 localValue ← compute η-deg_H(v)
6:                 max ← localValue
7:                 m ← C[v] − 1
8:                 for all j ← m down to localValue do
9:                     if Pr[d_H'(v) ≥ j] ≥ η then
10:                        max ← j
11:                        break
12:                    else
13:                        value ← compute η-deg_H'(v)
14:                        if value ≥ max then
15:                            max ← value
16:                C[v] ← max
17:                etadegree_change ← true
18:                for all u : (u, v) ∈ N_v do
19:                    if C[u] ≥ C[v] then
20:                        check[u] ← true
```

optimize DP to compute the probabilities very fast since V_j is typically small (e.g., about 100 in our later evaluated large-scale graphs). For the next iteration, we store all the probabilities computed in the previous iteration and use them to compute new probabilities. In the following we describe our DP process in more detail. Let assume that we have computed $\Pr[d_{\mathcal{H}}(v) = k]$, where $k = 0, 1, ..., j$, and $\mathcal{H}$ is a subgraph of the input probabilistic graph in which each vertex has core value at least j. Also, let V_{j-1} contain all the vertices (including the neighbours of v) whose core value is exactly $j-1$. Thus, $\mathcal{H}' = \mathcal{H} \cup V_{j-1}$ is the subgraph whose vertices have core value at least $j-1$. We denote $\{e_1, ..., e_x\}$ to be the set of edges incident to v such that for each $e_i = (u_i, v)$, $u_i \in V_{j-1}$, where $i = 1, ..., x$. In order to evaluate $\Pr[d_{\mathcal{H}'}(v) = k']$, where $0 \leq k' \leq k$, and avoid from scratch computation, we denote the degree of v in the subgraph $\mathcal{H}'$ by $d(v \mid (\mathcal{H} \cup \{e_1, ..., e_x\}))$. Then, it holds that:

$$\begin{aligned} &\Pr[d(v \mid (\mathcal{H} \cup \{e_1, ..., e_x\})) = k'] = \\ &= p_{e_x} \Pr[d(v \mid (\mathcal{H} \cup \{e_1, ..., e_{x-1}\})) = k' - 1] + \\ &+ (1 - p_{e_x}) \Pr[d(v \mid (\mathcal{H} \cup \{e_1, ..., e_{x-1}\})) = k']. \end{aligned} \tag{2.18}$$

Letting $T(x,k') = \Pr[d(v \mid (\mathcal{H} \cup \{e_1,...,e_x\})) = k']$, we have the following recursive formula:

$$T(x,k') = p_{e_x}T(x-1,k'-1) + (1-p_{e_x})T(x-1,k') \tag{2.19}$$

with the following base cases:

$$\begin{cases} T(0,k') = \Pr[d_{\mathcal{H}}(v) = k'], \quad 0 \le k' \le k \\ T(x,-1) = 0, \end{cases} \tag{2.20}$$

As can be seen, in the base case of the recursive formula, we are using the previously computed probabilities to compute the probabilistic degree of vertex v in the new subgraph, which results in saving time significantly.

Since the vertex value of v is updated, all its neighbors with value at least the vertex value of v should be checked for validity of their vertex value. We show in the following that the PBT estimate of the coreness eventually equals the true coreness for each vertex.

Correctness of the SA algorithm. We prove this by contradiction. Suppose that after the last PBT phase, there is vertex u_1 such that $k(u_1) = k_1$ (real coreness) and $core[u_1] = k' > k_1$, where *core* is the coreness assigned to u_1 after PBT phase. Since $k(u_1) = k_1$, k_1 is the maximum value such that $\Pr[d_{\mathcal{H}}(u_1) \ge k_1] \ge \eta$, and $\Pr[d_{\mathcal{H}'}(u_1) \ge i] < \eta$ for all $i > k_1$, where $\mathcal{H}$ and $\mathcal{H}'$ are the maximal induced subgraphs in which each vertex has degree at least k_1 and i, respectively, with probability no less than η. Based on the properties of cores of a graph, we know that $\mathcal{H}' \subseteq \mathcal{H}$. If all the neighbors of u_1 in $\mathcal{H}$ have coreness k_1, then u_1 will not have any neighbors with coreness greater than k_1 in $\mathcal{H}'$, so u_1 eventually sets $core[u_1]$ equal to k_1, which is a contradiction. The similar argument holds if all the neighbors of u_1 in $\mathcal{H}$ have coreness greater than k_1. Since $k(u_1) = k_1$ and $core[u_1] = k' > k_1$, there should exist a neighbor u_2 with $k(u_2) = k_1$ and $core[u_2] > k_1$ such that $\Pr[deg_{\mathcal{H}' \cup \{u_2\}}(u_1) \ge k'] \ge \eta$, because otherwise $\Pr[d_{\mathcal{H}'}(u_1) \ge k'] < \eta$.

In fact, the existence of this neighbor will contribute to the assigned coreness of u_1 to be greater than k_1.

Now by reasoning similar to [54], we can build a sequence of vertices $S = \{u_i, u_{i+1}, u_{i+2}, ..., u_j = u_i\}$ connected to each other with $k(u_i) = k_1$ and $core[u_i] > k_1$. For each vertex u_i in S, let V_i be the set of all neighbors of u_i in $\mathcal{H}'$. Now, we can define a set $U = S \cup \bigcup_{u_i \in S} V_i$. The corresponding induced subgraph $\mathcal{G}(U)$ is a k'-core,

because all the vertices in V_i have coreness at least k' with probability greater than or equal to η. Also, since for each vertex u_i in S, $\Pr[d_{V_i \cup \{u_{i+1}\}}(u_i) \geq k'] \geq \eta$, we have that $\mathcal{G}(U)$ is a k'-core where $k' > k_1$. Hence, we find a subgraph whose vertices have coreness $k' > k_1$ which is a contradiction because we assumed that each vertex in S has coreness k_1. Therefore, k_1 is not a maximal (i.e. true) coreness.

Running time of the SA algorithm. The time complexity of the algorithm is dominated by time complexity of the PBT because LBT has a time complexity of $O(N - K + 1)$ [54], where K is the number of vertices with minimal degree (in probabilistic graphs it refers to η-degree). To analyze the time complexity of a PBT step (Algorithm 6), let Δ be the maximum upper-bound on the η-degree over all the vertices in all the probabilistic bound tightening steps. Lines 4 and 5 (η-degree computation) are done simultaneously, and take time $O(d_v C[v])$ for each vertex v, where $C[v]$ is the upper-bound on the η-degree of v. In practice this computation is fast because the η-degree computation is performed on the subgraph $\mathcal{H}$ which contains only those neighbors of v whose upper-bound is at least $C[v]$ (not all the d_v vertices). In the worst case the inner loop is repeated $C[v]$ times, and each time the η-degree computation and the checking of the probability threshold (lines 9 and 13) are performed similarly to what explained above. Therefore, the time complexity of this part is: $\sum_{j=1}^{C[v]} O(jd_v)$. It should be noted that at each iteration in the for loop, we use the previously computed probabilities to avoid computing η-degrees from scratch. However, here we consider the worst case analysis of the algorithm. The time complexity of lines 18-20 is $O(d_v)$. As a result, each PBT iteration takes $\sum_{v \in V} \Big(O(C[v]d_v) + \sum_{j=1}^{C[v]} O(jd_v) + O(d_v) \Big) = \sum_{v \in V} \Big(O(\Delta d_v) + O(\theta d_v) + O(d_v) \Big)$, where $\theta = \Delta^2$. Therefore, the time complexity of each PBT round is $O(m\theta)$, where m is the total number of edges. In the worst case, we assume that at each PBT round, the difference between the actual coreness of a vertex and its initial estimate (the initial η-degree) decreases by one unit. Thus, in the worst case $\Gamma = \sum_{v \in V} \eta\text{-deg}(v)$ PBT steps are required. Therefore, the total running time can be expressed as: $O(\Lambda m)$, where $\Lambda = \Gamma\theta$.

As in [54], the complexity upper bounds for such iterative algorithms are not representative of practical performance. In practice, LBT and PBT are fast, and the number of iterations is only a handful, thus SA is an efficient algorithm for large datasets while requiring only $O(n)$ memory footprint.

2.6 Experiments

In this section, we present our experimental results. Our implementations are in Java and the experiments are conducted on a commodity machine with Intel i7, 2.2Ghz CPU, and 12Gb RAM, running Ubuntu 14.03. The hard disk is Seagate Barracuda ST31000524AS 2TB 7200 RPM.

The statistics for all of the datasets we consider are shown in Table 2.5. We obtained flicker, dblp, and biomine from the authors of [9], and the rest of the datasets from Laboratory of Web Algorithmics.[1] The datasets are divided by horizontal lines according to their size, small (S), medium (M), large (L), and extra large (XL). We use the Webgraph framework [8] to store these datasets. The flickr, dblp, and biomine datasets already contained probability values. For the other datasets we generated probability values uniformly distributed in $[0, 1]$.

Name	$\|V\|$	$\|E\|$
flickr	24,125	300,836
dblp	684,911	2,284,991
cnr-2000	325,557	2,738,969
biomine	1,008,201	6,722,503
ljournal-2008	5,363,260	49,514,271
arabic-2005	22,744,080	553,903,073
uk-2005	39,459,925	783,027,125
twitter-2010	41,652,230	1,202,513,046

Table 2.5: Dataset Statistics

We evaluate our algorithms in three important aspects. First is numerical stability. As [9] points out, using probability values may lead to numerical instability. We discuss this in Subsection 2.6.1. Second is the quality of our η-degree lower-bounds using Lyapunov CLT. We evaluate and discuss this in Subsection 2.6.2. Third is the efficiency of our proposed algorithms. We show our performance results in Subsection 2.6.3.

2.6.1 Numerical Stability

For evaluating numerical stability we refer to the results of η-degree computations shown in Table 2.6. The table contains results for $\eta = 0$, $\eta = 10^{-9}$ and $d_v = 100$,

[1] `http://law.di.unimi.it/datasets.php`

	DP	DP128	DP256	DPU	DPlog2	CLT
NE	100%	99%	0%	0%	0%	0%
AE	8%	5%	0%	0%	0%	0%
AT	0.09	11.44	12.57	27.71	1.56	0.05
NE	100%	100%	100%	0%	0%	0%
AE	40%	35%	28%	0%	0%	0%
AT	3.3	1238	1499	26355	222	0.1
NE	0%	0%	0%	0%	0%	43%
AE	0%	0%	0%	0%	0%	1%
AT	0.19	14.41	15.34	43.86	2.14	0.27
NE	0%	0%	0%	0%	0%	0%
AE	0%	0%	0%	0%	0%	0%
AT	3.7	1244	1382	23934	171	0.3

Table 2.6: Error statistics and average running time for different precision levels for $\eta = 0$ and $\eta = 10^{-9}$. NE stands for Number of Errors, AE for Average Relative Error, and AT for Average Time (ms). Specifically, AE=$\|error\|/true\_value$.
1st part*: $\eta = 0$, $d_v = 100$;* **2nd part***: $\eta = 0$, $d_v = 1000$;* **3rd part***: $\eta = 10^{-9}$, $d_v = 100$;* **4th part***: $\eta = 10^{-9}$, $d_v = 1000$*
DP*: DP using plain numbers in Java;* **DP128, DP256, DPU***: DP using BigDecimal in Java and setting the precision to 128 bits, 256 bits, and unlimited, respectively;* **DPlog2***: DP doing computations in log space.*

$d_v = 1000$. For each combination of η and d_v we compute η-degrees for 1000 randomly selected vertices. We show results for twitter-2010, but any of the other datasets above can be used for this experiment to obtain similar results.

Using the BigDecimal class in Java we adjust the numerical precision to different levels when using DP for computing η-degrees. We compare numerical results of DP using different precision levels in Java, where DPU (DP with unlimited precision) is the highest level of precision and thus the gold standard. The DP128 and DP256 columns in the table correspond to computing η-degrees using DP by setting precision to 128 bit and 256 bit, respectively. We observe that the more we increase the level of precision, the longer it takes to perform the computation. For example, when executing the DP algorithm for computing η-degrees for vertices with $d_v = 1000$, it takes more than 7000 times longer to perform the computation using unlimited precision than using the default (plain) number computation in Java.

Following [9], we start by setting $\eta = 0$ (top two parts). For this η the (k, η)-core decomposition of a probabilistic graph should coincide with the core decomposition of the deterministic graph derived by ignoring probabilities. The accuracy can thus be

measured by comparing, for each vertex, the $(k, 0)$-core number with the core number obtained on the deterministic version of the graph.

We can see that for $\eta = 0$, we need a lot of precision (bits) to achieve error free computation. For example, when $d_v = 100$, we need at least 256-bit precision, whereas when the degree is 1000, we need DP with unlimited precision.

The DPLog2 column of the table shows the results if we operate in logarithmic space[2]. We can see that DP operating in logarithmic space is very stable and never produces any error at all, while being much faster than the DP variants operating with specified precision.

We test $\eta = 0$ to compare our results to previous work [9]. However, the situation changes significantly if η is greater than zero even by a very small amount. If we set $\eta = 10^{-9}$ (one billionth) or higher, we never get an error even for plain DP which is much faster than any other DP variant including the one in log space (see the two bottom parts). We tested with η in $[10^{-9}, 0.5]$ and obtained similar results.

The last column of Table 2.6 shows the accuracy of η-degree computation when using Lyapunov CLT instead of DP. For $\eta = 0$, computing η-degrees using CLT is error-free. More interesting are the bottom two parts for η greater than zero. For vertices with $d_v = 100$, CLT makes errors in computation (on 43% of the vertices), however, those errors are small; only 1% on average of the η-degree value.

When considering vertices with $d_v = 1000$, the computation using CLT is error-free and furthermore it is orders of magnitude faster than the variants of DP. Again, we also tested with a variety of η levels and obtained similar results. Guided by the above, in our algorithms, we set the threshold on d_v to start using Lyapunov CLT for computing η-degrees at 1500. Recall that the η-degree computation is needed in PA when a vertex becomes candidate for removal, while in SA it is needed when initializing vertex values.

In summary, regarding numerical stability, our contribution is to show that DP is sensitive to the setting of η value and becomes error-free once η is greater than zero even by a small amount. This was not investigated thoroughly before. On the other hand, CLT is resilient to different η values (even $\eta = 0$). With respect to efficiency in computing η-degrees, when $d_v \geq 1000$, CLT can be used to produce error-free computations orders of magnitude faster than all the DP variants.

[2] `https://en.wikipedia.org/wiki/Logarithm`

Dataset	Max Error	
	CLT	Beta Function
biomine	1	1,663
cnr-2000	1	9,056
ljournal-2008	1	9,453

Table 2.7: Maximum error of CLT and regularized beta function for selected datasets.

2.6.2 Accuracy of CLT as a lower-bound

Here we investigate the quality of the lower-bounds on η-degrees obtained using CLT. Namely, we compare CLT with another method for deriving lower-bounds used in [9], which utilizes a formula based on the regularized beta function.[3] We computed initial η-degrees for biomine, cnr-2000, and ljournal-2008 with $\eta = 0.1$ using DP, Lyapunov CLT, and regularized beta function. Then, we computed the maximum error for CLT and regularized beta function. We report the results in Table 2.7. As can be seen, the maximum error for CLT for all the datasets is one which means that the difference between the values obtained by CLT and the true values is either zero or at most one. On the other hand the max error for the regularized beta function is big, in the order of thousands.

The small value of error for CLT is of great importance in the main cycle of the PA algorithm where each vertex is processed based on its lower-bound before its true η-degree is computed. To summarize, since the difference between each vertex's lower-bound (computed by CLT) and its true η-degree is small, we only need to do a small number of swaps to place a vertex in the proper place in the array $\mathbf{A}$ resulting in significant savings in running time.

2.6.3 Efficiency of the Proposed Algorithms

Table 2.8 shows the running times of the PA (left) and SA (right) algorithms on the selected datasets. For twitter-2010 we report the results for different values of η ranging from 0.1 to 0.5. For the other datasets, we only show results for $\eta = 0.1$ and omit results for $\eta = 0.2, \ldots, 0.5$, since they are similar to those for $\eta = 0.1$ and their nature is the similar to what we see for twitter-2010.

For the small and medium datasets, PA produced the results in about 2 sec for

[3] `http://mathworld.wolfram.com/RegularizedBetaFunction.html`

Dataset	η	PA-Running Time	SA-Running Time
flickr	0.1	2.05	1.88
dblp	0.1	5.81	9.57
cnr-2000	0.1	3.99	8.49
biomine	0.1	40	40
ljournal-2008	0.1	120	342
arabic-2005	0.1	2,539	2,513
uk-2005	0.1	1,709	1,961
twitter-2010	0.1	8,096	21,662
	0.2	8,547	20,268
	0.3	8,358	19,670
	0.4	8,364	20,544
	0.5	8,929	20,111

Table 2.8: Running times (sec) of PA and SA. Numbers less than 10 have been rounded to 2 decimal places, and those above 10 have been rounded to the nearest integer.

the small and 4 to 6 sec for the medium datasets. PA also performed well on the large datasets; biomine and ljournal-2008, computing the core decomposition of these two graphs on average in only 40 sec and 2 min, respectively. Notably, for biomine for instance, our algorithm is 32 times faster than the algorithm of [9] (see also Table 2.10 which we discuss later).

The running time of PA is good on the very large datasets as well. On uk-2005 and arabic-2005, PA completed in about 28 min and 42 min, respectively; less than one hour; in contrast the state-of-art [9] was not able to complete for these datasets in our machine after one day. For twitter-2010, which is much larger than uk-2005 and arabic-2005, with a maximum η-degree of $1,500,282$, our PA algorithm completed in around two hours, which is impressive for processing such a big dataset on our consumer-grade machine.

The running times of SA are shown in Table 2.8 (right part). For the small and medium datasets the total running times of SA were similar to those of PA. For the large datasets, SA needed more time, 2-3 times more, than PA. However, we recall that SA has a much smaller memory footprint than PA, namely $O(n)$ as opposed to $O(m)$ for PA. As such we are trading time for space when using SA over PA, a beneficial feature to have when using low-cost, cloud-server machines.

Another benefit of using SA is that it can produce good approximate results after only a small fraction of total iterations. We discuss this more later in this section.

Dataset	η-degmax	kmax	kavg	η
flickr	78	46	3.70439	0.1
dblp	162	26	1.99825	0.1
cnr-2000	9,255	38	5.77113	0.1
biomine	6,269	79	3.08697	0.1
ljournal-2008	9,664	156	5.4735	0.1
arabic-2005	287,949	1,088	15.2903	0.1
uk-2005	888,658	274	13.0279	0.1
twitter-2010	1,500,282	986	15.7391	0.1
	1,499,970	976	15.0873	0.2
	1,499,744	970	14.6407	0.3
	1,499,552	964	14.2919	0.4
	1,499,372	959	13.9927	0.5

Table 2.9: Maximum η-degree, maximum probabilistic coreness, average probabilistic coreness, value of the threshold η.

Effect of η values. We observe that the total running time does not change much as the value of η increases. This is because when η increases, the η-degree of a vertex might not change or slightly decrease and as such this does not have a significant effect on the total running time of the algorithms. This is also evident in Table 2.9 where the average coreness (kavg) and the maximum coreness (kmax) decrease only slightly as η increases. As before, we report the average coreness, the maximum coreness, and the maximum η-degree for twitter-2010 for $\eta = 0.1, \ldots, 0.5$, whereas for the other datasets we only show the values for $\eta = 0.1$.

Comparing with the algorithm of [9]. We also compared the running times of our algorithms, PA and SA, to that of the algorithm of [9] (which we call BGKV[4]), for $\eta = 0.1$. We considered the (k, η)-core implementation made available to us by the authors of [9] with numbers represented by using BigDecimal in Java. From this, we also created a second version, which we call BGKV-2, where we replaced the computations using BigDecimal with plain computations in Java. This is because as shown in Subsection 2.6.1, for η greater than 0, the use of BigDecimal for DP is not warranted.

Table 2.10 summarizes the running times on three datasets, flickr, dblp and biomine. For biomine, which is a large dataset, we were only able to use up to 64 precision bits. We can see that both PA and SA are significantly faster than

[4]Abbreviation using the first letters of the authors' names.

Algorithm		flickr	dblp	biomine
BGKV	pr=64	18	90	2,493
	pr=128	27	133	N.P.
	pr=256	35	148	N.P.
BGKV-2		0.49	4.26	85
PA		2.05	5.81	40
SA		1.88	9.57	40

Table 2.10: Running time (sec) of the algorithm in [9] with BigDecimal (BGKV), and without (BGKV-2) versus PA and SA. "pr" is the precision (bits) used. BGKV cannot run for biomine to completion after one day (we use N.P. for "Not Possible"). BGKV-2 is faster for the small datasets, flickr and dblp, but twice slower for biomine than PA and SA. For the rest of the datasets that are bigger than biomine, both BGKV and BGKV-2 cannot run to completion in our machine after one day. Our algorithms PA and SA can produce results for every dataset, see Table 3.2.

BGKV. For example, for biomine our algorithms are more than 32 times faster than BGKV.

BGKV-2 is faster than BGKV. For the small datasets, flickr and dblp, it is even faster than PA and SA. This can be attributed to the fact that since the memory footprint is small, the benefit of using our algorithms is outweighed by the overhead of using the Webgraph compression structures in PA and SA.

The situation changes significantly as the size of the datasets grows. For biomine, BGKV-2 is about twice slower than PA and SA. For the rest of the datasets that are bigger than biomine, BGKV-2 cannot run to completion in our machine after one day. For those datasets, our algorithms, PA and SA, are the only algorithms that can produce results in a matter of minutes or few hours (for twitter-2010), Table 2.8.

Convergence speed of SA. To further investigate the execution of SA as it unfolds with time, we look at average error and maximum difference (max error) from the true core value over the sequence of iterations (see Figure 2.2). As shown by the plots in the top part of the figure, the average error sharply decreases for all the datasets which we consider, except for uk-2005 and arabic-2005 whose average errors decrease more gradually.

Similar to the average error, the maximum difference from the true core value (shown in the plots in the bottom part) drops quickly, becoming 1 in only a fraction of the total number of iterations. Furthermore, as the value of η increases, the total number of iterations required decreases.

These results show that SA produces approximate results of good quality in only

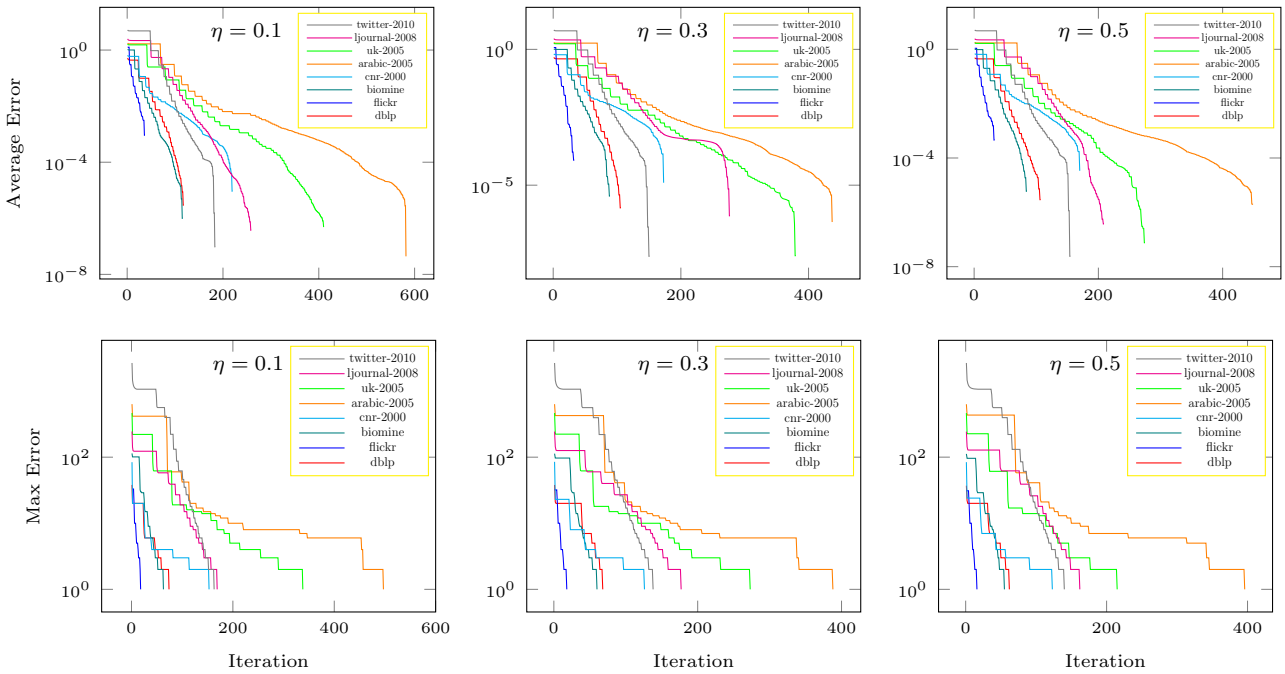


Figure 2.2: Average error (average of difference from true core coreness) and max error (maximum difference from true coreness) versus iterations for different values of η.

a fraction of iterations needed for completion. For instance, for arabic-2005 with $\eta = 0.1$, the average error drops below 0.01 at iteration 200, only one third of the total number of required iterations (about 600, see the end of the curve). Depending on the application domain this can be a desirable property during data analysis.

2.7 Conclusions

In this chapter, we presented two efficient algorithms, PA and SA, for computing the core decomposition of probabilistic graphs at web scale. An important contribution of this work is the use of Lyapunov Central Limit Theorem in these algorithms to compute tail probabilities for η-degrees. We evaluated our algorithms, and showed that they are efficient and numerically stable. Our algorithms were considerably faster than the-state-of-the-art for large datasets. For datasets larger than biomine, our algorithms PA and SA, were the only algorithms able to run to completion on a consumer grade machine. In particular, PA was able to compute probabilistic core decomposition for uk-2005, arabic-2005, and twitter-2010 in 28 min, 42 min and 2.2 hours, respectively, which is impressive for such large datasets. SA has smaller

memory footprint and can produce approximate results of high quality in only a fraction of iterations needed for full completion.

Chapter 3

Fast Truss Decomposition in Large-scale Probabilistic Graphs

In this chapter we introduce our peeling algorithm for computing truss decomposition in probabilistic graphs. We start with problem formulation and necessary background for truss decomposition. We show efficiency of our proposed algorithm through extensive experimental results.

3.1 Related Work

In the literature, much research has been done in the area of mining and querying probabilistic graphs [15, 39, 57, 86, 87, 98, 36], such as the k-nearest neighbor search over probabilistic graphs [61], uncertain graph sparsification [58], and mining top k maximal cliques in probabilistic graphs [99].

Recently k-truss has attracted a lot of attention due to its cohesive structure and the fact that it can be used to compute other definitions of dense subgraphs, such as k-clique. In deterministic graphs, truss decomposition has been studied extensively in different settings (cf. [78, 33, 85, 14]). For instance, Xing et al. in [85], propose a novel traversal in-memory algorithm for truss decomposition in deterministic graphs. In external-memory setting, Wang and Cheng in [78] propose an efficient algorithm for deterministic graphs that don't fit in memory. Huang et al. [33] study truss decomposition of dynamic deterministic graphs.

For probabilistic graphs, the notion of (k, η)-truss is introduced by Huang, Lu, and Lakshmanan in [34]. Their algorithm for computing (k, η)-truss is based on iterative

edge peeling and uses dynamic programming for computing support probability of edges. While this algorithm runs in polynomial time, it does not scale well to large graphs, especially those having a high maximum vertex degree.

[34] also proposes the notion of global (k, η)-truss based on the probability of each edge belonging to a connected k-truss in a possible world. An algorithm based on sampling is proposed in [34] to find global (k, η)-trusses. This notion of probabilistic truss decomposition falls in the category of #P-hard problems.

3.2 Background

Trusses in deterministic graphs. Let $G = (V, E)$ be an undirected graph with no self-loops. For a vertex $u \in V$, the set of its neighbors is denoted by $N_G(u)$ and defined as $N_G(u) = \{v : (u, v) \in E\}$. A *triangle* in G is defined as a set of three vertices $\{u, v, w\} \subseteq V$ such that all three edges (u, v), (v, w) and (u, w) exist. This triangle is denoted by Δ_{uvw}. The *support* of an edge $e = (u, v)$ in G, denoted by $\mathsf{sup}_G(e)$, is defined as the number of triangles in graph G containing e. Formally, $\mathsf{sup}_G(e) = |N_G(u) \cap N_G(v)|$.

The k-truss of G is defined as the largest subgraph F of G in which each edge e has $\mathsf{sup}_F(e) \geq k$. The set of all k-trusses forms the truss decomposition of G, where $0 \leq k \leq k_{\max}$, and $k_{\max}$ is the largest support of any edge in G.

Trusses in Probabilistic graphs. Let $\mathcal{G} = (V, E, p)$ be a probabilistic graph, which is defined over a set of vertices V, a set of edges E and a probability function $p : E \to (0, 1]$ which maps every edge $e \in E$ to an existence probability $p(e)$. Again we assume that edges exist independent of each other. Also, let $G = (V, E_G) \sqsubseteq \mathcal{G}$ be a possible world of $\mathcal{G}$.

Given edge $e = (u, v)$, let $k_e = |N_{\mathcal{G}}(u) \cap N_{\mathcal{G}}(v)|$. We define an integer random variable $\mathsf{sup}_{\mathcal{G}}(e)$ with values in $[0, k_e]$ and distribution:

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) = t] = \sum_{G \sqsubseteq \mathcal{G}} \Pr[G] \cdot \mathbb{1}(\mathsf{sup}_G(e) = t), \tag{3.1}$$

where $\mathbb{1}(\mathsf{sup}_G(e) = t)$ is an indicator function which takes on 1 if edge e has support equal to t in G, and 0 otherwise.

Given a user-specified threshold $\eta \in (0, 1]$, the *probabilistic support* of an edge e, denoted by η-$\mathsf{sup}_{\mathcal{G}}(e)$, is the maximum integer $t \in [0, k_e]$ for which $\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t] \geq \eta$.

It should be noted that as t increases (decreases), $\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t]$ decreases (increases).

Definition 2. *Let η be a user defined threshold.*

- (k, η)-truss *of $\mathcal{G}$ is the largest subgraph $\mathcal{F}$ of $\mathcal{G}$ in which each edge e has* probabilistic support *in $\mathcal{F}$ no less than k, i.e. η-$\mathsf{sup}_{\mathcal{F}}(e) \geq k$.*
- Truss decomposition *of $\mathcal{G}$ is the set of all (k, η)-trusses, for $k \in [0, k_{\max,\eta}]$, where $k_{\max,\eta} = \max_e\{\eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)\}$.*
- Truss value *of an edge e, $\kappa_\eta(e)$, is the largest integer k for which e belongs to a (k, η)-truss.*

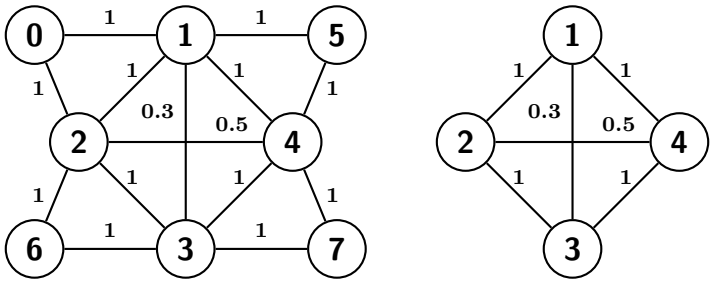
Proposition 2. *(k, η)-truss of $\mathcal{G}$ is the subgraph of $\mathcal{G}$ containing all and only the edges e in $\mathcal{G}$ with $\kappa_\eta(e) \geq k$.*

In this dissertation (as in [34, 25]) we focus on finding the truss values of the edges in an input graph. Then (k, η)-truss for any k is constructed by collecting all edges e with $\kappa_\eta(e) \geq k$.

Example 1. *Consider Figure 3.1a, edge $e = (1, 2)$, and $\eta = 0.20$. We have $Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq 3] = 1 \cdot 0.3 \cdot 0.5 = 0.15$ (product of probabilities that $\triangle_{012}$, $\triangle_{123}$, $\triangle_{124}$ exist), and $Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq 2] = 0.65$. Since 0.65 is greater than η, η-$\mathsf{sup}_{\mathcal{G}}(e) = 2$.*

Figure 3.1b shows a $(2, 0.15)$-truss $\mathcal{F}$ of $\mathcal{G}$. Each edge $e \in \mathcal{F}$, is contained in 2 triangles with probability 0.15.

Consider $e = (1, 2)$ and $\eta = 0.15$. Now, η-$\mathsf{sup}_{\mathcal{G}}(e) = 3$. Edge e is in $(1, 0.15)$-truss ($\mathcal{G}$ itself) and $(2, 0.15)$-truss ($\mathcal{F}$). There is no $(3, 0.15)$-truss, thus, $\kappa_\eta(e) = 2$.



(a) Probabilistic graph $\mathcal{G}$ (b) (2,0.15)-truss $\mathcal{F}$ of $\mathcal{G}$

Figure 3.1: A running example for probabilistic truss decomposition.

Obtaining η-$\mathsf{sup}_{\mathcal{G}}(e)$ using Dynamic Programming. To obtain η-$\mathsf{sup}_{\mathcal{G}}(e)$, we need to compute $\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t]$, which can be written in the form of the following recursive formula:

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t] = \Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t-1] - \Pr[\mathsf{sup}_{\mathcal{G}}(e) = t-1],$$

Computing $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$ for different values of i can be done using dynamic programming as proposed in [34].

3.3 Peeling Algorithm framework

We describe a CLT-focused algorithm to compute truss decomposition in probabilistic graphs.

3.3.1 Computing η-Supports using Central Limit Theorem

We start with defining the problem in appropriate way so that Central Limit Theorem can be applied to provide close approximation of $\Pr\big[\mathsf{sup}_{\mathcal{G}}(e) \geq k\big]$. Next, we show the conditions on which CLT can produce close approximation to tail probabilities of the support edge.

As described in Chapter 2, based on CLT, the distribution of properly scaled sum of a sequence of random variables converges to normal distribution under specific conditions. We repeat the formulation of Lyapunov CLT for the ease of the reader.

Theorem 3. ***Lyapunov CLT.*** *Let $\zeta_1, \zeta_2, \cdots, \zeta_n$ be a sequence of independent, but non-identically distributed random variables, each with finite expected value μ_k and variance σ_k. Let*

$$s_n^2 = \sum_{k=1}^{n} \sigma_k^2, \tag{3.2}$$

Lyapunov CLT states that if

$$\lim_{n\to\infty} \frac{1}{s_n^{2+\delta}} \sum_{k=1}^{n} \mathrm{E}[|\zeta_k - \mu_k|^{2+\delta}] = 0, \tag{3.3}$$

for some $\delta > 0$, then $\frac{1}{s_n}\sum_{k=1}^{n}(\zeta_k - \mu_k)$ converges in distribution to a standard normal random variable.

Equation (3.3) is called Lyapunov's condition which in practice is usually tested for the special case $\delta = 1$. The proof for this theorem can be found in [18].

Computing η-supports using Lyapunov CLT. Given an edge e, to compute $\Pr\big[\mathsf{sup}_{\mathcal{G}}(e) \geq k\big]$ we assume that e exists. Let $\Pr\big[\mathsf{sup}_{\mathcal{G}}(e) \geq k | e \text{ exists}\big]$ be the probability that e is contained in at least k triangles with the assumption that e exists. Thus, the true edge support probability is obtained by:

$$\Pr\big[\mathsf{sup}_{\mathcal{G}}(e) \geq k\big] = \Pr\big[\mathsf{sup}_{\mathcal{G}}(e) \geq k | e \text{ exists}\big] \cdot p(e), \tag{3.4}$$

Each edge e_j in probabilistic graph has existence probability of $p(e_j)$, which is independent of the other edge probabilities. As a result, associated with each edge e_j we can define a Bernoulli random variable ζ_{e_j} which takes on 1 with probability $p(e_j)$ and 0 with probability $(1 - p(e_j))$. Since each edge is assumed to exist independently of other edges, ζ_{e_j}'s are independent. Given an edge $e = (u, v)$, let $\mathcal{T}_e$ be a set of all the common neighbors of u and v in $\mathcal{G}$. We have,

$$\mathcal{T}_e = N(u) \cap N(v) = \{t_1, \cdots, t_{k_e}\}.$$

For each common neighbor t_i, let ζ_{u,t_i} and ζ_{v,t_i} be the corresponding Bernoulli random variables to the edges (u, t_i) and (v, t_i), respectively. Let $X_i = \zeta_{u,t_i} \cdot \zeta_{v,t_i}$. The following observations hold for each random variable X_i: **(1)** X_i's are independent, since ζ_{u,t_i} and ζ_{v,t_i} are independent random variables. **(2)** X_i's are Bernoulli random variables which take on 1 with probability $p(u, t_i) \cdot p(v, t_i)$. This is because X_i can be equal to 1 when both $\zeta_{u,t_i} = 1$ and $\zeta_{v,t_i} = 1$, with probability $p(u, t_i) \cdot p(v, t_i)$. Otherwise, if at least one of them is 0, the value of X_i will become zero. The probability that at least one of these random variables become zero is $1 - (p(u, t_i) \cdot p(v, t_i))$.

Now, let us consider the triangle $\triangle_{uvt_i}$. It should be noted that only common neighbours can create a triangle containing edge e. With the assumption that e exists, the triangle $\triangle_{uvt_i}$ exists if both edges (u, t_i) and (v, t_i) exist, which is associated with $X_i = 1$. On the other hand, the triangle does not exist if at least one of edges, (u, t_i) and (v, t_i), does not exist, which corresponds to $X_i = 0$. Therefore, corresponding to each triangle $\triangle_{uvt_i}$, we can define the Bernoulli random variable X_i.

Let $p_i = p(u, t_i) \cdot p(v, t_i)$. Since X_i is a Bernoulli random variable, we know that

$\mathrm{E}[X_i] = \mu_i = p_i$ and $\mathrm{Var}[X_i] = p_i(1-p_i)$. Since $\mathsf{sup}_{\mathcal{G}}(e) = \sum_{i=1}^{k_e} X_i$, we have:

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq k | e \text{ exists}] = \Pr\left[\sum_{i=1}^{k_e} X_i \geq k\right]. \tag{3.5}$$

Bernoulli random variables X_i's are independent, but may not be identically distributed. Thus, if condition (3.3) is satisfied and if k_e is large enough, we can conclude that $\frac{1}{s_{k_e}} \sum_{i=1}^{k_e} (X_i - \mu_i)$ has standard normal distribution, where $s_{k_e} = \sqrt{\sum_{i=1}^{k_e} p_i(1-p_i)}$. In order to compute the right-hand side of equation (3.5), we can subtract $\sum_{i=1}^{k_e} \mu_i$ from both sides of the inequality, and then divide by s_{k_e} which results in:

$$\Pr\Big[\sum_{i=1}^{k_e} X_i \geq k\Big] = \Pr\Big[\frac{1}{s_{k_e}} \sum_{i=1}^{k_e} (X_i - \mu_i) \geq \frac{1}{s_{k_e}} (k - \sum_{i=1}^{k_e} \mu_i)\Big]. \tag{3.6}$$

Using Lyapunov CLT and setting

$$Z = \frac{1}{s_{k_e}} \sum_{i=1}^{k_e} (X_i - \mu_i), \tag{3.7}$$

we can conclude that Z has standard normal distribution. Thus

$$\Pr\big[\mathsf{sup}(e) \geq k | e \text{ exists}\big] \cong \Pr\big[Z \geq z\big], \tag{3.8}$$

where $z = \frac{1}{s_{k_e}}(k - \sum_{i=1}^{k_e} \mu_i)$. Using the complementary cumulative distribution function [100] of standard normal variable Z, we can simply evaluate the right-hand side of Equation (3.8) for each value of k. Thus, to find the η-support for an edge, we start with $k = 1$, approximate $\Pr[\mathsf{sup}(e) \geq k] = \Pr[\mathsf{sup}(e) \geq k | e \text{ e exists}] \cdot p(e)$ using Lyapunov CLT (for the first factor), and find the maximum k for which the probability multiplied by $p(e)$ is above threshold η. For an edge e, the obtained value of k, which can be in range from one to k_e, is set as initial η-support for that edge. Given an edge $e = (u, v)$, time complexity of finding η-support is $O(k_e)$, where $k_e = |N(u) \cap N(v)|$. Recall that DP required $O(k_e^2)$ for this step.

In the following we show that Lyapunov's condition in Theorem 3 is satisfied for our problem. We set $\delta = 1$ in Equation (3.3) to show that this condition holds for a sequence of non-identically distributed Bernoulli random variables.

Theorem 4. *Given a sequence of random variables $X_i \sim Bernoulli(p_i)$, where $1 \leq i \leq n$, the Lyapunov's condition (3.3) for $\delta = 1$ is satisfied whenever $s_n^2 = \sum_{k=1}^{n} p_k(1-p_k) \to \infty$.*

Accuracy of the Approximation. Using Berry–Esseen theorem [101], in the following corollary we show how to obtain an upper-bound on the maximal error while approximating the true distribution of the sum of X_i's with the normal distribution.

Corollary 2. *For each edge e in the probabilistic graph $\mathcal{G}$ with X_i's being Bernoulli random variables defined as above in this section, where $i = 1, \ldots, k_e$, the error bound on the approximation of the right-hand side of Equation (3.8) to the standard normal distribution is given as follows:*

$$\sup_{x \in \mathbb{R}} |F_{k_e}(x) - \Phi(x)| \leq \frac{0.56}{\sqrt{p_1(1-p_1) + \cdots + p_{k_e}(1-p_{k_e})}}.$$

3.3.2 Peeling Algorithm (PA)

In [82], a peeling algorithm was proposed to calculate the k-truss in deterministic graphs. While the algorithm is not applicable to probabilistic graphs, its optimized array-based data structures for storing edge information of the graph are useful. Our new peeling algorithm, termed as *CLT_based-PA* algorithm, is built on the same array-based data structures but utilizes central limit theorem to compute and update support probabilities of edges which participate in more than 100 triangles.[1]

The *CLT_based-PA* algorithm consists of two main parts: **(1)** initial probabilistic support computation, and **(2)** probabilistic truss computation which involves updating probabilistic support values once an edge is removed.

In initial support computation step, the η-support of each edge e is computed using CLT and Equation (3.8), if k_e is greater than 100. Otherwise, DP can be used safely. The details on DP approach can be found in [34]. The initial phase can be executed in parallel, since probabilistic support of each edge can be computed independently of other edges.

After initialization, the *CLT_based-PA* algorithm runs in three steps: First, sort edges in ascending order of their η-support in the array *sortedEdge*, and store their positions in the array.

[1]This value was chosen because it was large enough to keep the approximation error obtained from Corollary 2 small.

Then, remove edges with the lowest η-support. The removal of an edge $e = (u, v)$ affects the η-support of all edges that can constitute triangles with (u, v). As a result, the algorithm finds all the common neighbors w of u and v, i.e., $\triangle_{uvw}$ is a triangle containing edge (u, v).

At the third step, the η-support of (u, w) and (v, w) is updated if their η-supports are greater than e's η-support. In the updating part, if the number of remaining triangles which contain edges (u, w) and (v, w) is greater than 100, we perform update phase using CLT approach. Otherwise, we apply DP. Since the η-support has been changed, we change the position of edges (v, w) and (u, w) in *sortedEdge* array in constant time [82]. The algorithm continues until all the edges in the graph are removed. Then, the truss value of each edge is obtained by adding 2 to the final η-support.

3.4 Experiments

Our implementations are in Java and the experiments are conducted on a commodity machine with Intel i7, 2.2Ghz CPU, and 12Gb RAM, running Ubuntu 14.03. The hard disk is Seagate Barracuda ST31000524AS 2TB 7200 RPM.

The statistics for the datasets are shown in Table 3.1. Datasets with real edge probabilities are flicker, dblp, and biomine which are obtained from the authors of [9], and the rest of the datasets from Laboratory of Web Algorithmics.[2] For these datasets we generated probability values uniformly distributed in $[0, 1]$. The datasets are categorized according to their size, small (S), medium (M), and large (L), and separated using horizontal line in the table. We use the Webgraph framework [8] to store these datasets.

Name	$\|V\|$	$\|E\|$
flickr	24,125	300,836
dblp	684,911	2,284,991
cnr-2000	325,557	2,738,969
biomine	1,008,201	6,722,503
ljournal-2008	5,363,260	49,514,271

Table 3.1: Dataset Statistics

Table 3.2 represents the running times of our proposed approach, versus the running times of the state-of-the-art, which uses dynamic programming (DP) only and is

[2] `http://law.di.unimi.it/datasets.php`

Dataset	η	Running Time DP_Pure	Running Time CLT_based–PA	gain (%)
flickr	0.1	351	94	73%
dblp	0.1	37	34	8.50%
biomine	0.1	7642	2554	67%
cnr-2000	0.1	N.P.	7874	100%+
ljournal-2008	0.1	54627	26129	52%
	0.2	50614	27064	47%
	0.3	45052	24799	45%
	0.4	36563	21332	42%
	0.5	28773	16291	43

Table 3.2: Running times (sec) of DP_Pure and CLT_based–PA. The column "gain (%)" reports the gain of CLT_with_DP algorithm over DP_Pure algorithm. We use N.P. for "Not Possible".

referred as *DP_Pure*. The last column shows the gain of *CLT_based–PA* algorithm over *DP_Pure*. For ljournal-2008, we present the results for different values of η ranging from 0.1 to 0.5. However, for the other datasets, we only show the results for $\eta = 0.1$, and omit results for $\eta = 0.2, \ldots, 0.5$, since they are similar to those for $\eta = 0.1$ and their performance trend is similar to what we see for ljournal-2008. As can be seen, *CLT_based–PA* algorithm is significantly faster than *DP_Pure*. For instance, for biomine, which is a large dataset, the gain of our algorithm is 67 percent, making *CLT_based–PA* truss decomposition algorithm three times faster than *DP_Pure*.

CLT_based–PA produced the results in about 1.5 minutes and about 34 seconds for flickr and dblp, respectively. Although flickr is smaller than dblp in terms of the number of vertices and edges, its probabilistic maximum truss is much greater at value 47 compared to 14 in dblp. We represent the maximum probabilistic truss and maximum probabilistic support in Table 3.3. *These values are the same as those obtained by DP_Pure.* On biomine which is a large dataset, our proposed algorithm completed in about 43 minutes; which is quite impressive. In contrast, *DP_Pure* produced the results in more than 2 hours. The running time for ljournal-2008 increases, which is quite reasonable, because this graph has 49 million edges with probabilistic support of 1030 when $\eta = 0.1$. The same argument holds for cnr-2000 which has probabilistic support of 4672 which is significantly big. *DP_Pure wasn't able to run to completion in our machine after one day.*

Dataset	η-suppmax	kmax	η
flickr	49	47	0.1
dblp	42	14	0.1
biomine	151	33	0.1
cnr-2000	4672	13	0.1
ljournal-2008	1030	51	0.1
	1015	43	0.2
	1001	35	0.3
	980	27	0.4
	530	19	0.5

Table 3.3: Maximum η-support, maximum probabilistic truss value, value of the threshold η.

Effect of η values. The running time of both algorithms increases as η becomes small. This is because as η decreases, the chance for support probabilities to pass the threshold increases, resulting in larger values of η-supports. This is particularly important in performance evaluation of *DP_Pure* algorithm– as η decreases the DP algorithm approaches its worst case time complexity, $O(k_e^2)$, for an edge e. As a result, for larger values of η, the running time of *DP_Pure* improves, but is still by far slower than *CLT_based–PA*. In terms of the effect of η on truss decomposition and support values, as can be seen in Table 3.3, the maximum truss and maximum initial probabilistic support decrease as η increases. As before, we report the maximum truss, and the maximum η-support for ljournal-2008 for $\eta = 0.1, \ldots, 0.5$, whereas for the other datasets we only show the values for $\eta = 0.1$.

3.5 Conclusions

We presented a peeling algorithm for computing truss decomposition in probabilistic graphs at web scale. Our peeling algorithm uses Lyapunov's central limit theorem to obtain the probabilistic support for an edge. Unlike the dynamic programming approach, the computation does not rely on incremental evaluation of support probabilities. In addition, it can efficiently update probabilistic support when a triangle is removed from the input graph, without the need of storing all the previously computed support probabilities. We evaluated our algorithm and showed that it is significantly faster than state-of-the-art for large datasets. For large and medium datasets our

algorithm obtained approximately 50 percent gain over the proposed algorithm in the literature, completing the biomine dataset in less than one hour.

Chapter 4

Truss Decomposition on Large Probabilistic Networks using H-Index

In this chapter we present our second approach which is an exact and scalable algorithm for truss decomposition of probabilistic graphs. The algorithm is based on progressive tightening of the estimate of the truss value of each edge based on h-index computation and novel use of dynamic programming. In Chapter 3, we reviewed literature on truss decomposition in both deterministic and probabilistic context. We also defined problem formulation and necessary background. Therefore, in this chapter we start with a brief reminder of notations introduced in the previous chapter and we do not repeat in-detail the explanation of these notations in this chapter. Please refer to Table 4.1 for the main notations we will use in this chapter.

4.1 Algorithm Framework

Here we propose an algorithm based on h-index updating, which has been introduced in the context of *deterministic* graphs by [65]. Given a set of real numbers, the h-index of the set is defined as the largest number h such that there are at least h elements in the set that are equal to h or higher. For instance, the h-index of the set $\{1, 2, 3, 3, 5\}$ is 3 because the set includes three numbers no less than 3.

We also have the notion of the h-index of an edge which is an integer variable initialized to the edge's initial support (as a first approximation of the edge's truss

Symbol	Description
$\mathcal{G} = (V, E, p)$	probabilistic graph
$G \sqsubseteq \mathcal{G}$	possible world G of probabilistic graph $\mathcal{G}$
$e = (u, v)$	edge e with endpoint vertices u and v
$p(e)$	existence probability of edge e
$\triangle = (u, v, w), \triangle_{uvw}$	triangle with vertices u, v, and w
$N_{\mathcal{G}}(u)$	set of neighbor vertices to vertex u in $\mathcal{G}$
$N_G(u)$	set of neighbor vertices to vertex u in G
k_e	$\lvert N_{\mathcal{G}}(u) \cap N_{\mathcal{G}}(v)\rvert$, for a given edge $e = (u, v)$
$\mathsf{sup}_G(e)$	$\lvert N_G(u) \cap N_G(v)\rvert$, for a given edge $e = (u, v)$
$\mathsf{sup}_{\mathcal{G}}(e)$	integer random variable with range $[0, k_e]$
η	user-specified probability threshold
η-$\mathsf{sup}_{\mathcal{G}}(e)$	largest value of t s.t. $\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t] \geq \eta$ (probabilistic support of e in $\mathcal{G}$)
$k_{\max}$	$\max_e\{\mathsf{sup}_G(e)\}$
$k_{\max,\eta}$	$\max_e\{\eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)\}$
$\kappa_\eta(e)$	largest k s.t. e belongs to a (k, η)-truss (truss value of e in $\mathcal{G}$ for threshold η)

Table 4.1: Main Notations

value). Then the algorithm iterates multiple times over the edges tightening up their h-index as described below. In fact, truss values are related to h-indices. For instance, truss value of an edge can be defined as the largest k such that it is contained in at least k triangles (or with probability $\geq \eta$ in the probabilistic context) whose edges have truss value of at least k.

Let e be an edge and (e, e', e'') be a triangle supporting e. For such a triangle, we define its *support* to e as the minimum of h-indices of e' and e''. The support values of *all* triangles supporting e are collected in a set L and its h-index is computed. At each iteration, the h-index of e is updated to the smallest of its current value and the h-index of L.

In our algorithm, we refer to this process as *Phase I*. This phase corresponds to the h-index based algorithm of [65] for the deterministic case. In deterministic graphs, once the process terminates, the h-index of each edge becomes equal to the truss value of that edge. However, we show that this does not solve our problem.

Deterministic h-index updating, *Phase I*. In the following we provide explanation of *Phase I* of our algorithm, which is based on [65].

Definition 3. *Given a set K of natural numbers, $\mathcal{H}(K)$ is the largest $k \in \mathbb{N}$ such*

Algorithm 7 Phase I

```
1: function PHASE I(G, h, scheduled)
2:     update_Phase I ← true
3:     while update_Phase I do
4:         update_Phase I ← false
5:         for all edge e ∈ E do
6:             L ← empty set
7:             for all △ containing e do
8:                 e', e'' ← the two edges in △ other than e
9:                 ρ_△ ← min {h(e'), h(e'')}
10:                L.add(ρ_△)
11:            updated-h_e ← H(L)
12:            if updated-h_e < h(e) then
13:                update_Phase I ← true
14:                h(e) ← updated-h_e
15:                scheduled[e] ← true
```

that at least k elements of K are greater than or equal to k.

Phase I is given in Algorithm 7. Let $h(e)$ denotes the h-index value of edge e at each iteration of our algorithm. *Phase I* tightens $h(e)$ values for each edge e and iterates until no further updates occur for any $h(.)$ value irrespective of edge probabilities. The flag *update_Phase I* is used to check termination of *Phase I* (line 3). The flag is initially set to **true** (line 2), and stays **true** as long as there is an update on a $h(.)$ value (lines 4,12, and 13). For each triangle $\triangle = (e, e', e'')$ which contains e, the algorithm computes its $\rho_\triangle$ value that is the minimum value of $h(e')$ and $h(e'')$ and collects them in a set L (lines 7-10). Then, function $\mathcal{H}$ is applied on set L (line 11). If the h-index of set L is smaller than $h(e)$, it is assigned as a new index for edge e in array h (line 11). The validity of the assigned value is checked by *Phase II* in the next iterations of our proposed *proHIT* algorithm using the *scheduled* array (line 15).

We demonstrate how *Phase I* works in the following example:

Example 2. To illustrate how h-index works on deterministic graphs, we refer to Figure 4.1. The figure shows a deterministic graph with 6 vertices. Initially, the triangle counts of all the edges are computed and are set as initial values on the h-index of the edges. Let h_0 be the list of these initial values, which are shown with blue color in the figure. Then, the algorithm starts updating the h-indices based on the initial values. Let h_1 be the list of updated values at this step (red). Edge $e = (0, 2)$, for instance, participates in 4 triangles and *in each of them*, the

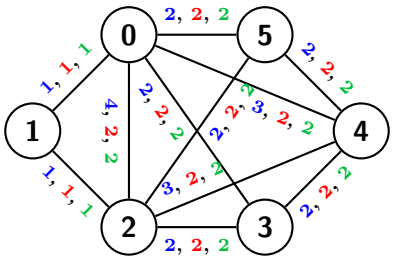


Figure 4.1: A running example of h-index algorithm on a deterministic graph.

algorithm finds the edge neighbor to $(0, 2)$ with minimum h_0 value and records this value in an array. Then the algorithm updates the h-index of e. So, $L = \{\min(h_0(0, 1), h_0(1, 2)), \min(h_0(0, 3), h_0(2, 3)), \min(h_0(0, 4),$
$h_0(2, 4)), \min(h_0(0, 5), h_0(2, 5))\} = \{1, 2, 3, 2\}$. As a result, $h_1(0, 2) = \mathcal{H}(L) = 2$. The h-index of edges $(0, 4)$ and $(2, 4)$ are updated similarly. No more updates happen in the next iteration. Since the given graph is a deterministic graph, at the end, each edge obtains its truss value (green).

Phase II. Since *Phase I* does not take into account an edge having enough support probability to be part of a (k, η)-truss, it may not converge to the true truss value of the edge.

For an example, consider Figure 4.2 and $\eta = 0.2$. In Table 4.2 we show the execution of our algorithm at each phase. The first column shows edges in the graph. The last shows their truss values. Column ph_I shows the h-index values at the end of *Phase I*. Initially h-index, $h(e)$, of each edge e is set to η-$\mathsf{sup}_{\mathcal{G}}(e)$ (second column).

edge e	h-index $h(e)$			
	η-$\mathsf{sup}_{\mathcal{G}}(e)$	ph_I	ph_II	truss value
$(i, j), 1 \leq i < j \leq 4$	2	2	1	1
$(0, 1), (0, 2), (1, 4), (1, 5)$	1	1	1	1
$(2, 6), (3, 6), (3, 7), (4, 7)$	1	1	1	1

Table 4.2: η-$\mathsf{sup}_{\mathcal{G}}(e)$, values obtained by Phase I (Ph_I) and Phase II (Ph_II), respectively, truss values. $\eta = 0.2$ for Figure 4.2.

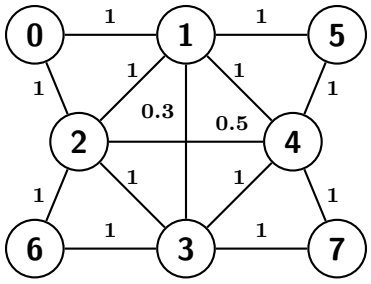


Figure 4.2: A running example.

Now, consider edge $e = (1, 2)$. For each triangle containing e, *Phase I* finds the minimum h-index of the two edges other than e in the triangle, and adds this minimum to a set L. For e, we have $L = \{\min(h(0,1), h(0,2)), \min(h(1,3), h(2,3)), \min(h(1,4), h(2,4))\} = \{1, 2, 2\}$. Since there are two numbers on this list that are equal to 2, *Phase I* sets 2 as the h-index of edge e. Further execution of *Phase I* cannot produce any updates. However, the truss value of edge e is in fact 1 (see last column of Table 4.2) as we explain later in this section. Therefore, *Phase I* is not able to converge to the truss value of e. Nevertheless, we prove that *Phase I* can be used to provide upper-bounds to true truss values. The proof of this fact, Theorem 5, is presented in Section 4.2.

We tackle the problem by introducing a process we call *Phase II*. Although *Phase I* is not able to compute exact truss values, it can provide good upper-bounds which can be used in *Phase II*. Therefore, we combine *Phase II* with *Phase I* to speed up convergence as *Phase I* runs faster than *Phase II*.

Algorithm 8 Probabilistic h-index Truss (proHIT)

1: **function** PROBABILISTIC h-INDEX UPDATING($\mathcal{G}$,*support*,η)
2: **for all** edge $e \in E$ **do**
3: $h(e) \leftarrow \eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)$, *scheduled*[e] $\leftarrow$ **true**
4: *Phase I* $(\mathcal{G}, h, \textit{scheduled})$ ▷ Deterministic h-index
5: *updated* $\leftarrow$ **false** ▷ True if any $h(e)$ is updated
6: **while** true **do**
7: *Phase II* $(\mathcal{G}, h, \textit{scheduled})$
8: **if** *updated* is **false then break**
9: **else**
10: *Phase I* $(\mathcal{G}, h, \textit{scheduled})$, *updated* $\leftarrow$ **false**
11: **for all** edge $e \in E$ **do** $\kappa_\eta(e) \leftarrow h(e)$
12: **return** array of $\kappa_\eta(\cdot)$

The major steps of our algorithm, *probabilistic h-index truss* (*proHIT*), are summarized in Algorithm 8. At a high level, we maintain an array h indexed by edges where we initially store the h-index value for each edge. Then, we tighten up these values using *Phase I* and *Phase II*, and by the end of the iterations, we have the output truss values in array h.

Checking whether an edge requires processing or not is done by array *scheduled*, which is initialized to **true** for each edge. Variable *updated* records whether there is some edge with its $h(.)$ value changed or not. Line 4 invokes *Phase I*. Then, *Phase II* starts and processes the $h(.)$ values (upper-bound on truss value) of all the edges for the possibility of gap between current value and truss value (line 7). If after *Phase II* terminates, there is some edge with its $h(.)$ value updated, *Phase I* (line 10) starts again. The process continues until each $h(.)$ value achieves convergence (lines 6-10). The final truss value of each edge is set to the final h-index.

Algorithm 9 Phase II

```
1: function PHASE II(G, h, scheduled)
2:     for all edge e ∈ E do
3:         if scheduled[e] is false then continue
4:         Γ ← ConstructGamma(e)
5:         h_e-changed ← false
6:         while Pr[sup(e) ≥ h(e) | Γ] < η and h(e) ≥ 0 do
7:             h_e-changed ← true
8:             h(e) ← h(e) − 1
9:             Γ ← ConstructGamma(e)
10:        if h_e-changed is true then
11:            updated ← true
12:            for all edge e' ∈ E_Γ \ {e} do
13:                scheduled[e'] ← true
14:        scheduled[e] ← false
15: function CONSTRUCTGAMMA(e)
16:    Γ ← empty set
17:    for all △ containing e do
18:        e', e'' ← the two edges in △ other than e
19:        ρ_△ ← min{h(e'), h(e'')}
20:        if ρ_△ ≥ h(e) then Γ.add(△)
21:    return Γ
```

Phase II of our approach is given in Algorithm 9. *This part crucially differentiates our approach from h-index based algorithms for deterministic graphs.*

Let e be an edge. We define Γ to be the set of (e, e', e'') triangles that contain e and $h(e'), h(e'') \geq h(e)$. It is only the triangles in Γ that can contribute to updating $h(e)$.

Also, we denote by $\Pr\big[\mathsf{sup}(e) \geq h(e) \;\big|\; \Gamma\big]$ the probability that e is contained in at least $h(e)$ triangles selected from Γ. Now, in order to possibly tighten up the current upper bound value of e, we check the condition

$$\Pr\big[\mathsf{sup}(e) \geq h(e) \;\big|\; \Gamma\big] \geq \eta. \tag{4.1}$$

For each scheduled edge e, line 4 constructs the set Γ using function *ConstructGamma*, and the above condition is checked in line (6). Checking the condition presents its own challenges and is presented in detail later in this section. If the condition fails, integer values less than $h(e)$ are checked one at a time until we find a value for $h(e)$ for which the condition is true. Set Γ is updated each time to correspond to the $h(e)$ value being used for edge e (lines 8-9). This guarantees that the assigned value does not go below the true truss value of each edge. Variable h_e*-changed* records whether a new $h(e)$-value for e is obtained or not, and initially is set to **false** (line 5).

For instance, let us consider again the example in Fig. 4.2. For $e = (1, 2)$ in Table 4.2 with $h(e) = 2$ (which is obtained by *Phase I*, see third column in Table 4.2), we verify the condition $\Pr[\mathsf{sup}(e) \geq 2 \;\big|\; \Gamma] \geq \eta$, where $\Gamma = \{\triangle_{123}, \triangle_{124}\}$ (set of triangles containing e with other edges having an $h(.)$ value of at least 2). We have $\Pr[\mathsf{sup}(e) \geq 2 \;\big|\; \Gamma] = 0.3 \cdot 0.5 = 0.15 < 0.2$. As a result, e cannot have a truss value of 2. As such, we update $h(e)$ to be 1 and check the condition again. We have $\Pr[\mathsf{sup}(e) \geq 1 \;\big|\; \Gamma] = 1 > \eta$, where $\Gamma = \{\triangle_{012}, \triangle_{123}, \triangle_{124}\}$ (set of triangles containing e with other edges having an $h(.)$ value of at least 1). Since the new probability is greater than η, $h(e)$ is settled to 1.

Let E_Γ be the edges of the triangles in Γ. If a new $h(e)$ value is obtained and checked in line 10, the edges in $E_\Gamma \setminus \{e\}$ may change their $h(.)$ values and thus are scheduled to be processed in the next iteration (lines 12-13).

In the following section we provide the proof of the correctness of the algorithm as well as time complexity analysis.

The main challenge in *Phase II* is efficient checking of condition 4.1 for different values of $h(e)$ until a proper value is obtained. For this, we introduce a modified dynamic programming (DP) process to avoid computation of these probabilities from scratch.

Modified DP. This process is invoked when we check the condition on line 6 of Algorithm 9.

Let $H = h(e)$, and Γ be the set of (e, e', e'') triangles as defined earlier. For a triangle $\triangle = (e, e', e'') \in \Gamma$, we denote by $\rho_\triangle$ the minimum value of $h(e'), h(e'')$. We have $\rho_\triangle \geq H$.

The probability $\Pr[\mathsf{sup}(e) \geq H \mid \Gamma]$ is computed using DP [34]. However, Γ changes in each iteration of the while loop (see line 9). We would like to avoid the computation of $\Pr[\mathsf{sup}(e) \geq H \mid \Gamma]$ from scratch each time. It should be noted that the probability computation is valid if e exists, with existence probability $p(e)$. So, based on statistics we can write:

$$\Pr[\mathsf{sup}(e) \geq H \mid \Gamma] = p(e) \cdot \Pr[\mathsf{sup}(e) \geq H \mid \Gamma, e \text{ exists}], \tag{4.2}$$

Initially, the following probabilities are computed

$$\Pr[\mathsf{sup}(e) = 0 \mid \Gamma, e \text{ exists}], \cdots, \Pr[\mathsf{sup}(e) = H \mid \Gamma, e \text{ exists}], \tag{4.3}$$

We now cache these probabilities.

Given a Γ set, let $\Pr_{(H,\Gamma)} = \Pr[\mathsf{sup}(e) \geq H \mid \Gamma]$. For $H - 1$, we define $\mathcal{T}^{(H-1)} = \{\triangle_1, \cdots, \triangle_j\}$ to be the set of $\triangle$ triangles which contain e, and $\rho_\triangle = H - 1$. Let Γ^{new} be the set of all triangles $\triangle$ which contain e, and have $\rho_\triangle \geq H - 1$. Clearly, $\Gamma^{new} = \Gamma \cup \mathcal{T}^{(H-1)}$. Now, we need to compute $\Pr_{(H-1,\Gamma^{new})}$ efficiently using the probabilities in Equation 4.3. For this, we only need to look at set $\mathcal{T}^{(H-1)}$, which is usually small (i.e., not more than 50 in our tested real graphs). As such, the computation is done very fast.

Given an edge $e = (u, v)$, let us assume that we have computed $\Pr[\mathsf{sup}(e) = k \mid \Gamma, e \text{ exists}]$, where $k = 0, \cdots, H$, and Γ is as before. We have:

$$\begin{aligned}
\Pr[\mathsf{sup}(e) = k \mid \Gamma^{new}, e \text{ exists}] \\
&= \Pr[\mathsf{sup}(e) = k \mid \Gamma \cup \mathcal{T}^{(H-1)}, e \text{ exists}] \\
&= \Pr[\mathsf{sup}(e) = k \mid \Gamma \cup \{\triangle_1, \cdots, \triangle_j\}, e \text{ exists}] = T(j, k).
\end{aligned}$$

By $T(j, k)$ we denote the probability that e participates in k triangles selected from $\Gamma \cup \{\triangle_1, \cdots, \triangle_j\}$, given that e exists.

Let $\triangle_l = (u, v, w_l)$, where $l \in [1, j]$, be a triangle in $\mathcal{T}^{(H-1)}$. With the assumption that e exists, we consider the following two exclusive events (in terms of possible worlds). Event 1: $\triangle_l$ exists and e participates in $k-1$ other triangles of $\mathcal{T}^{(H-1)}$. Event 2: $\triangle_l$ does not exist and e participates in k other triangles of $\mathcal{T}^{(H-1)}$. The sum of probabilities of events (1) and (2) gives us the probability that e participates in k triangles in $\mathcal{T}^{(H-1)}$. Formally,

$$\begin{aligned} T(j,k) = {} & p(u,w_l)p(v,w_l)T(j-1,k-1) \\ & + (1 - p(u,w_l)p(v,w_l))T(j-1,k). \end{aligned} \tag{4.4}$$

The base cases for the above formula are: **(1)** $T(0,k) = \Pr[\mathsf{sup}(e) = k \mid \Gamma, e \text{ exists}]$, $0 \leq k \leq H$, **(2)** $T(j,-1) = 0$.

As can be seen, in the recursive formula, we use the previously computed support probabilities to compute new probability values. This significantly speeds up the process. By multiplying $T(j,k)$ by $p(e)$ we obtain the desired probability $\Pr[\mathsf{sup}(e) = k \mid \Gamma^{new}]$.

Note. The central limit theorem can be used for approximating $\Pr[\mathsf{sup}(e) \geq H \mid \Gamma]$ as well as obtaining an estimate for initial probabilistic support of edges [25]. Approximation can make *proHIT* algorithm faster. However, in this work, we focus on proposing an exact algorithm for solving truss decomposition.

4.2 Proofs of Correctness

In this section, we present the proofs of correctness of our algorithm, *proHIT*, propsed in Section 4.1. In particular, we show that convergence can be obtained in a finite number of iterations. We start by showing that the values obtained by *Phase I* are upper-bounds on the truss values.

Theorem 5. *In every iteration, Phase I provides upper-bounds on truss values of edges in the input probabilistic graph.*

Proof. Given an edge e, let assume that the index value by *Phase I* is fixed at H. This means that H is the maximum value such that there exists at least H triangles (regardless of their existence probability), which contain e, and have $\rho_{\triangle} \geq H$ for each triangle $\triangle$.

Let Γ be the set of (e, e', e'') triangles that contain e and $h(e'), h(e'') \geq h(e)$.

Given the threshold η, the probability $\Pr[\mathsf{sup}(e) \geq H \mid \Gamma]$ might be either (1) less than η or (2) greater than or equal to η.

If the first case holds, the truss value of e should be in the interval $[0, H)$.

Now, let us consider the second case. Since H is the maximum value obtained by *Phase I*, e cannot be contained in $H' > H$ triangles, with ρ-value at least H' because otherwise, *Phase I* would have produced an estimate of H' for e. Thus, the probability that the truss value of e is equal to H' is 0. Furthermore, since $\Pr[\mathsf{sup}(e) \geq H \mid \Gamma] \geq \eta$, we can conclude that the truss value of e should be in the interval $[0, H]$, i.e. the truss value of e can be H but also can be lowered in future iterations.

Therefore, considering the first and second cases, we can conclude that the true truss value of e should be in the interval $[0, H]$. As a result, the theorem follows.

□

In the following, we first generalize some definitions and properties of deterministic truss decomposition to the probabilistic context.

Let $\mathcal{G}$ be a probabilistic graph, and η be a user-defined threshold. Given an edge e, recall that by $\kappa_\eta(e)$ we denote the largest integer k for which e belongs to a (k, η)-truss. Also, the probabilistic support of e, η-$\mathsf{sup}_\mathcal{G}(e)$, is the maximum integer $t \in [0, k_e]$ for which $\Pr[\mathsf{sup}_\mathcal{G}(e) \geq t] \geq \eta$, where $k_e = |N_\mathcal{G}(u) \cap N_\mathcal{G}(v)|$, and $N_\mathcal{G}(u)$ and $N_\mathcal{G}(v)$ are the set of neighbor vertices to u and v, respectively. Let $\delta_\eta(\mathcal{G})$ be the minimum probabilistic support in $\mathcal{G}$; i.e. $\delta_\eta(\mathcal{G}) = \min_e\{\eta\text{-}\mathsf{sup}_\mathcal{G}(e)\}$. Thus, we have:

$$\Pr[\mathsf{sup}_\mathcal{G}(e) \geq \delta_\eta(\mathcal{G})] \geq \eta, \quad \forall e \in E(\mathcal{G}), \tag{4.5}$$

We use $E(\mathcal{G})$ to denote the set of edges in graph $\mathcal{G}$. Moreover, let W be the set of *all* the triangles in $\mathcal{G}$ which contain e. We note that the computation of $\Pr[\mathsf{sup}_\mathcal{G}(e) \geq k]$ is done by considering the triangles which contain e. Thus, the values obtained by $\Pr[\mathsf{sup}_\mathcal{G}(e) \geq k]$ and $\Pr[\mathsf{sup}(e) \geq k \mid W]$ are basically the same, and as a result we use $\Pr[\mathsf{sup}_\mathcal{G}(e) \geq k]$ interchangeably with $\Pr[\mathsf{sup}(e) \geq k \mid W]$ to refer to same concept. We have the following proposition.

Proposition 3. *Given a subgraph $\mathcal{G}' \subseteq \mathcal{G}$ and an edge $e = (u, v)$ in $\mathcal{G}'$, let W and W' be the sets of all the triangles in $\mathcal{G}$ and $\mathcal{G}'$, respectively, which contain e. We have that $Pr[\mathsf{sup}(e) \geq k \mid W'] \leq Pr[\mathsf{sup}(e) \geq k \mid W]$, where $k = 0, \cdots, k_e$. (As mentioned earlier, this is equivalent to $Pr[\mathsf{sup}_{\mathcal{G}'}(e) \geq k] \leq Pr[\mathsf{sup}_\mathcal{G}(e) \geq k]$) [34].*

The following Lemma is a generalization of a property of truss values in deterministic graphs [65] to the probabilistic context.

Lemma 1. *Given threshold η, for all $e \in E(\mathcal{G})$, we have*

$$\kappa_\eta(e) = \max_{\mathcal{G}' \subseteq \mathcal{G}} \delta_\eta(\mathcal{G}'), \tag{4.6}$$

where $\mathcal{G}'$ is a subgarph of $\mathcal{G}$ which contains e (i.e. $e \in E(\mathcal{G}')$).

Proof. Let $\mathcal{F}$ be the $(\kappa_\eta(e), \eta)$-truss which contains e. By the definition of truss subgraph we have: $\delta_\eta(F) = \kappa_\eta(e)$. Thus, $\kappa_\eta(e) \leq \max_{\mathcal{G}'} \delta_\eta(\mathcal{G}')$, for any $\mathcal{G}'$ which contains e.

Now, we show that $\kappa(e) \geq \max_{\mathcal{G}'} \delta_\eta(\mathcal{G}')$. We use proof by contradiction. Let $\mathcal{G}''$ be the largest subgraph of $\mathcal{G}$ that contains e and has $\delta_\eta(\mathcal{G}'') > \kappa_\eta(e)$. Based on Equation 4.5 we have $\Pr[\mathsf{sup}_{\mathcal{G}''}(e') \geq \delta_\eta(\mathcal{G}'')] \geq \eta$, for any edge $e' \in E(\mathcal{G}'')$, including e. Hence, $\mathcal{G}''$ is a $(\delta_\eta(\mathcal{G}''), \eta)$-truss and contains e. This is a contradiction by the definition of $\kappa_\eta(e)$ which is the largest value of k such that e is contained in a (k, η)-truss. □

Following [65], we define the concept of degree (support) levels of edges in a probabilistic graph. First, we start with some technical definitions. Let $\mathcal{G}$ be a probabilistic graph, and η be a user-defined threshold. Also, let $\mathcal{C}(\mathcal{G})$ be the set of edges and their containing triangles. We define the following features for edges and triangles in $\mathcal{C}(\mathcal{G})$:

- Triangle $\triangle \in \mathcal{C}(\mathcal{G})$, if $\forall e \in \triangle$, $e \in \mathcal{C}(\mathcal{G})$.
- If e is removed from $\mathcal{C}(\mathcal{G})$, all $\triangle \supset e$ are also removed from $\mathcal{C}(\mathcal{G})$.

Remark. We could have created two separate sets for edges and triangles, but doing so would significantly complicate the notation and its use in the proof as maintaining the relationship between these two sets would be cumbersome. This definition of $C(G)$ is chosen purely for notational convenience and is similar to the definition used in [65] where it is defined as the set of all r-cliques and s-cliques.

Definition 4. ***Degree Levels.*** *We define degree levels in a recursive way in a probabilistic graph $\mathcal{G}$. Let set L_i denote the i-th degree level. L_0 is defined as the*

set of edges e which have minimum probabilistic support in $\mathcal{C}(\mathcal{G})$. L_1 is defined as the set of edges which have minimum probabilistic support in $\mathcal{C}(\mathcal{G}) \setminus L_0$, and so on. In general, L_i contains the set of edges which have minimum probabilistic support in $\mathcal{C}(\mathcal{G}) \setminus \bigcup_{j<i} L_j$. The maximum value of i for which L_i can be non-empty is equal to $k_{\max,\eta}$. We recall that $k_{\max,\eta} = \max_e\{\eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)\}$.

Theorem 6. *Given integers i and j such that $i \leq j$ and a threshold η, for any $e_i \in L_i$ and $e_j \in L_j$, $\kappa_\eta(e_i) \leq \kappa_\eta(e_j)$.*

Proof. Let $L' = \bigcup_{r\geq i} L_r$ be the union of all levels i and above. Also, let $\mathcal{G}'$ be the graph such that $L' = E(\mathcal{G}')$. Based on definition of levels, for $e_i \in L_i$, we have $\eta\text{-}\mathsf{sup}_{\mathcal{G}'}(e_i) = \delta_\eta(\mathcal{G}')$. Moreover, $e_j \in L_j$ implies $\eta\text{-}\mathsf{sup}_{\mathcal{G}'}(e_j) \geq \eta\text{-}\mathsf{sup}_{\mathcal{G}'}(e_i)$. Since the truss value of e_i is $\kappa_\eta(e_i)$, there should exist a $(\kappa_\eta(e_i), \eta)$-truss $\mathcal{F}$ which contains e_i. We can have two following cases:
(1) $E(\mathcal{F}) \subseteq L'$. Using Proposition 3 and the fact that each edge in $\mathcal{F}$ is in $\mathcal{G}'$ (because $L' = E(\mathcal{G}')$), we have $\Pr[\mathsf{sup}_{\mathcal{F}}(e) \geq k] \leq \Pr[\mathsf{sup}_{\mathcal{G}'}(e) \geq k]$. For edge e_i, $\kappa_\eta(e_i) = \delta_\eta(\mathcal{F})$. Thus, setting $k = \delta_\eta(\mathcal{F})$, we have $\eta \leq \Pr[\mathsf{sup}_{\mathcal{F}}(e_i) \geq \delta_\eta(\mathcal{F})] \leq \Pr[\mathsf{sup}_{\mathcal{G}'}(e_i) \geq \delta_\eta(\mathcal{F})]$. Since $\eta\text{-}\mathsf{sup}_{\mathcal{G}'}(e_i)$ is the maximum value of k such that $\Pr[\mathsf{sup}_{\mathcal{G}'}(e_i) \geq k] \geq \eta$, we have $\eta\text{-}\mathsf{sup}_{\mathcal{G}'}(e_i) \geq \delta_\eta(\mathcal{F})$. Thus, we obtain that $\eta\text{-}\mathsf{sup}_{\mathcal{G}'}(e_i) = \delta_\eta(\mathcal{G}') \geq \delta_\eta(\mathcal{F}) = \kappa_\eta(e_i)$. On the other-hand, based on Lemma 1, for $\mathcal{G}' \subseteq \mathcal{G}$ which contains e_j, $\delta_\eta(\mathcal{G}') \leq \kappa_\eta(e_j)$. Combining the above, $\kappa_\eta(e_i) \leq \kappa_\eta(e_j)$.
(2) $E(\mathcal{F}) \setminus L' \neq \emptyset$. This means that there should exist at least one edge in $E(\mathcal{F})$, but not in L' (e.g. in the levels $< i$). Let e' be one of these edges such that $e' \in E(\mathcal{F}) \cap L_b$ with the minimum value of b, where $b < i$. Since $e' \in \mathcal{F}$ and $\mathcal{F}$ is a $(\kappa_\eta(e_i), \eta)$-truss, then $\eta\text{-}\mathsf{sup}_{\mathcal{F}}(e') \geq \kappa_\eta(e_i)$. Set $M = \bigcup_{r\geq b} L_r$. It should be noted that $E(\mathcal{F}) \subseteq M$. Let $\mathcal{Q}$ be the corresponding subgraph such that $M = E(\mathcal{Q})$. We have $\eta\text{-}\mathsf{sup}_{\mathcal{Q}}(e') \geq \eta\text{-}\mathsf{sup}_{\mathcal{F}}(e') \geq \kappa_\eta(e_i)$. Also, $\eta\text{-}\mathsf{sup}_{\mathcal{Q}}(e') = \delta_\eta(\mathcal{Q})$, because $e' \in L_b$. Since $j > b$ and $e_j \in M$, $\kappa_\eta(e_j) \geq \delta_\eta(\mathcal{Q})$ (based on Lemma 1). Combining the above, we conclude $\kappa_\eta(e_i) \leq \kappa_\eta(e_j)$. □

We prove the convergence of our proposed algorithm using ideas similar to the proof of deterministic h-index algorithm in [65]. In Theorem 5 we showed that *Phase I* provides upper-bounds on truss values of the input probabilistic graph. In the following, we prove that upper-bounds are monotonically non-increasing and are lower-bounded by truss values.

Theorem 7. *For all t and all edges e in $\mathcal{G}$, we have* ***(1)*** *$h_{t+1}(e) \leq h_t(e)$,* ***(2)*** *$h_t(e) \geq \kappa_\eta(e)$, where by $h_t(e)$ we denote the h-index of e after the t-th iteration of Phase I and Phase II together.*

Proof. **(1)** We prove this by induction on t. Initially, when $t = 0$, $h_0(e)$ is equal to $\eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)$. Let $h_1^p(\cdot)$ be the processed values after completion of *Phase I* at iteration 1. As shown in [65], throughout *Phase I*, the upper-bounds can only decrease, so $h_1^p(e) \leq h_0(e)$, for each edge e. The $h_1^p(\cdot)$ values are passed to *Phase II*. The block of steps 6-9 of *Phase II* (Algorithm 9) checks all the values equal or less than $h_1^p(e)$ for each edge e, and finds the maximum value for which the condition in line 6 holds. Let $h_1(e)$ be the obtained maximum value. Thus, we have $h_1(e) \leq h_1^p(e) \leq h_0(e)$. Assume the property is true up to t. For iteration $t+1$, *Phase I* needs to process the values $h_t(\cdot)$ obtained from the previous iteration (i.e. t) by *Phase II*. Let $h_{t+1}^p(e)$ be the processed values after completion of *Phase I* at iteration $t+1$. For an edge e, by the induction hypothesis, and monotonicity of *Phase I* itself [65], we have $h_{t+1}^p(e) \leq h_t(e) \leq h_{t-1}(e)$. Then, this value is passed through *Phase II*. As discussed above, this value is processed using lines 6-9 in Algorithm 9 which make sure that $h_{t+1}(e) \leq h_t^p(e)$. Thus, we have $h_{t+1}(e) \leq h_t(e)$.

(2) We prove the property by induction on t. For $t = 0$, $h_0(e) = \eta\text{-}\mathsf{sup}_{\mathcal{G}}(e) \geq \kappa_\eta(e)$. Let us assume that for t, $h_t(e) \geq \kappa_\eta(e)$. Now, we focus on the computation of $h_{t+1}(e)$. Using the induction step and the fact that *Phase I* provides an upper-bound on $\kappa_\eta(e)$ for each edge e (please refer to Theorem 5), we can write: $h_{t+1}^p(e) \geq \kappa_\eta(e)$. Consider the computation of $h_{t+1}(e)$ by *Phase II* which is based on the value produced by *Phase I* (i.e. $h_{t+1}^p(e)$). Let $\mathcal{F}$ be $(\kappa_\eta(e), \eta)$-truss which contains e. Also, let S be the set of all supporting triangles $\triangle$ in $\mathcal{F}$ for edge e, such that $\forall e', e'' \neq e \in \triangle$, $\min(\kappa_\eta(e'), \kappa_\eta(e'')) \geq \kappa_\eta(e)$. Using the property of truss value we know that $\Pr[\mathsf{sup}(e) \geq \kappa_\eta(e) \mid S] \geq \eta$. To obtain $h_{t+1}(e)$, *Phase II* checks the condition $\Pr[\mathsf{sup}(e) \geq h_{t+1}^p(e) \mid \Gamma] \geq \eta$ (line 6, Algorithm 9), where Γ is the set of all the triangles that contain e, and is detected by *Phase II* since $\rho_\triangle \geq h_{t+1}^p(e)$, for each $\triangle \in \Gamma$, where $\rho_\triangle$ is the minimum h-index value of the edges other than e in $\triangle$ (line 19, Algorithm 9). If $\Pr[\mathsf{sup}(e) \geq h_{t+1}^p(e) \mid \Gamma] \geq \eta$ holds, then $h_{t+1}(e) = h_{t+1}^p(e) \geq \kappa_\eta(e)$. Otherwise, all the k values smaller than $h_{t+1}^p(e)$ are checked. In the worst case, consider the computation of the probability when k becomes equal to $\kappa_\eta(e)$. Let Γ be the updated set to contain all $\triangle$ with $\rho_\triangle \geq k$. We claim that $S \subseteq \Gamma$. For each triangle $\triangle \in S$, and $\forall e', e'' \neq e \in \triangle$, we have $\kappa_\eta(e'), \kappa_\eta(e'') \geq \kappa_\eta(e)$. In addition, based on Theorem 5,

$h^p_{t+1}(e') \geq \kappa_\eta(e') \geq \kappa_\eta(e) = k$, and $h^p_{t+1}(e'') \geq \kappa_\eta(e'') \geq \kappa_\eta(e) = k$. Thus, $\rho_\triangle = \min(h^p_{t+1}(e'), h^p_{t+1}(e'')) \geq k = \kappa_\eta(e)$, which results in $\triangle \in \Gamma$. Using Proposition 3, $\Pr[\mathsf{sup}(e) \geq k \mid \Gamma] \geq \Pr[\mathsf{sup}(e) \geq k \mid S]$. If for $k = \kappa_\eta(e)$, $\Pr[\mathsf{sup}(e) \geq \kappa_\eta(e) \mid \Gamma] < \eta$, then $\Pr[\mathsf{sup}(e) \geq \kappa_\eta(e) \mid S] < \eta$, which is a contradiction with the definition of $\kappa_\eta(e)$. As a result, we should have $\Pr[\mathsf{sup}(e) \geq \kappa_\eta(e) \mid \Gamma] \geq \eta$. Thus, $h_{t+1}(e) \geq \kappa_\eta(e)$. □

Theorem 8. *Given any level L_i, for all $t \geq i$, and $e \in L_i$, we have $h_t(e) = \kappa_\eta(e)$.*

Proof. We prove this by induction on i. For $i = 0$, let us consider the set of edges e with minimum η-$\mathsf{sup}_\mathcal{G}(e)$ in $\mathcal{G}$. For these edges, $h_t(e) = \eta$-$\mathsf{sup}_\mathcal{G}(e) = \max_k\{\Pr[\mathsf{sup}_\mathcal{G}(e) \geq k] \geq \eta\} = \kappa_\eta(e)$. Assume that the theorem is true up to level i. As a result, $\forall t \geq i$, and $\forall e \in \bigcup_{j \leq i} L_j$, $h_t(e) = \kappa_\eta(e)$. Let e_a be an arbitrary edge in level $i+1$, and $L' = \bigcup_{j \geq i+1} L_j$. Consider the partition of all the triangles which contain e_a into two sets S_l and S_h. Triangles in S_l contain some edge outside L', and those in S_h have all their edges contained in L'. For each triangle $\triangle \in S_l$, there is some $e_b \neq e_a \in \triangle$ such that $e_b \in L_k$, where $k \leq i$. Using induction hypothesis, we have $h_t(e_b) = \kappa_\eta(e_b)$. Also, since $e_a \in L_{i+1}$, using Theorem 6, we have $h_t(e_b) = \kappa_\eta(e_b) < \kappa_\eta(e_a) \leq h_t(e_a)$, where for the last inequality we have used the property (2) in Theorem 7.
Let us focus on the computation of $h_{t+1}(e_a)$ (lines 6-9, Algorithm 9). The algorithm checks the condition $\Pr[\mathsf{sup}(e_a) \geq r \mid \Gamma] \geq \eta$, where Γ is the set of triangles $\triangle$ which contain e_a, and have $\rho_\triangle \geq r$, where $r = h_t(e_a)$. We recall that $\rho_\triangle = \min\{h_t(e'), h_t(e'')\}$ (line 19, Algorithm 9), where $e', e'' \neq e_a \in \triangle$. *Set Γ is updated each time to correspond to the r value being used for computation of the condition.* For every $\triangle \in S_l$, by the previous argument, there is some $e_b \neq e_a \in \triangle$, such that $h_t(e_b) < h_t(e_a)$. Thus, $\rho_\triangle < h_t(e_a)$, and these triangles are not considered in the computation. As a result set Γ will consist of triangles from set S_h only; $\Gamma \subseteq S_h$. Let $\mathcal{G}'$ be the graph such that $L' = E(\mathcal{G}')$. Using Proposition 3, we can write

$$\Pr[\mathsf{sup}(e_a) \geq r \mid \Gamma] \leq \Pr[\mathsf{sup}(e_a) \geq r \mid S_h], \text{ for any } r, \tag{4.7}$$

Since $e_a \in L_{i+1}$, η-$\mathsf{sup}_{\mathcal{G}'}(e_a) = \delta_\eta(\mathcal{G}')$. Thus, we have

$$\Pr[\mathsf{sup}_{\mathcal{G}'}(e_a) \geq \delta_\eta(\mathcal{G}')] \geq \eta, \tag{4.8}$$

$$\Pr[\mathsf{sup}_{\mathcal{G}'}(e_a) \geq r'] < \eta, \text{ for any } r' > \delta_\eta(\mathcal{G}'), \tag{4.9}$$

The above equations are based on the definition of probabilistic support of an edge: η-$\mathsf{sup}_{\mathcal{G}'}(e_a) = \max_k\{\Pr[\mathsf{sup}_{\mathcal{G}'}(e_a) \geq k] \geq \eta\}$. By definition of S_h, edges contained in

the triangles of S_h are part of $L' = E(\mathcal{G}')$. Thus, triangles in S_h are contained in $\mathcal{G}'$. As mentioned earlier, since computation of $\Pr[\mathsf{sup}_{\mathcal{G}'}(e_a) \geq r']$ is done by considering triangles in S_h, the values of $\Pr[\mathsf{sup}_{\mathcal{G}'}(e_a) \geq r']$ and $\Pr[\mathsf{sup}(e_a) \geq r' \mid S_h]$ are the same. Therefore, $\Pr[\mathsf{sup}(e_a) \geq r' \mid S_h] < \eta$. Combining this with Equation 4.7, for $r > \delta_\eta(\mathcal{G}')$ we obtain:

$$\Pr[\mathsf{sup}(e_a) \geq r \mid \Gamma] \leq \Pr[\mathsf{sup}(e_a) \geq r \mid S_h] < \eta, \tag{4.10}$$

Since $\Pr[\mathsf{sup}(e_a) \geq r \mid \Gamma] < \eta$, the algorithm checks r values less than or equal to $\delta_\eta(\mathcal{G}')$, thus $h_{t+1}(e_a) \leq \delta_\eta(\mathcal{G}')$. In addition, based on Lemma 1, we have $\delta_\eta(\mathcal{G}') \leq \kappa_\eta(e_a)$. Thus, $h_{t+1}(e_a) \leq \kappa_\eta(e_a)$. On the other-hand, based on property (2) in Theorem 7, we have $h_{t+1}(e_a) \geq \kappa_\eta(e_a)$. Combining $h_{t+1}(e_a) \leq \kappa_\eta(e_a)$ and $h_{t+1}(e_a) \geq \kappa_\eta(e_a)$, we conclude that $h_{t+1}(e_a) = \kappa_\eta(e_a)$. Since e_a was an arbitrary edge in L_{i+1}, this concludes the proof by induction. □

Based on the above theorem, we can express the following corollary which shows that convergence is guaranteed in a finite number of iterations.

Corollary 3. *Given a probabilistic graph $\mathcal{G}$, and threshold η, let l be the maximum value for the degree level, such that $L_l \neq \emptyset$. There exists some $t \leq l$ such that $h_t(e) = \kappa_\eta(e)$, for all edges.*

4.3 Complexity Analysis

In this section we present the time complexity of our proposed algorithm, *proHIT*.

Theorem 9. *Given a probabilistic graph $\mathcal{G}$, proHIT computes the truss decomposition of $\mathcal{G}$ in $O\big(tk_{\max,\eta}\psi m\big)$, where t is the total number of iterations $k_{\max,\eta} = \max_e\{\eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)\}$, ψ is the minimum number of spanning forests needed to cover all edges of $\mathcal{G}$, and m is the number of the edges.*

Proof. The time complexity of Algorithm 8 is dominated by the time complexity of *Phase II*, since h-index computation of edges is done by dynamic programming (DP) algorithm which has quadratic time complexity. In contrast, the h-index computation in *Phase I* can be done in linear time.

To analyze the time complexity of *Phase II* (given in Algorithm 9), we should note that for each edge $e = (u, v)$, the first time computation of the probability

$\Pr[\mathsf{sup}(e) \geq H \mid \Gamma]$ in line 6 (Algorithm 9), takes $O(Hj_0)$ time, where $j_0 = |\Gamma|$, $H = h(e)$, and Γ is as given in the algorithm. For the next iterations in the while loop (line 6, Algorithm 9), using *Modified DP*, the computation is performed on $\mathcal{T}^i$ only, where $i = H-1, \cdots, 0$, and $\mathcal{T}^i$ is as before. In the worst case, the while loop is repeated H times. Let us assume that $j_1 = \left|\mathcal{T}^{H-1}\right|$, $j_2 = \left|\mathcal{T}^{H-2}\right|$, $\cdots$, $j_{k_e} = |\mathcal{T}^0|$. It is obvious that $j_0 + j_1 + \cdots + j_{k_e} = k_e$, where k_e is the number of common neighbors of u and v. We have that $k_e \subseteq O(\min\{d(u), d(v)\})$, where $d(u)$ and $d(v)$ are the degree of vertices u and v, respectively. Therefore, the while loop takes $O(j_0 H) + O(j_1(H-1)) + \cdots + O(j_{k_e-1}1)$ time. In the worst case then, the time complexity of the while loop is bounded by $O(j_0 k_{\max,\eta}) + O(j_1 k_{\max,\eta}) + \cdots + O(j_{k_e-1} k_{\max,\eta})$, which is equal to $O(k_{\max,\eta} k_e) \subseteq O(k_{\max,\eta} \min\{d(u), d(v)\})$.

Moreover, the iteration over each neighbor of edge e in line 12 (Algorithm 9), takes $O(\min\{d(u), d(v)\})$. As a result, the time complexity of *Phase II* is bounded by

$$\sum_{e \in E} \Big(O\big(k_{\max,\eta} \min\{d(u), d(v)\}\big) + O\big(\min\{d(u), d(v)\}\big) \Big)$$

Thus, the time complexity becomes:

$$\sum_{e \in E} O\big(k_{\max,\eta} \min\{d(u), d(v)\}\big) \subseteq O\big(k_{\max,\eta} \psi m\big).$$

It should be noted that $\psi \leq \min\{d_{\max}, \sqrt{m}\}$, where $d_{\max}$ is the maximum degree in the graph. Let t be the total number of iterations. The total time complexity is $O\big(t k_{\max,\eta} \psi m\big)$. In the worst case the number of iterations, t, is bounded by the degree levels as discussed in Theorem 8 and Corollary 3 in Section 4.2. The number of degree levels are bounded by $\beta = k_{\max,\eta}$.

□

The running times of the baseline algorithms, *PDT* and *PAPT* are dominated by $O(d_{max} \psi m)$. However, *proHit* algorithm performs much better in practice. This is because β in the above proof is worst-case upper-bound on t, the number of iterations, and is not representative of practical performance. As shown in our experiments t is much less than β in practice as the h-index of several edges will decrease simultaneously in each iteration.

For example, let us consider the flickr dataset, with $\beta = k_{\max,\eta} = 49$. However, as can be seen in Figure 4.6, for flickr with $\eta = 0.1$, the total number of iterations is about 18 which is much smaller than β. This trend is also evident for other datasets.

Graph	$\|V\|$	$\|E\|$	$\|\triangle\|$	Reference
flickr	24,125	300,836	8,857,038	[9]
dblp	684,911	2,284,991	4,582,169	[9]
biomine	1,008,201	6,722,503	93,716,868	[9]
uk-2014-tpd	1,766,010	15,283,718	259,040,749	[8, 7]
itwiki-2013	1,016,867	23,429,644	89,901,299	[8, 7]
in-2004	1,382,908	27,182,946	464,257,245	[8, 7]
ljournal-2008	5,363,260	49,514,271	411,155,444	[8, 7]
enwiki-2013	4,206,785	91,939,728	304,083,160	[8, 7]

Table 4.3: Dataset Statistics

4.4 Experiments

In this section, we present our experimental results. Our implementations are in Java and the experiments are conducted on a machine with Intel i7, 2.2Ghz CPU, and 12Gb RAM, running Ubuntu 18.04. The statistics for the datasets are shown in Table 4.3. We report the number of vertices $|V|$, the number of edges $|E|$, and the number of triangles $|\triangle|$. Datasets with real probability values are *flickr*, *dblp*, and *biomine*.

flickr is a popular online community for sharing photos. Nodes are users in the network, and the probability of an edge between two users is obtained based on the Jaccard coefficient of the interest groups of the two users [9, 61].

dblp comes from the well-known bibliography website. Nodes correspond to authors, and there is an edge between two authors if they co-authored at least one publication. The existence probability of each edge is measured based on an exponential function of the number of collaborations between two users [9, 61].

biomine contains biological interactions between proteins. The probability of an edge represents the confidence level that the interaction actually exists [9].

The rest of the datasets are social networks and web graphs which are obtained from Laboratory of Web Algorithms [8, 7]. For these datasets we generated probability values uniformly distributed in $(0, 1]$.

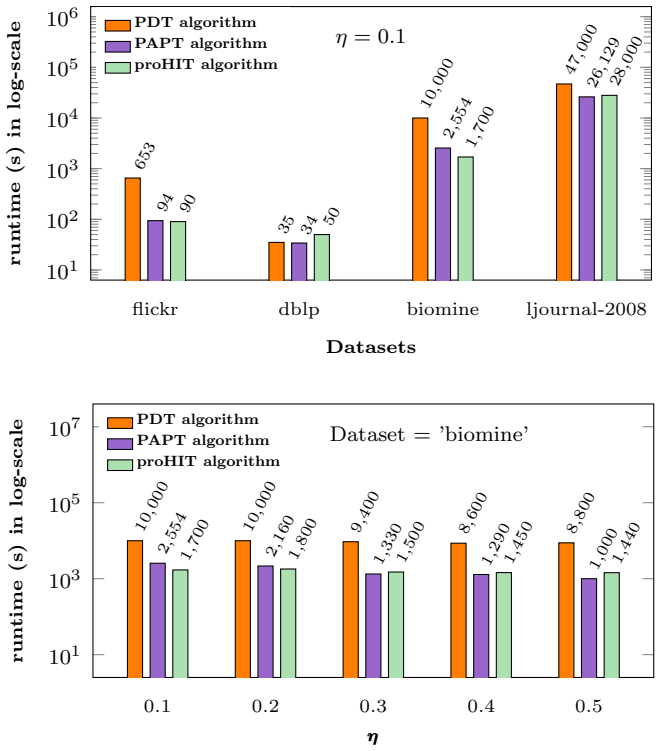


Figure 4.3: Running time of our proposed algorithm, proHIT, versus PDT and PAPT (baselines) for truss decomposition in probabilistic graphs.

4.4.1 Efficiency Evaluation

In this section we report the running time of our proposed algorithm, *proHIT*, versus the state-of-the-art peeling algorithms, which we refer to as *PDT* (peeling-DP-truss) [34] and *PAPT* (peeling-approximate-truss) [25] (which we discussed it in Chapter 3). Both *PDT* and *PAPT* algorithms are based on iterative edge removal process which removes edges e with smallest probabilistic support, $\eta\text{-}\mathsf{sup}_{\mathcal{G}}(e)$, and updating probabilistic support, $\eta\text{-}\mathsf{sup}_{\mathcal{G}\setminus\{e\}}(e')$, of the affected edges e' in $\mathcal{G}\setminus\{e\}$. *PDT* uses dynamic programming for computing and updating probabilistic support of edges. However, *PAPT* uses statistical methods to approximate probabilistic support of edges, and as such, is an approximate algorithm. We use DP as an abbreviation for dynamic programming.

In our experiments, we set threshold $\eta = 0.1, \cdots, 0.5$. The running times in log-scale are shown in Figures 4.3 and 4.4. In Figure 4.3 we present the running times for *flickr*, *dblp*, *biomine*, and *ljournal-2008* using $\eta = 0.1$ as an example. In Figure 4.4, we separate the running times for the rest of the datasets due to different scales in their

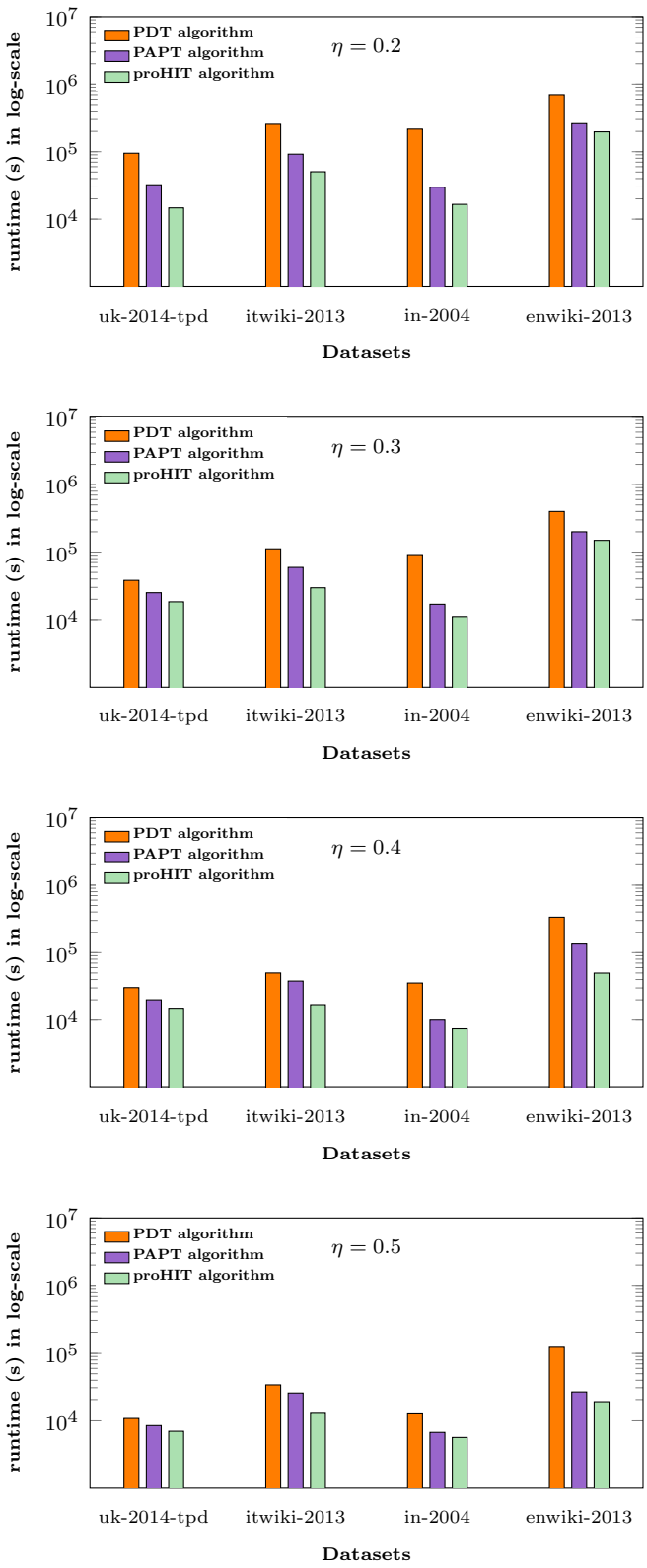


Figure 4.4: Running time of our proposed algorithm, proHIT, versus PDT and PAPT edge peeling (baselines) for truss decomposition on larger datasets with different values of η.

plot of running times. For these datasets we show the results for $\eta = 0.2, \cdots, 0.5$, since *PDT* cannot complete in reasonable time for $\eta = 0.1$. Moreover, for each dataset, we obtain the average of the maximum probabilistic support, $\text{avg}_\eta\{k_{\max,\eta}\}$, and the average of maximum truss value, $\text{avg}_\eta\{\max_e\{\kappa_\eta(e)\}\}$, over $\eta = 0.1, \cdots, 0.5$. These statistics are shown in Table 4.4, second and third columns, respectively.

As can bee seen in Figures 4.3 and 4.4, our *proHIT* algorithm is significantly faster than *PDT*, especially on networks containing a large number of triangles, and having large value of $\text{avg}_\eta\{k_{\max,\eta}\}$. For instance, for *biomine* (Figure 4.3) which is such a dataset, the gain of our algorithm compared to *PDT* is 84%, making *proHIT* six times faster than *PDT*. In fact, for biomine, for η equal to 0.1 and 0.2, *proHIT* is even better than the approximate algorithm *PAPT*, which, we recall, is an approximate algorithm. For the other η's for biomine, *proHIT* is slightly slower than *PAPT*. To reiterate, this is a welcome surprise because our proposed algorithm, *proHIT*, achieves a similar performance as *PAPT*, but without sacrificing the exactness of the solution.

In terms of running time on the smaller datasets, *flickr* and *dblp*, *proHIT* produces the results in 1.5 minutes and 1 minute, respectively. The number of triangles in *flickr* is twice larger than in *dblp* while having much less edges. We observe that *proHit* has a similar performance as *PAPT*. Both *proHit* and *PAPT* are faster than *PDT*, except on *dblp*. We recall that *dblp* is the smallest dataset in terms of probabilistic support and truss value of its edges, and as such it does not cause too much work for Dynamic Programming needed for *PDT*. As we see in the rest of the charts in Figure 4.4 *proHIT* significantly outperforms *PDT* and *PAPT* as the datasets get larger.

The running times of all algorithms increase for *ljournal-2008*, which is reasonable, because this graph has 49 million edges with $\text{avg}_\eta\{k_{\max,\eta}\}$ equal to 911. For *ljournal-2008*, *proHIT* computes truss decomposition faster than *PDT* with a gain of 40 percent.

The running times continue to increase for the remaining datasets. This is because for these datasets $\text{avg}_\eta\{k_{\max,\eta}\}$ is much larger as shown in Table 4.4. For instance, for *itwiki-2013*, $\text{avg}_\eta\{k_{\max,\eta}\}$ is 4574. Moreover, for *uk-2014-tpd* and *in-2004* the ratio of the number of triangles to the number of edges is much higher than *ljournal-2008*.

As can be seen, for these graphs, *proHIT* is again significantly faster than its counterpart *PDT* (as an exact method). For instance, for *uk-2014* and *itwiki-2013* with $\eta = 0.3$, *proHIT* is about 2 and 3 times faster than *PDT*. Comparing *proHIT* with *PAPT* (which is an approximate method) shows that *proHIT* is on average 24%

Dataset	$\mathbf{avg_\eta\{k_{max,\eta}\}}$	$\mathbf{avg_\eta\{max_e\{\kappa_\eta(e)\}\}}$
flickr	48	47
dblp	38	11
biomine	135	27
ljournal-2008	911	35
uk-2014-tpd	1252	51
in-2004	1890	35
itwiki-2013	4574	6
enwiki-2013	14429	8

Table 4.4: The values of $avg_\eta\{k_{\max,\eta}\}$, *and* $avg_\eta\{\max_e\{\kappa_\eta(e)\}\}$ *over* $\eta = 0.1, \cdots, 0.5$.

Dataset	$\boldsymbol{k_{max,\eta}}$	$\mathbf{max_e\{\kappa_\eta(e)\}}$	$\boldsymbol{\eta}$
	151	33	0.1
	143	30	0.2
biomine	135	28	0.3
	125	25	0.4
	121	18	0.5

Table 4.5: $k_{\max,\eta}$, $\max_e\{\kappa_\eta(e)\}$, η.

faster than *PAPT* without sacrificing the exactness of the solution. For *itwiki-2013* with $\eta = 0.5$, *proHIT* can complete truss decomposition in about 4 hours, while *PAPT* takes about 7 hours. Also, truss decomposition of *in-2004* using *proHIT* is 30 min faster than the one using *PAPT*. A similar trend can be observed for other values of η.

In general, as the number of edges and triangles in the input graph increase, the running times of the algorithms becomes larger. The conclusion that we get is that for large graphs the performance of the *proHIT* algorithm is better than the peeling approaches since they require updating probabilistic supports many times during the algorithm process to obtain the truss values of the edges.

Note. It should be noted that as η increases, probabilistic support and truss values of edges decrease which lead to decrease in the running times of the algorithms. In Table 4.5, we show the trend for one of our dataset, *biomine*, in which $k_{\max,\eta}$ and $\max_e\{\kappa_\eta(e)\}$ decrease as η increases.

Next, we discuss why *proHIT* is faster than *PDT*. The most expensive part of both algorithms is executing DP routines, with quadratic run-times in the number of triangles containing each edge. However, their number and sizes are different in

Dataset	Size of DP				# of times DP is executed	
	Avg	Max	Avg	Max		
	PDT		proHIT		PDT	proHIT
flickr	154	452	85	280	12 M	1.7 M
dblp	28	220	6	114	2.8 M	3.6 M
biomine	249	27970	62	17042	85.6 M	19.7 M
ljournal-2008	159	4324	44	503	505 M	247 M

Table 4.6: Average and maximum sizes of dynamic programming (DP), as well as the number of executions of DP for PDT and proHIT.

proHIT and *PDT*. Step 6 in *Phase II* (Algorithm 9) of *proHIT* uses DP to check the validity of the upper-bounds on the truss value of edges at each iteration of the algorithm. Also, at the beginning of *proHIT*, the upper-bound of each edge e is set to its η-$\mathsf{sup}_{\mathcal{G}}(e)$ which is obtained using DP (Algorithm 8, line 3). In *PDT*, DP is used after each edge removal, and all the edges that are neighbors of a peeled edge need to have their probabilistic support recomputed using DP.

Given a probabilistic graph $\mathcal{G}$, and edge $e = (u, v)$, let k_e be the number of common neighbors of u and v used for computing probabilistic support of e in $\mathcal{G}$. The time complexity of the computation by DP is $O(k_e^2)$ [34]. We refer to k_e as the *size* of DP. In *proHIT*, in Phase II, not all neighbors of u and v are used for DP but rather only those neighbors that can contribute to the final truss value of e (recall set Γ and Equation 4.1 in Section 4.1). As such, in *proHIT*, the size of DP is typically smaller than the total number of all the common neighbors of u and v. This is in contrast to *PDT*, which runs DP using all the remaining neighbors of an edge at that point in the peeling process. In essence, *proHIT* performs DP on smaller and only the effective set of neighbours for each edge, resulting in a considerable speedup.

We report the average and maximum sizes of DP for both algorithms in Table 4.6, for *flickr*, *dblp*, *biomine*, and *ljournal-2008*. As can be seen, for all the selected datasets, these sizes are much smaller for *proHIT* than for *PDT*. This is particularly important in large datasets, *biomine* and *ljournal-2008*, in which the average size for *proHIT* is about 3.5 and 4 times smaller than for *PDT*. In addition, in the last column of Table 4.6, we report the number of times DP is performed for both *PDT* and *proHIT* algorithms. The difference is noticeable for large datasets. For instance, on *ljournal-2008*, the number of executions of DP by *proHIT* is half of those performed by *PDT*.

Memory Usage. In Figure 4.5, we compare the memory consumption of *proHIT* versus that of *PDT* and *PAPT* on selected datasets. The trend can be verified for other datasets as well. As can be seen, for *biomine* the memory consumption of *proHIT* is 12 times and 6 times smaller than those of *PAPT* and *PDT*, respectively. This also holds for other datasets. For instance, for *ljournal-2008*, which is a large dataset, *PAPT* requires 90% more space than *proHIT*. Thus, Figure 4.5 confirms that *proHIT* consumes a smaller amount of memory for computing truss decomposition. This is because *PDT* and *PAPT* are edge peeling based algorithms which require maintaining the global information of the graph at each step of the algorithm, while *proHIT* uses local information only.

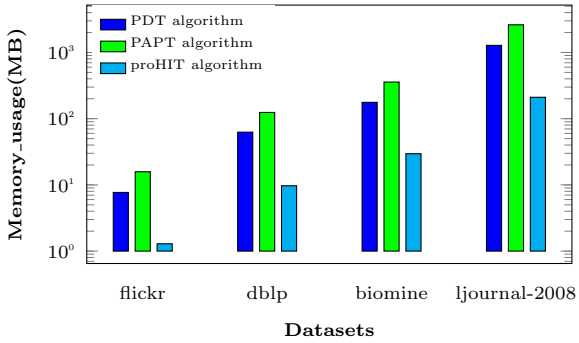


Figure 4.5: Memory usage of proposed algorithm versus the state-of-the-art edge peeling algorithms

4.4.2 Convergence Speed

In this section we further evaluate the execution of *proHIT* as it unfolds with time. We look at the average distance from the truss values over the sequence of iterations for selected small to large datasets (see Figure 4.6). The average distance decreases fast for *flickr*, *dblp*, and *biomine*, and more gradually for ljournal-2008, in-2004, and uk-2014-tpd. These results show that *proHIT* can produce high-quality near-results in only a fraction of the total number of iterations. For instance, for ljournal-2008 with $\eta = 0.1$, the average distance becomes less than 0.01 at iteration 20, which is about one third of the total number of required iterations (about 60, see the end of the curve). This can be a desirable property in graph mining where the user would like to see near-results as the execution progresses.

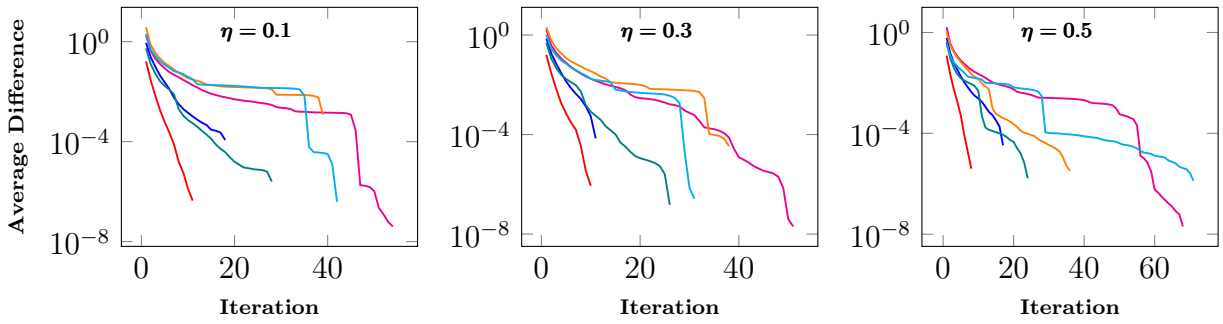


Figure 4.6: Average difference between the truss value and the upper bound over iterations for different values of η, for DBLP—, Flickr—, biomine—,ljournal-2008—, in-2004, uk-2014-tpd (best viewed in color).

4.5 Conclusions

We presented a novel algorithm, *proHIT*, for computing truss decomposition in large probabilistic graphs. Our algorithm is based on an h-index updating approach. Unlike the edge peeling strategy, *proHIT* accesses the edges in a local fashion which makes it memory efficient. *proHIT* includes two main phases. *Phase I* is responsible for updating the edges' h-index without considering edge probabilities. This phase can provide a fast-to-compute upper-bound on truss values of the edges. *Phase II* takes care of the probabilistic nature of truss decomposition and further tightens the upper-bounds obtained in the previous phase. *proHIT* is an exact algorithm and is significantly faster than the state-of-the-art exact algorithm of [34]. While being an exact algorithm, *proHIT* can also produce near-results in only a fraction of iterations needed for computing the full exact solution.

Chapter 5

Nucleus Decomposition in Probabilistic Graphs: Hardness and Algorithms

This chapter focuses on our proposed algorithms for solving nucleus decomposition in probabilistic graphs. The organization of this chapters is as follows. Section 5.1 provides an overview of literature on nucleus decomposition. Then, we introduce the concept of nucleus decomposition in deterministic graphs in Section 5.2 as well as our proposed definitions (local, global, and weakly-global nucleus decompositions) in probabilistic context in Section 5.3. Section 5.4 contains formal proofs to show hardness results for computing global and weakly-global decompositions. Next, we present our algorithms in Sections 5.5 and 5.6. Section 5.7 discusses our experimental results, and we conclude in Section 5.8.

5.1 Related Work

The related work to nucleus decomposition is core and truss decomposition which were discussed in more details in Chapters 2 and 3. Building on the well-studied notions of core and truss decomposition, Sarıyüce et al. [66] introduce nucleus decomposition in *deterministic* graphs. They propose efficient algorithms for finding nucleus decomposition in deterministic graphs. Also, they show that nucleus decomposition gives a global, hierarchical snapshot of dense substructures, and outputs dense subgraphs of higher quality than other state-of-the-art solutions. In a more

recent work, Sarıyüce et al. [64] propose efficient distributed algorithms for nucleus decomposition. Their approach uses local information of the graph and computes nucleus decomposition with convergence guarantees. Our work is the first to study nucleus decomposition in probabilistic graphs.

5.2 Deterministic Nuclei

Let $G = (V, E)$ be an undirected graph, where V is a set of vertices, and E is a set of edges. For a vertex $v \in V$, let $N(v)$ be the set of v's neighbors: $N(v) = \{u : (u, v) \in E\}$. The (deterministic) degree of v in G, is equal to $|N(v)|$.

Nucleus decomposition in deterministic graphs. Nucleus decomposition is a generalization of core and truss decompositions [66, 67]. Each nucleus is a subgraph which contains a dense cluster of cliques. The formal definitions are as follows.

Let r, s with $r < s$ be positive integers. We call cliques of size r, *r-cliques*, and denote them by R, R', etc. Analogously, we call cliques of size s, *s-cliques*, and denote them by S, S', etc.

Definition 5. *The* s-support *of an r-clique R in G, denoted by $s\text{-supp}_G(R)$, is the number of s-cliques in G that contain R.*

Definition 6. *Two r-cliques R and R' in G, are* s-connected*, if there exists a sequence $R = R_1, R_2, \cdots, R_k = R'$ of r-cliques in G such that for each i, there exists some s-clique in G that contains $R_i \cup R_{i+1}$.*

Now nucleus decomposition is as follows.

Definition 7. *Let k be a positive integer. A* ***k-(r, s)-nucleus*** *is a maximal subgraph H of G with the following properties.*

1. *H is a union of s-cliques: every edge in H is part of an s-clique in H.*
2. *$s\text{-supp}_H(R) \geq k$ for each r-clique R in H.*
3. *Each pair R, R' of r-cliques in H is s-connected in H.*

For simplicity, whenever clear from the context, we will drop the use of prefix s from the definition of support and connectedness.

When $r = 1, s = 2$, r-cliques are nodes, s-cliques are edges, and k-$(1, 2)$-nucleus is the well-known notion of k-core. When $r = 2, s = 3$, r-cliques are edges, s-cliques

are triangles, and k-$(2,3)$-nucleus is the well-known notion of k-truss. [66] shows that k-$(3,4)$-nucleus, where we consider triangles contained in 4-cliques, provides much more interesting insights compared to k-core and k-truss in terms of density and hierarchical structure. As such, in this dissertation, we also focus on the $r = 3, s = 4$ case. For simplicity, we will drop using r and s and assume them to be 3 and 4, respectively. In particular, we will refer to k-$(3,4)$-nucleus as simply k-nucleus.

5.3 Probabilistic Nuclei

Let $\mathcal{G} = (V, E, p)$ be a probabilistic graphs, where where V and E are as before and $p : E \to (0, 1]$ is a function that maps each edge $e \in E$ to its existence probability p_e. We consider the most common probabilistic model in which edges exist independent of each other (cf. [9, 35, 37]). Also, let $G \sqsubseteq \mathcal{G}$ be a possible world for $\mathcal{G}$. As explained in the previous chapters, conceptually, the possible worlds are obtained by flipping a biased coin for each edge independently, according to its probability. The probability of a possible world $G = (V, E_G) \sqsubseteq \mathcal{G}$ is as follows:

$$\Pr[G \mid \mathcal{G}] = \prod_{e \in E_G} p_e \prod_{e \in E \setminus E_G} (1 - p_e). \tag{5.1}$$

We will use $\mathcal{G}$, $\mathcal{G}'$, $\mathcal{F}$, $\mathcal{F}'$ to denote probabilistic graphs.

Nucleus decomposition in probabilistic graphs. We now define three variants of nucleus decomposition in probabilistic graphs which are based on Definitions 8 and 9 we give below. These variants relate to the nature of nucleus and we refer to them as **local ($\boldsymbol{\ell}$)**, **global (g)**, and **weakly-global (w)**.

Definition 8. *Let $\mathcal{H}$ be a probabilistic graph, $\triangle$ a triangle, and μ a mode in set $\{\boldsymbol{\ell}, \mathbf{g}, \mathbf{w}\}$. Then, $X_{\mathcal{H},\triangle,\mu}$ is a random variable that takes integer values k with tail probability*

$$Pr(X_{\mathcal{H},\triangle,\mu} \geq k) = \sum_{H \sqsubseteq \mathcal{H}} Pr[H \mid \mathcal{H}] \cdot \mathbb{1}_{\mu}(H, \triangle, k), \tag{5.2}$$

where indicator variable $\mathbb{1}_{\mu}(H, \triangle, k)$ is defined depending on mode μ as follows.

$\mathbb{1}_{\boldsymbol{\ell}}(H, \triangle, k) = 1$ *if $\triangle$ is in H, and the support of $\triangle$ in H is at least k.*

$\mathbb{1}_{\mathbf{g}}(H, \triangle, k) = 1$ *if $\triangle$ is in H, and H is a deterministic k-nucleus.*

$\mathbb{1}_{\mathbf{W}}(H, \triangle, k) = 1$ *if* $\triangle$ *is in* H*, and there is a subgraph* H' *of* H *that contains* $\triangle$ *and is a deterministic* k*-nucleus.*

It is clear that $(\mathbb{1}_{\mathbf{g}}(H, \triangle, k) = 1) \implies (\mathbb{1}_{\mathbf{W}}(H, \triangle, k) = 1) \implies (\mathbb{1}_{\boldsymbol{\ell}}(H, \triangle, k) = 1)$.

In the above definition, $\mathbb{1}_{\boldsymbol{\ell}}(H, \triangle, k)$ has a local quality because a possible world G satisfies its condition if it provides sufficient support to triangle $\triangle$ without considering other triangles in H. On the other hand, $\mathbb{1}_{\mathbf{g}}(H, \triangle, k)$ and $\mathbb{1}_{\mathbf{W}}(H, \triangle, k)$ have a global quality because a possible world H satisfies their conditions only when other triangles in H are considered as well (creating a nucleus together).

In the following, as *preconditions* for cohesiveness, we will assume *cliqueness* and *connectedness* for the nuclei subgraphs we define. Specifically, we will only consider subgraphs $\mathcal{F}$, which, ignoring edge probabilities, are unions of 4-cliques, and where each pair of triangles in $\mathcal{F}$ is connected in $\mathcal{F}$.

Definition 9. *Let* $\mathcal{G} = (V, E, p)$ *be a probabilistic graph. Given threshold* $\theta \in [0, 1]$*, integer* $k \geq 0$*, and* $\mu \in \{\boldsymbol{\ell}, \mathbf{g}, \mathbf{w}\}$*, a* μ*-*(k, θ)*-nucleus* $\mathcal{F}$ *is a maximal subgraph of* $\mathcal{G}$*, such that* $Pr(X_{\mathcal{H},\triangle,\mu} \geq k) \geq \theta$ *for each triangle* $\triangle$ *in* $\mathcal{F}$*.*

Moreover, the (μ, θ)- nucleusness *(or simply nucleusness when* μ *and* θ *are clear from context) of a triangle* $\triangle$ *is the largest value of* k *such that* $\triangle$ *is contained in a* μ*-*(k, θ)*-nucleus.*

Intuitively for $\mu = \boldsymbol{\ell}$, from a probabilistic perspective, a subgraph $\mathcal{H}$ of $\mathcal{G}$ can be regarded as a cohesive subgraph of $\mathcal{G}$ if the support of every triangle in $\mathcal{H}$ is no less than k with high probability (no less than a threshold θ). We call this version local nucleus.

Local nucleus is a nice concept for probabilistic subgraph cohesiveness, however, it has the following shortcoming. While it ensures that every triangle $\triangle$ in $\mathcal{H}$ has support at least k in a good number of instantiations of $\mathcal{H}$, it does not ensure those instantiations are deterministic nuclei themselves or they contain some nucleus which in turn contains $\triangle$. Obviously, nucleusness is a desirable property to ask for in order to achieve a higher degree of cohesiveness and this leads to the other two versions of probabilistic nucleus of a global nature, which we call global and weakly-global (obtained for $\mu = \mathbf{g}$ and $\mu = \mathbf{w}$).

In general, $\mathbf{g}$-(k, θ)-nuclei are smaller and more cohesive than $\mathbf{w}$-(k, θ)-nuclei. We remark that, every $\mathbf{g}$-(k, θ)-nucleus is contained in a $\mathbf{w}$-(k, θ)-nucleus which in turn is contained in an $\boldsymbol{\ell}$-(k, θ)-nucleus.

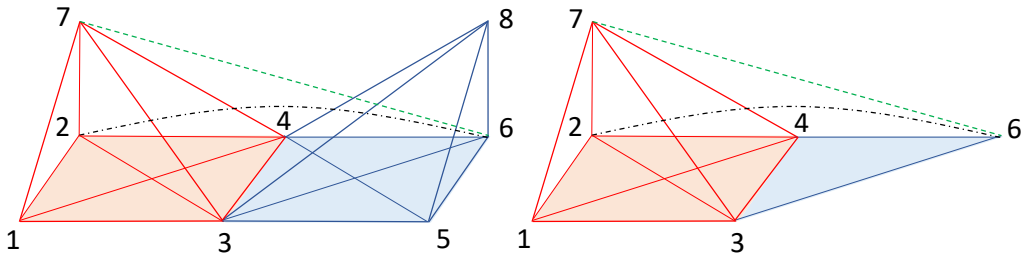


Figure 5.1: [Left] Probabilistic graph $\mathcal{G}$. Red edges have probability $P = 0.9$, blue edges have probability $P' = 0.8$, the green dashed edge has probability 1, and the black dot-dashed edge has probability $P'' = 0.5$. [Right] Subgraph $\mathcal{H}$ which is $\mathbf{w}$-$(2, 0.13)$ nucleus. $\mathcal{H}_1$ and $\mathcal{H}_2$ induced by $\{1, 2, 3, 4, 7\}$ and $\{2, 3, 4, 6, 7\}$ are $\mathbf{g}$-$(2, 0.13)$ nuclei.

Example 3. *Consider graph $\mathcal{G}$ shown in Figure 5.1 [Left]. Let us assume that the red edges have probability $P = 0.9$, the blue edges have probability $P' = 0.8$, the green dashed edge has probability 1, and the black dot-dashed edge has probability $P'' = 0.5$. Let $\theta = 0.13$. It can be verified that each triangle $\triangle$ in $\mathcal{G}$ is contained in at least 2 4-cliques with probability at least 0.134, i.e. $Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq 2) \geq 0.134 \geq \theta$. Thus, $\mathcal{G}$ is a $\boldsymbol{\ell}$-$(2, 0.13)$ nucleus.*

However, $\mathcal{G}$ cannot be a $\mathbf{w}$-$(2, 0.13)$ or $\mathbf{g}$-$(2, 0.13)$ nucleus. For instance, consider triangle $\triangle = (3, 5, 6)$. In all the possible worlds of $\mathcal{G}$, the clique on vertices $\{3, 4, 5, 6, 8\}$ should exist since this is the only deterministic 2-nucleus which contains $\triangle$. Thus, the edges of this clique (9 blue and 1 red) should exist and the other edges in $\mathcal{G}$ can either exist or not in the possible worlds of $\mathcal{G}$. As a result, we get $Pr(X_{\mathcal{G},\triangle,\mathbf{w}} \geq 2) = 0.8^9 \cdot 0.9 = 0.120 < \theta$.

Now, consider subgraph $\mathcal{H}$ induced by vertices $\{1, 2, 3, 4, 6, 7\}$, Figure 5.1 [Right]. We show that $\mathcal{H}$ is $\mathbf{w}$-$(2, 0.13)$-nucleus. Our reasoning is as follows. Ignoring probabilities, $\mathcal{H}$ consists of two deterministic 2-nuclei, one induced by $\{1, 2, 3, 4, 7\}$ (call it cl_1) and the other induced by $\{2, 3, 4, 6, 7\}$ (call it cl_2). Triangles in $\mathcal{H}$ can belong to either cl_1 or cl_2. Consider an arbitrary triangle $\triangle$ in cl_1. To compute $Pr(X_{\mathcal{H},\triangle,\mathbf{w}} \geq 2)$, all the possible worlds of $\mathcal{H}$ which contain cl_1 as a deterministic 2-nucleus are valid. As a result, all the edges in cl_1 should exist, and the edges $(2, 6)$, $(3, 6)$ and $(4, 6)$ can either exist or not exists in the valid possible worlds (edge $(7, 6)$ has probability 1). As such, we have $2^3 = 8$ valid possible worlds. Summing over the existence probability of each possible world, we get $Pr(X_{\mathcal{H},\triangle,\mathbf{w}} \geq 2) = 0.9^{10} = 0.348 > \theta$. A similar reasoning can be applied for an arbitrary triangle $\triangle'$ in cl_2 which gives $Pr(X_{\mathcal{H},\triangle',\mathbf{w}} \geq 2) = 1 \cdot 0.5 \cdot 0.8^2 \cdot 0.9^6 = 0.170 > \theta$. Thus, we can say that $\mathcal{H}$ is a $\mathbf{w}$-$(2, 0.13)$-nucleus.

Let us consider $\mathcal{H}$ in more detail. This subgraph cannot be a $\mathbf{g}$-$(2, 0.13)$-nucleus.

For instance, consider triangle $\triangle = (1, 2, 3)$*. For this triangle, there are only two valid possible worlds which are deterministic* 2*-nucleus:* (1) *the one in which all the edges exist* (H_1)*,* (2) *the one in which none of edges* $(2, 6)$*,* $(3, 6)$ *and* $(4, 6)$ *exist* (H_2)*. Adding one of the edges* $(2, 6)$*,* $(3, 6)$ *and* $(4, 6)$ *creates one extra triangle which will not be part of two cliques. This results in the possible world not being a deterministic* 2*-nucleus. So, summing over these two possible worlds we get:*

$$Pr(X_{\mathcal{H},\triangle,\mathbf{g}} \geq 2) = 0.9^{10} \cdot 1 \cdot 0.5 \cdot 0.8^2$$
$$+ 0.9^{10} \cdot 1 \cdot (1 - 0.5) \cdot (1 - 0.8)^2 = 0.118 < \theta.$$

However consider subgraphs $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *induced by* $\{1, 2, 3, 4, 7\}$ *and* $\{2, 3, 4, 6, 7\}$*, respectively. The only possible worlds of* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *that are deterministic* 2*-nuclei are the ones in which all their edges exist. So for each triangle* $\triangle$ *and* $\triangle'$ *in* $\mathcal{H}_1$ *and* $\mathcal{H}_2$*, we have* $Pr(X_{\mathcal{H}_1,\triangle,\mathbf{g}} \geq 2) = 0.9^{10} = 0.348 > \theta$ *and* $Pr(X_{\mathcal{H}_2,\triangle',\mathbf{g}} \geq 2) = 1 \cdot 0.5 \cdot 0.8^2 \cdot 0.9^6 = 0.170 > \theta$*. Thus,* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *are global* $\mathbf{g}$*-*$(2, 0.13)$*-nuclei subgraphs.*

Nucleus Decomposition. The *nucleus decomposition* finds the set of all the μ-(k, θ)-nuclei for different values of k. We will study the problem in the three different modes we consider. Specifically, we call nucleus-decomposition problems for the different modes $\boldsymbol{\ell}$-*NuDecomp*, $\mathbf{g}$-*NuDecomp*, and $\mathbf{w}$-*NuDecomp*, respectively.

We show that $\boldsymbol{\ell}$-*NuDecomp* can be computed in polynomial time and furthermore we give several algorithms to achieve efficiency for large graphs. Before this, we start by showing that $\mathbf{g}$-*NuDecomp* and $\mathbf{w}$-*NuDecomp* are $\#P$-hard and NP-hard, respectively. Nevertheless, as we show later in the dissertation, once we obtain the $\boldsymbol{\ell}$-*NuDecomp*, we can use it as basis, combined with sampling techniques, to effectively approximate $\mathbf{g}$-*NuDecomp* and $\mathbf{w}$-*NuDecomp*.

5.4 Hardness Results

In this section, we show that $\mathbf{g}$-*NuDecomp* and $\mathbf{w}$-*NuDecomp* are NP-hard. For this we use a reduction from the k-clique problem. Furthermore, we can show that $\mathbf{g}$-*NuDecomp* is even harder, namely #P-hard, using a reduction from the network reliability problem.

Definition 10. ***The*** k***-clique Problem [35].*** *Given a graph* G*, and input parameter* k*, the* k*-clique problem is to check whether there is a clique of size* k *in the graph.*

The k-clique problem is NP-complete.

We note the following interesting property about k-nucleus.

Lemma 2. For any k, the only graph on $(k+3)$ vertices which is a deterministic k-nucleus is a $(k+3)$-clique.

Proof. Recall that based on the definition of the k-nucleus, each triangle is contained in at least k 4-cliques. Given the vertices $\{v_1, v_2, \cdots, v_{k+3}\}$, without loss of generality, let $\triangle_{123} = (v_1, v_2, v_3)$ be a triangle with vertices v_1,v_2, and v_3. The triangle $\triangle_{123}$ must be part of k 4-cliques; therefore, there must be an edge between each of the remaining k vertices and all the vertices of the triangle $\triangle_{123}$. Now, new triangles are created, containing vertices $\{v_4, \cdots, v_{k+3}\}$. Let $\triangle_{ijt}$ be one of them, where $i, j = 1, 2$, $i \neq j$, and $t = 4, \cdots, k+3$. This triangle must be contained in k 4-cliques as well. Thus, there should be edges between each vertex in the triangle $\triangle_{ijt}$ to the other k vertices. Thus, each vertex v_t becomes connected to all the other vertices creating a clique on $k+3$ vertices. □

Theorem 10. **w**-*NuDecomp and* **g**-*NuDecomp are NP-hard.*

Proof. Given a graph $G = (V, E)$, we define a probabilistic graph $\mathcal{G} = (V, E, p)$ as follows: For each edge e in $\mathcal{G}$, $p(e) = \frac{1}{2^{2m}+1}$, where m is the number of the edges in $\mathcal{G}$. Let $\theta = \left(\frac{1}{2^{2m}+1}\right)^{(k+3)\cdot(k+2)/2}$.

We prove that **w**-(k,θ)-nucleus $\big($**g**-(k,θ)-nucleus$\big)$ of $\mathcal{G}$ exists if and only if a $(k+3)$-clique exists in G. Let C be a $(k+3)$-clique in G. Since C has $(k+3)\cdot(k+2)/2$ edges, its existence probability is $\left(\frac{1}{2^{2m}+1}\right)^{(k+3)\cdot(k+2)/2} = \theta$. In addition, in a $(k+3)$-clique, each triangle is contained in exactly k 4-cliques. Thus, as a subgraph, C is a both **w**-(k,θ)-nucleus and **g**-(k,θ)-nucleus of $\mathcal{G}$.

In the following we show that if G does not contain a $(k+3)$-clique, the **w**-(k,θ)-nucleus and **g**-(k,θ)-nucleus are empty. We prove the case for weakly-global, and the same reasoning can be applied for the global case as well.

Suppose that G does not contain a $(k+3)$-clique. For a contradiction, let us assume that a **w**-(k,θ)-nucleus of $\mathcal{G}$ exists and denote it by $\mathcal{F}$. Based on Lemma 2, a $(k+3)$-clique is the only graph which has $k+3$ vertices and is a deterministic k-nucleus. Since, ignoring edge probabilities, $\mathcal{F}$ cannot be a $(k+3)$-clique, it must have at least $(k+4)$ vertices. Furthermore, since it contains a k-nucleus for each triangle in it, the degree of each vertex is at least $(k+2)$.

Let $\triangle \in \mathcal{F}$ and let $\{H_1, H_2, \cdots, H_l\}$ be a set of all the valid possible worlds of $\mathcal{F}$, i.e. $\mathbb{1}_{\mathbf{W}}(H_i, \triangle, k) = 1$ ($\mathbb{1}_{\mathbf{g}}(H_i, \triangle, k) = 1$ for the proof of the global case), for all H_i (refer to Definition 8). The maximum value for l can be 2^m. For each H_i, we have:

$$\Pr(H_i) \leq p(e)^{(k+4)\cdot(k+2)/2}. \tag{5.3}$$

Therefore, for triangle $\triangle \in \mathcal{F}$, $\Pr(X_{\mathcal{F},\triangle,\mathbf{W}} \geq k)$ is at most: $\beta = l \cdot \left(\frac{1}{2^{2m}+1}\right)^{(k+4)\cdot(k+2)/2}$. Thus, we have:

$$\begin{aligned}
\beta &\leq 2^m \cdot \left(\frac{1}{2^{2m}+1}\right)^{((k+4)\cdot(k+2))/2} \\
&= 2^m \cdot \left(\frac{1}{2^{2m}+1}\right)^{((k+3)\cdot(k+2))/2} \cdot \left(\frac{1}{2^{2m}+1}\right)^{(k+2)/2} \\
&= 2^m \cdot \theta \cdot \left(\frac{1}{2^{2m}+1}\right)^{(k+2)/2} < 2^m \cdot \theta \cdot \left(\frac{1}{2^{2m}+1}\right) < \theta. \\
&< 2^m \cdot \theta \cdot \left(\frac{1}{2^{2m}+1}\right) < \theta.
\end{aligned} \tag{5.4}$$

Thus, $\mathbf{w}$-(k, θ)-nucleus is empty. □

In the following, we show that that $\mathbf{g}$-*NuDecomp* is even harder, namely #P-hard. We use a reduction from the network reliability problem.

Definition 11. ***Network Reliability [77].*** *Given a probabilistic graph $\mathcal{G} = (V, E, p)$, its* reliability *is the probability that its possible worlds are connected: $conn(\mathcal{G}) = \sum_{G \sqsubseteq \mathcal{G}} Pr[G \mid \mathcal{G}] \cdot C(G)$, where G is a possible world of $\mathcal{G}$, and $C(G)$ is an indicator function taking on 1 if the possible world G is connected and 0 otherwise.*

Computing reliability is #P-hard [77]. We can show that its decision version is also #P-hard.

Definition 12. ***Decision Version of Network Reliability.*** *Given $\mathcal{G} = (V, E, p)$ and threshold θ, is $conn(\mathcal{G}) \geq \theta$?*

Note that an algorithm for the decision version of reliability can be employed to produce a solution for the original reliability problem. This can be done by using binary search, starting with a guess, $\theta = 0.5$, and halving the search space each time, until the exact value of reliability is deduced, up to the machine supported precision. Therefore we state that

Lemma 3. *Decision version of network reliability is #P-hard.*

We now provide a reduction from this problem to **g**-*NuDecomp*. Let v be an arbitrary vertex in $\mathcal{G}$. We create two dummy vertices u and w, and the corresponding edges (u, v), (u, w), and (v, w) with probabilities $p(u, v) = 1$, $p(u, w) = 1$, and $p(v, w) = 1$, respectively. Let $\triangle = (u, v, w)$ be the resulting triangle created with existence probability of 1. Also, let the resulting graph be $\mathcal{F}$.

Lemma 4. *$\mathcal{G}$ has reliability at least θ iff $Pr(X_{\mathcal{F},\triangle,\mathbf{g}} \geq 0) \geq \theta$.*

Proof. We begin the proof with a remark. Consistent with previous works[66, 67, 64], when k is 0, we define a deterministic 0-nucleus of a graph G to be a maximal, connected subgraph H of G.

Triangle $\triangle$ appears in each possible world F of $\mathcal{F}$. Corresponding to each possible world $F = (V \cup \{u, w\},\ E_G \cup \{(u, v),\ (u, w),\ (v, w)\})$ of $\mathcal{F}$, let $G = (V, E_G)$ be the possible world of $\mathcal{G}$. We can write $F = G \cup \{\triangle\}$. If F is a 0-nucleus, F is connected and there is a path between each vertex in F, including vertices in V. Thus, G is connected as well. On the other hand, if G is connected, there is a path between vertex v and other vertices in V. Also, since u and w are connected to v with probability 1, thus, F is connected as well. Therefore, F is a 0-nucleus. Thus, connected possible worlds of $\mathcal{G}$ exactly correspond with 0-nucleus possible worlds of $\mathcal{F}$. In addition, $\Pr[F \mid \mathcal{F}] = \Pr[G \mid \mathcal{G}] \cdot \Pr(\triangle)$. Since $\Pr(\triangle) = 1$, we have $\Pr[F \mid \mathcal{F}] = \Pr[G \mid \mathcal{G}]$. Thus, we obtain $\sum_{F \sqsubseteq \mathcal{F}} \Pr[F \mid \mathcal{F}] \cdot \mathbb{1}_{\mathbf{g}}(F, \triangle, 0) = \sum_{G \sqsubseteq \mathcal{G}} \Pr[G \mid \mathcal{G}] \cdot \mathrm{C}(G)$. Using network reliability problem and the last equation, we have that $\Pr(X_{\mathcal{F},\triangle,\mathbf{g}} \geq 0) \geq \theta$ if and only if $\mathcal{G}$ has reliability with at least θ. □

Based on Lemmas 3 and 4 we finally have that

Theorem 11. **g**-*NuDecomp is #P-hard.*

5.5 Local Nucleus decomposition

Here we propose efficient algorithms for solving ℓ-*NuDecomp*. Peeling is a general strategy that has been used broadly in core and truss decompositions as well as in deterministic nucleus decomposition [66]. However, generalizing peeling to compute ℓ-*NuDecomp* creates significant computational challenges. For example, a challenge is finding the support score for each triangle. This is because of the combinatorial

nature of finding the maximum value of k such that $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k) \geq \theta$ for a triangle $\triangle$. In particular, triangle $\triangle$ in a probabilistic graph can be part of different numbers of 4 cliques with different probabilities. As a result, considering all the subsets of 4-cliques which contain $\triangle$ results in exponential time complexity. In our algorithm, we identify two challenging tasks, namely computing and updating nucleus scores.

5.5.1 Computing initial nucleus scores

Our process starts by computing a nucleus score $\kappa_\triangle$ for each triangle $\triangle$, which initially is the maximum k for which $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k) \geq \theta$.

Given a probabilistic graph $\mathcal{G} = (V, E, p)$, let $\triangle = (u, v, w)$ be a triangle in $\mathcal{G}$. For $i = 1, \ldots, c_\triangle$, where $c_\triangle = |N(u) \cap N(v) \cap N(w)|$, let $z_i \in N(u) \cap N(v) \cap N(w)$ and $S_i = \{u, v, w, z_i\}$. In other words, for each i, S_i is the set of vertices of a 4-clique that contains $\triangle$. For notational simplicity, we will also denote by S_i the 4-clique on $\{u, v, w, z_i\}$.

Similarly, for each i, let $\mathcal{E}_i = \{(u, z_i), (v, z_i), (w, z_i)\}$ be the set of edges which connect vertex z_i to vertices of $\triangle$. Let $\Pr(\mathcal{E}_i) = p(u, z_i) \cdot p(v, z_i) \cdot p(w, z_i)$ be the existence probability of $\mathcal{E}_i$. We have:

$$\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k) = \Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k-1) - \Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} = k) \tag{5.5}$$

Thus, we need to compute $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} = k)$ for any k, and find the maximum value of k for which the probability on the left-hand side of Equation 5.5 is greater than or equal to θ.

Let us denote by $\mathcal{S}_\triangle$ the set of 4-cliques containing $\triangle$. We fix an arbitrary order on $\mathcal{S}_\triangle$. Now, let $\mathcal{X}(\mathcal{S}_\triangle, k, j)$ denote the probability that $\triangle$ is contained in k of 4-cliques from $\{S_1, \cdots, S_j\}$.

The event that $\triangle$ is contained in k of 4-cliques from $\{S_1, \cdots, S_j\}$, can be expressed as the union of the following two sub-events: **(1)** the event that the 4-clique S_j exists and $\triangle$ is contained in $(k-1)$ of 4-cliques from $\{S_1, \cdots, S_{j-1}\}$, and **(2)** the event that the S_j does not exist and $\triangle$ is part of k of 4-cliques from $\{S_1, \cdots, S_{j-1}\}$. Thus, we have the following recursive formula:

$$\mathcal{X}(\mathcal{S}_\triangle, k, j) = \Pr(\mathcal{E}_j) \cdot \mathcal{X}(\mathcal{S}_\triangle, k-1, j-1) + (1 - \Pr(\mathcal{E}_j)) \cdot \mathcal{X}(\mathcal{S}_\triangle, k, j-1), \tag{5.6}$$

where $k \in [0, c_\triangle]$, and $j \in [1, c_\triangle]$. Initially we set $\mathcal{X}(\mathcal{S}_\triangle, 0, 0) = 1$, $\mathcal{X}(\mathcal{S}_\triangle, -1, k) = 0$, for any k, and $\mathcal{X}(\mathcal{S}_\triangle, k, j) = 0$, if $k > j$. Setting $j = c_\triangle$ in Equation 5.6, and multiplying $\mathcal{X}(\mathcal{S}_\triangle, k, j)$ by $\Pr(\triangle)$ (existence probability of $\triangle$), gives the desired probability $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} = k)$. The above recursive formula forms the basis for a dynamic programming approach to computing the initial values of κ scores.

Given a triangle $\triangle$, let the *neighbor triangles* of $\triangle$ be those triangles which form a 4-clique with $\triangle$. In the following we show how we can update $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k)$ when a neighbor triangle is processed in the decomposition.

5.5.2 Updating nucleus scores

Once the κ scores have been initialized as described above, a process of peeling "removes" the triangle $\triangle^*$ of the lowest κ-score, specifically marks it as *processed*, and updates the neighboring triangles $\triangle$ (those contained in the same 4-cliques as the removed triangle) in terms of $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k)$. Because of the removal of $\triangle^*$ the cliques containing it cease to exist, thus $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k)$ of the neighbors $\triangle$ will change. We recompute this probability using the formula in Equation 5.6, where the sets of cliques $\mathcal{S}_\triangle$ are updated to remove the cliques containing $\triangle^*$.

Algorithm 10 *$\boldsymbol{\ell}$-NuDecomp*

```
1:  function ℓ-NucleusNess(G, θ)
2:      for all triangles △ ∈ G do
3:          κ(△) ← arg max_k{X(S_△, k, c_△) ≥ θ}
4:          processed[△] ← false
5:      for all unprocessed △ ∈ G with minimum κ(△) do
6:          ν(△) ← κ(△)
7:          Find set S_△ of 4-cliques containing △
8:          for all S ∈ S_△ with non-processed triangles do
9:              for all △' ⊂ S, △' ≠ △, κ(△') > κ(△) do
10:                 κ(△') ← arg max_k{X(S_△' \ S, k, c_△' − 1) ≥ θ}
11:         processed[△] ← true
12:     return array ν(·)
```

Algorithm 10 computes the nucleusness of each triangle in $\mathcal{G}$. In line 3, for each triangle $\triangle$, $\kappa(\triangle)$ is initialized using Equation 5.6. Array *processed* records whether a triangle has been processed or not in the algorithm (line 4). At each iteration (line 5-11), an unprocessed triangle $\triangle$ with minimum $\kappa(\triangle)$ is considered, and its

nucleus score is set and stored in array ν (line 6). Then, the $\kappa(\triangle')$ values of all the neighboring triangles $\triangle'$ are updated using Equation 5.6. The affected triangles are those unprocessed triangles which are part of the same 4-clique with triangle $\triangle$. The algorithm continues until all the triangles are processed. At the end, each triangle obtains its nucleus score and array ν with these scores is returned (line 12). Once all the nucleus scores are obtained, we build $\boldsymbol{\ell}$-(k, θ)-nuclei for each value of k.

Observe that the κ values for each triangle at each iteration decrease or stay the same. This implies that κ for each triangle $\triangle$ is a monotonic property function similar to properties described in [4] for vertices. Now, we can use a reasoning similar to the one in [4] to show that our algorithm, which repeatedly removes a triangle with the smallest κ value, gives the correct nucleusness for each triangle.

Time complexity: Using dynamic programming, Lines 2-3 take $O\left(\sum_{\triangle \in \mathcal{G}} \kappa_{\triangle} \cdot c_{\triangle}\right)$, where $\kappa_{\triangle}$ is the nucleusness obtained for each triangle $\triangle$ in line 3. Let $\kappa_{\max}$ be the maximum $\kappa_{\triangle}$ over all the triangles in $\mathcal{G}$. Since $c_{\triangle} \in O(d(u) + d(v) + d(w)) \subseteq O(d_{\max})$, running time of line 3 is $O\left(\sum_{\triangle \in \mathcal{G}} \kappa_{\triangle} \cdot c_{\triangle}\right) = O\left(\sum_{\triangle \in \mathcal{G}} \kappa_{\max} \cdot d_{\max}\right) = O\left(\kappa_{\max} d_{\max} T_{\mathcal{G}}\right)$, where $T_{\mathcal{G}}$ is the total number of triangles in the graph, and $d_{\max}$ is the maximum degree in $\mathcal{G}$. For each triangle $\triangle$, finding all $\mathcal{S}_{\triangle}$'s in line 7, takes $O\left(d(u) + d(v) + d(w)\right) = O(d_{\max})$. In addition, lines 9-10 take $O\left(\sum_{\triangle' \in N(\triangle)} (\kappa_{\triangle'} \cdot c_{\triangle'})\right)$ time, where $N(\triangle)$ is the triangles which form a 4-clique with $\triangle$. Note that $N(\triangle) = O(c_{\triangle})$. Therefore, the running time for processing all the triangles is

$$
\begin{aligned}
& O\left(\sum_{\triangle \in \mathcal{G}} \left((d(u) + d(v) + d(w)) + \sum_{\triangle' \in N(\triangle)} \kappa_{\triangle'} \cdot c_{\triangle'} \right) \right), \\
& = O\left(\sum_{\triangle \in \mathcal{G}} \left(d_{\max} + (\kappa_{\max} \sum_{\triangle' \in N(\triangle)} d_{\max}) \right) \right), \\
& = O\left(\sum_{\triangle \in \mathcal{G}} \left(d_{\max} + \kappa_{\max} \cdot d_{\max}^2 \right) \right) = O\left(\kappa_{\max} d_{\max}^2 T_{\mathcal{G}}\right),
\end{aligned}
$$

Thus, the total running time of Algorithm 10 is bounded by $O\left(\kappa_{\max} d_{\max}^2 T_{\mathcal{G}}\right)$, and we can state the following.

Theorem 12. *$\boldsymbol{\ell}$-NuDecomp can be computed in polynomial time.*

The space complexity is $O(T_{\mathcal{G}})$. This space is needed to store triangles (not 4-

cliques) and their κ values. This is the same as the space complexity of deterministic nucleus decomposition.

While being able to compute ℓ-*NuDecomp* in polynomial time is good news, finding the maximum k such that $\Pr(X_{\mathcal{G},\triangle,l} \geq k) \geq \theta$ is quadratic in $c_\triangle$ which is not efficient for large probabilistic graphs. As an alternative approach, we will now propose efficient methods to approximate $\Pr(X_{\mathcal{G},\triangle,l} \geq k)$ in $O(c_\triangle)$ time such that the results are practically distinguishable from the exact values. The approximation is based on limit theorems, such as Le Cam's Poisson Limit Theorem [45] and Lyapunov's Central Limit Theorem [51].

5.5.3 Approximating κ scores

Framework. Given a triangle $\triangle = (u, v, w)$, let $S_i = \{u, v, w, z_i\}$ for $i = 1, \ldots, c_\triangle$, as before. Also, let $\mathcal{E}_i = \{(u, z_i), (v, z_i), (w, z_i)\}$ be the edges that connect z_i to the vertices of $\triangle$.

With slight abuse of notation, we also define each $\mathcal{E}_i$ as an indicator random variable which takes on 1, if all the edges in $\mathcal{E}_i$ exist, and takes on 0, if at least one of the edges in the set does not exist. We observe that the variables $\mathcal{E}_i$ are mutually independent since the sets $\mathcal{E}_i$ do not share any edge. Also, each Bernoulli variable $\mathcal{E}_i$ takes value 1 with probability $p(u, z_i) \cdot p(v, z_i) \cdot p(w, z_i)$ and 0 with $1 - (p(u, z_i) \cdot p(v, z_i)) \cdot p(w, z_i))$.

Let $\zeta = \sum_{i=1}^{c_\triangle} \mathcal{E}_i$. We can verify the following proposition.

Proposition 4. *$Pr(X_{\mathcal{G},\triangle,\ell} \geq k) = Pr(\triangle) \cdot Pr[\zeta \geq k]$.*

The expectation and variance of ζ are $\mu = \sum_{i=1}^{c_\triangle} \Pr(\mathcal{E}_i)$ and $\sigma^2 = \sum_{i=1}^{c_\triangle} \big(\Pr(\mathcal{E}_i) \cdot (1 - \Pr(\mathcal{E}_i)\big)$, respectively. Now we show that we can approximate the distribution of ζ using Le Cam's Theorem which makes use of Poisson Distribution [45].

Poisson Distribution [31]: A discrete random variable X is said to have Poisson distribution with positive parameter λ, if the probability mass function of X is given by:

$$\Pr[X = k] = \frac{\lambda^k e^{-k}}{k!}, \quad k = 0, 1, \cdots, \tag{5.7}$$

The expected value of a Poisson random variable is λ. Setting λ to μ, we can approximate the distribution of ζ by the Poisson distribution. Using Le Cam's Theorem

[45], the error bound on the approximation is as follows:

$$\sum_{k=0}^{c_\triangle} \left| \Pr(\zeta = k) - \frac{\lambda^k e^{-\lambda}}{k!} \right| < 2 \sum_{i=1}^{c_\triangle} \big(\Pr(\mathcal{E}_i)\big)^2 = 2(\mu - \sigma^2). \tag{5.8}$$

Equation 5.8 shows that the Poisson distribution is reliable if $\Pr(\mathcal{E}_i)$ and $c_\triangle$ are small.

We observe that computing tail probabilities for the Poisson distribution is easy in practice as these probabilities satisfy a simple recursive relation.

$$\begin{aligned} \Pr[\zeta < k] &\approx \sum_{j<k} \frac{e^{-\lambda}\lambda^j}{j!} = \sum_{j<k-1} \frac{e^{-\lambda}\lambda^j}{j!} + \frac{e^{-\lambda}\lambda^{k-1}}{(k-1)!} \\ &= \Pr[\zeta < k-1] + \frac{\lambda}{k-1}\Pr[\zeta = k-2] \end{aligned} \tag{5.9}$$

with base case $\Pr[\zeta < 1] = \Pr[\zeta = 0] = e^{-\lambda}$. Using Equation 5.9, and iterating over all values of k from 1 to $c_\triangle$, we can evaluate each term $\Pr[\zeta \geq k] = 1 - \Pr[\zeta < k]$ in constant time, and find the maximum k such that $\Pr(\triangle) \cdot \Pr[\zeta \geq k] \geq \theta$. Thus, the time complexity of obtaining $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k)$ is $O(c_\triangle)$.

In some applications, $\sum_{i=1}^{c_\triangle} \big(\Pr(\mathcal{E}_i)\big)^2$ in Equation 5.8 can be large, even if each $\Pr(\mathcal{E}_i)$ is small. As a result, the difference between the variance $\sigma^2 = \sum_{i=1}^{c_\triangle} \Pr(\mathcal{E}_i) - \sum_{i=1}^{c_\triangle} (\Pr(\mathcal{E}_i))^2$ of ζ, and the variance $\lambda = \sum_{i=1}^{c_\triangle} \Pr(\mathcal{E}_i)$ of the Poisson approximation becomes large. To tackle the problem, we define a Translated Poisson [62] random variable $Y = \lfloor \lambda_2 \rfloor + \Pi_{\lambda - \lfloor \lambda_2 \rfloor}$, where $\lambda_2 = \lambda - \sigma^2$ and Π is Poisson distribution with parameter $\lambda - \lfloor \lambda_2 \rfloor$. In this formula $\lambda = \sum_{i=1}^{c_\triangle} \Pr(\mathcal{E}_i)$ is the expected value of distribution ζ. Thus, the difference between the variance of Y and ζ can be written as:

$$\begin{aligned} &\mathrm{Var(Y)} - \mathrm{Var}(\zeta) = \lambda - \lfloor \lambda_2 \rfloor - \sigma^2 = \lambda - \sigma^2 - \lfloor \lambda_2 \rfloor, \\ &= \lambda_2 - (\lambda_2 - \{\lambda_2\}) = \{\lambda_2\} < 1, \end{aligned} \tag{5.10}$$

where $\{\lambda_2\} = \lambda_2 - \lfloor \lambda_2 \rfloor$. As can be seen the difference between the variances becomes small in this case.

Equation 5.9 for Translated Poisson changes to

$$\begin{aligned}
&\Pr[\zeta < k] \approx \Pr[Y < k] = \Pr\Big[\Big(\lfloor\lambda_2\rfloor + \Pi_{\lambda-\lfloor\lambda_2\rfloor}\Big) < k\Big], \\
&= \Pr[\Pi_{\lambda-\lfloor\lambda_2\rfloor} < k - \lfloor\lambda_2\rfloor] = \Pr[\Pi_{\lambda-\lfloor\lambda_2\rfloor} < k - \lfloor\lambda_2\rfloor - 1], \\
&+ \frac{\lambda - \lfloor\lambda_2\rfloor}{k - \lfloor\lambda_2\rfloor - 1}\Pr[\Pi_{\lambda-\lfloor\lambda_2\rfloor} = k - \lfloor\lambda_2\rfloor - 2], \qquad (5.11)
\end{aligned}$$

and the complexity of obtaining $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k)$ remains the same.

We will now consider the scenario when $c_\triangle$ is large. In this case, the variance of ζ will be large. In the following, we show the use of Central Limit Theorem for this case.

Central Limit Theorem. An important theorem in statistics, Lyapunov's Central Limit Theorem (CLT) [51] states that, given a set of random variables (not necessarily i.i.d.), their properly scaled sum converges to a normal distribution under certain conditions (please see the chapters on core and truss decomposition for the formulation of this theorem).

If $c_\triangle$ and hence σ^2 are large, then by [51], $Z = \frac{1}{\sigma}\sum_{i=1}^{c_\triangle}(\mathcal{E}_i - \mu_i)$ has standard normal distribution, where $\mu_i = \Pr(\mathcal{E}_i)$. To approximate $\Pr[\zeta \geq k] = \Pr[\sum_{i=1}^{c_\triangle}\mathcal{E}_i \geq k]$ using CLT we can subtract $\sum_{i=1}^{c_\triangle}\mu_i$ from the sum of $\mathcal{E}_i$'s and divide by σ. As a result, we have:

$$\Pr\left[\sum_{i=1}^{c_\triangle}\mathcal{E}_i \geq k\right] = \Pr\left[\frac{1}{\sigma}\sum_{i=1}^{c_\triangle}(\mathcal{E}_i - \mu_i) \geq \frac{1}{\sigma}\left(k - \sum_{i=1}^{c_\triangle}\mu_i\right)\right] \qquad (5.12)$$

Since $Z = \frac{1}{\sigma}\sum_{i=1}^{c_\triangle}(\mathcal{E}_i - \mu_i)$ has standard normal distribution, we can find the maximum value of k such that the right-hand side of Equation 5.12 is at equal or greater than the threshold. Evaluation of each probability can be done in constant time. Thus, finding the maximum value of k can be done in $O(c_\triangle)$ time.

Binomial Distribution. In many networks, edge probabilities are close to each other and as a result, for each triangle $\triangle$, $\Pr(\mathcal{E}_i)$'s are also close to each other. In that case, the distribution of support of the triangle $\triangle$ can be well approximated by Binomial distribution. A random variable X is said to have Binomial distribution with parameters p and n, if the probability mass function of X is given by [56]:

$$\Pr[X = k] = \binom{n}{k}p^k(1-p)^{(n-k)}. \qquad (5.13)$$

In the above equation, p is success probability, and n is the number of experiments. In statistics, the sum of non-identically distributed and independent Bernoulli random variables can be approximated by the Binomial distribution [22]. As discussed in [22], the Binomial distribution provides a good approximation, if its variance is close to the variance of ζ. For the approximation, we set $n = c_{\triangle}$ and $n \cdot p = \mu$.

We observe that tail probabilities for the Binomial distribution can be calculated inexpensively as these probabilities satisfy the following well-known recursive relation

$$\Pr[\zeta = k] = \frac{(n-k+1)p}{k(1-p)} \Pr[\zeta = k-1]. \tag{5.14}$$

Using Equation 5.14, and iterating over values of k from 1 up to $c_{\triangle}$, we can evaluate $\Pr[\zeta \geq k]$ in $O(1)$ time, and find the maximum k such that $\Pr(\triangle) \cdot \Pr[\zeta \geq k] \geq \theta$. Thus, the time complexity of obtaining probabilistic support for a triangle $\triangle$ in this case is $O(c_{\triangle})$.

Summary. We compute $\Pr(X_{\mathcal{G},\triangle,\ell} \geq k)$ using the following set of conditions based on four thresholds A, B, C, D.

1. If $c_{\triangle}$ is large ($c_{\triangle} \geq A$), the *CLT* approximation is used.
2. If (1) does not hold, then if $c_{\triangle}$ and $\Pr(\mathcal{E}_i)$'s are small ($c_{\triangle} < B$ and $\Pr(\mathcal{E}_i)'s < C$), the *Poisson* approximation is used.
3. If (1) and (2) do not hold, then if $\sum_{i=1}^{c_{\triangle}} \left(\Pr(\mathcal{E}_i)\right)^2 > 1$, the *Translated Poisson* approximation is used.
4. If (1), (2), and (3) do not hold, then if the ratio of the variance of ζ to the variance of the Binomial distribution with $n = c_{\triangle}$ and $p = \mu/n$ is close to 1 (e.g. not less than a number D), the *Binomial* approximation is used.
5. Otherwise, *Dynamic Programming* is used.

For selecting the thresholds we refer to the literature in statistics. In particular, CLT gives a good approximation if the number (for our problem $c_{\triangle}$) of random variables in the sum is at least 30 ([55], p. 547). In fact, we set our threshold $A = 200$ to much higher than what is suggested by the literature. Also, regarding Poisson distribution, the existence probability (for our problem $\Pr(\mathcal{E}_i)$'s) of the indicator random variables in the sum should be less than 0.25 (see [45]). So, we set $C = 0.25$. We

set B to be half of A so that it is considerably far from A (threshold on $c_{\triangle}$). We set $D = 0.9$ which is close enough to 1.

When using $A = 200$, $B = 100$, $C = 0.25$, $D = 0.9$, we observed that the results of computing $\Pr(X_{\mathcal{G},\triangle,\boldsymbol{\ell}} \geq k)$ using an approximation are practically indistinguishable from the solution of dynamic programming. Furthermore, as we observed in our experiments, falling back to dynamic programming in point (5) happens only in a small amount of cases, i.e. most triangles in real world networks satisfy one of the earlier conditions (1)-(4). This means we can avoid dynamic programming for most of the triangles.

5.6 Global and Weakly-Global Nucleus Decomposition

In this section, we propose algorithms for computing global and weakly-global nucleus decomposition. Given a graph $\mathcal{F}$, computing $\Pr(X_{\mathcal{F},\triangle,\mathbf{g}} \geq k)$ and $\Pr(X_{\mathcal{F},\triangle,\mathbf{w}} \geq k)$ requires finding all the possible worlds of $\mathcal{F}$, which are in total $2^{|E(\mathcal{F})|}$, where $E(\mathcal{F})$ is the number of edges in $\mathcal{F}$. This is prohibitive. Thus, we use Monte Carlo sampling to estimate the probabilities, denoted by $\hat{\Pr}(X_{\mathcal{F},\triangle,\mathbf{g}} \geq k)$ and $\hat{\Pr}(X_{\mathcal{F},\triangle,\mathbf{w}} \geq k)$. The following lemma states a special version of the Hoeffdings inequality [32] that provides the minimum number of samples required to obtain an unbiased estimate.

Lemma 5. *Let $Y_1, \cdots, Y_n$ be independent random variables bounded in the interval $[0, 1]$. Also, let $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$. Then, we have that*

$$Pr\left[|\bar{Y} - \mathbb{E}[\bar{Y}]| \geq \epsilon\right] \leq 2e^{-2n\epsilon^2}. \tag{5.15}$$

In other words, for any $\epsilon, \delta \in (0, 1]$, $Pr\left[|\bar{Y} - \mathbb{E}[\bar{Y}]| \geq \epsilon\right] \leq \delta$, if $n \geq \left\lceil \frac{1}{2\epsilon^2} \ln\left(\frac{2}{\delta}\right)\right\rceil$.

Based on the above, using Monte Carlo sampling, we can obtain an estimate of $\Pr(X_{\mathcal{F},\triangle,\mathbf{g}} \geq k)$, and $\Pr(X_{\mathcal{F},\triangle,\mathbf{w}} \geq k)$ for any subgraph $\mathcal{F}$ by sampling n possible worlds of $\mathcal{F}$, $\{H_1, \cdots, H_n\}$, where $n = \left\lceil \frac{1}{2\epsilon^2} \ln\left(\frac{2}{\delta}\right)\right\rceil$, ϵ is an error bound, and δ is a probability guarantee. In particular, we have:

$$\hat{\Pr}(X_{\mathcal{F},\triangle,\mu} \geq k) = \sum_{i=1}^{n} \mathbb{1}_{\mu}(H_i, \triangle, k)/n, \tag{5.16}$$

Algorithm 11 **g**-*NuDecomp*

```
1: function g_Nucleus(G, θ, ϵ, δ)
2:     solution ← {}
3:     for all k ← 1 to k_max do
4:         C_k ← the union of ℓ-(k, θ)-nuclei by Algorithm 10
5:         for all △ ∈ C_k do
6:             F ← all 4-cliques in C_k containing △
7:             while ∃△′ ∈ F with less than k 4-cliques ∈ F containing it do
8:                 add all 4-cliques of C_k containing △′ to F
9:             condition_hold ← true
10:            sample ← {H_1, ··· , H_n}
11:            for all △ ∈ F do P̂r(X_{F,△,g} ≥ k) ← Eq.(5.16)
12:                if P̂r(X_{F,△,g} ≥ k) < θ then
13:                    condition_hold ← false
14:                    break
15:            if condition_hold == true then
16:                solution ← solution ∪ F
17:    return solution
```

where $\mu = \mathbf{g}$ or $\mathbf{w}$, and the indicator function $\mathbb{1}_\mu(H_i, \triangle, k)$ is given in Definition 8. Based on Lemma 5, what we obtain is an unbiased estimate. Thus, setting $\mu = \mathbf{g}, \mathbf{w}$, we have

$$\Pr\left[\left|\Pr\left(X_{\mathcal{F},\triangle,\mu} \geq k\right) - \hat{\Pr}\left(X_{\mathcal{F},\triangle,\mu} \geq k\right)\right| \geq \epsilon\right] \leq \delta. \tag{5.17}$$

g-(k,θ)-nucleus. In what follows, we propose an algorithm for finding all **g**-(k,θ)-nuclei for different values of $k = 1, \ldots, k_{\max}$, where $k_{\max}$ is the largest value for which the local nucleus is non-empty. This is because we extract global nuclei from local ones since every **g**-(k,θ)-nucleus is part of an $\boldsymbol{\ell}$-(k,θ)-nucleus. The main steps of our proposed algorithm are given in Algorithm 11.

Given a positive integer k, threshold θ, error-bound ϵ, and confidence level δ, the algorithm starts by creating subgraph $\mathcal{C}_k$ as the union of all $\boldsymbol{\ell}$-(k,θ)-nuclei (line 4). Then, the algorithm incrementally builds a candidate **g**-(k,θ)-nucleus $\mathcal{F}$ as follows. For each triangle $\triangle$ in $\mathcal{C}_k$, it adds to $\mathcal{F}$ all the 4-cliques in $\mathcal{C}_k$ containing $\triangle$ (line 6). By this process other triangles $\triangle'$ could potentially be added to $\mathcal{F}$ such that the number of 4-cliques containing $\triangle'$ is less than k. In order to remedy this, the algorithm adds all the 4-cliques of $\mathcal{C}_k$ containing $\triangle'$ to $\mathcal{F}$. This process continues until all the triangles in $\mathcal{F}$ are contained in at least k 4-cliques (lines 7-8). Once the candidate graph $\mathcal{F}$

is obtained, n samples of possible worlds of $\mathcal{F}$ are obtained (line 10). Then, the algorithm checks if the condition $\hat{\Pr}(X_{\mathcal{F},\triangle,\mathbf{g}} \geq k) \geq \theta$ is satisfied for each triangle $\triangle$ in $\mathcal{F}$ (lines 11-13). At the end, the algorithm returns all $\mathbf{g}$-(k,θ)-nuclei $\mathcal{F}$ (line 15-17), for all the possible values of k.

Algorithm 12 $\mathbf{w}$-*NuDecomp*

```
1: function w_NUCLEUS(G, θ, ϵ, δ)
2:     solution ← {}
3:     for all k ← 1 to k_max do
4:         for all ℓ-(k, θ) F do
5:             global_score[△] ← 0 for each △ ∈ F
6:             sample ← {H_1, ··· , H_n}
7:             for all H ∈ sample do
8:                 H' ← k-nucleus of H
9:                 for all triangle △ ∈ H' do
10:                    global_score[△] ++
11:            for all △ ∈ F do
12:                Pr̂(X_{H,△,w} ≥ k) ← global_score[△]/n
13:            solution ← solution ∪ connected union of △'s
               with Pr̂(X_{F,△,w} ≥ k) ≥ θ
14:    return solution
```

$\mathbf{w}$-(k,θ)-nucleus. In what follows, we propose an algorithm for finding all w-(k,θ)-nuclei, for different values of $k = 1, \ldots, k_{\max}$, where $k_{\max}$ is as before. We begin by noting that each $\mathbf{w}$-(k,θ)-nucleus is an $\boldsymbol{\ell}$-(k,θ)-nucleus. The steps of our proposed algorithm are given in Algorithm 12.

We use array *global_score* to store the number of deterministic k-nuclei that each triangle belongs to. The array is initialized to zero for all the triangles in the candidate graph (line 5). For each candidate graph which is a $\boldsymbol{\ell}$-(k,θ)-nucleus, we obtain the required number n of possible worlds for the given ϵ and δ. Then, we perform deterministic nucleus decomposition on each world (lines 6-8). If triangle $\triangle$ is in a deterministic k-nucleus of that possible world, the corresponding index of $\triangle$ in array *global_score* is incremented by one (lines 9-10). In line 12, the approximate value $\hat{\Pr}(X_{\mathcal{F},\triangle,\mathbf{w}} \geq k)$ is obtained for each triangle. Then, we start creating the connected components $\mathcal{F}$ using triangles with $\hat{\Pr}(X_{\mathcal{F},\triangle,\mathbf{w}} \geq k) \geq \theta$ (line 13). At the end, the algorithm returns all $\mathbf{w}$-(k,θ)-nuclei, for all the possible values of k.

Remark. Both of these algorithms run in polynomial time. They compute the correct answer provided the estimation of the threshold probabilities using Monte-Carlo

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\mathrm{d}_{\max}$	p_{avg}	$\lvert\triangle\rvert$
krogan	2,708	7,123	141	0.68	6,968
dblp	684,911	2,284,991	611	0.26	4,582,169
flickr	24,125	300,836	546	0.13	8,857,038
pokec	1,632,803	22,301,964	14,854	0.50	32,557,458
biomine	1,008,201	6,722,503	139,624	0.27	93,716,868
ljournal-2008	5,363,260	49,514,271	19,432	0.50	411,155,444

Table 5.1: Dataset Statistics

sampling is close to the true value. If not, they give approximate solutions.

Space Complexity. For global and weakly global decompositions the space needed is $O(T_{\mathcal{G}} + m \cdot n)$, where m is the number of edges in $\mathcal{F}$ and n is the number of possible worlds for $\mathcal{F}$ we sample.

From a theoretical point of view, n, the number of samples, is constant for fixed values of ϵ and δ, and since m, number of edges, is absorbed by $T_{\mathcal{G}}$, we can say that the above complexity is again $O(T_{\mathcal{G}})$, i.e. same as the space complexity for deterministic nucleus.

From a practical point of view, for each sample graph (possible world) we use a bit array to record whether an edge exists in the sample or not. For practical values of ϵ and δ, $m \cdot n$ is about $200 \cdot m$ bits, which is $200/(32+32) \sim 3$ times more than the space needed to store the edges as adjacency lists (assuming an integer node id is 32 bits, and the graph is undirected, i.e. each edge is represented as two directed edges). In other words, to store the n possible worlds we only need about three times more space than what is needed to store $\mathcal{G}$.

5.7 Experiments

We present our extensive experimental results to test the efficiency, effectiveness, and accuracy of our proposed algorithms. Our implementations are in Java and the experiments are conducted on a commodity machine with Intel i7, 2.2Ghz CPU, and 12Gb RAM, running Ubuntu 18.04.

Datasets and Experimental Framework. Statistics for our datasets are in Table 5.1. We order the datasets based on the number of triangles they contain. Datasets with real probabilities are *flickr*, *dblp*, and *biomine* from [9, 61] and *krogan* from [43].

We also consider datasets *ljournal-2008* and *pokec*. *ljournal-2008* is obtained from

Laboratory of Web Algorithmics [1] and *pokec* is from the Stanford Network Analysis Project [2]. For these networks, we generated edge probabilities uniformly distributed in $(0, 1]$.

We evaluate our algorithms on three important aspects. First is the efficiency. For this, we report the running time of our algorithms in Subsection 5.7.1. Second is the accuracy and closeness of our approximation methods. We discuss this in Subsection 5.7.2. Third is the quality of the output nucleus as measured by *density* and *probabilistic clustering coefficient* which are discussed in Subsection 5.7.3. Finally, in Subsection 5.7.4 we show the usefulness of our nucleus definitions over probabilistic graphs by presenting a detailed use-case.

5.7.1 Efficiency Evaluation

In this section, we report the running times of our proposed algorithms for local nucleus decomposition: one that uses dynamic programming and the other that uses statistical approximations for computing and updating the support of triangles. We denote them by DP and AP, respectively. Next, we report the running times of our (fully) global and weakly-global nucleus decomposition algorithms, which we denote by FG and WG. We set error-bound $\epsilon = 0.1$ and confidence level $\delta = 0.1$. Based on these values and Lemma 5, we set the number of samples to $n = 200 > \lceil \frac{1}{2\epsilon^2} \ln \left(\frac{2}{\delta}\right) \rceil$ (i.e. greater than what is required by Hoeffding's inequality). As such, our results for global and weakly-global notions of nuclueus are approximate but come with strong quality guarantees.

Running time results for DP and AP are shown in Figure 5.2 for different values of θ. Y-axis for the bottom plots is in log-scale.

Both algorithms perform well on medium-size datasets, *dblp* and *flickr*; computing the nucleus decomposition of these two graphs takes less than 1 sec. For a larger-size dataset, *pokec*, both algorithms complete in less than 10 min. Note that AP clearly outperforms DP on large-size datasets such as *biomine* and *ljournal-2008* for small values of θ. For instance, for *ljournal-2008* with $\theta = 0.1$, it is only AP that can run to completion, whereas DP could not complete after one day. Nevertheless, both DP and AP are able to run in reasonable time for all the other cases, which is good considering that nucleus decomposition is a harder problem than core and truss decomposition.

[1] `http://law.di.unimi.it/datasets.php`

[2] `http://snap.stanford.edu`

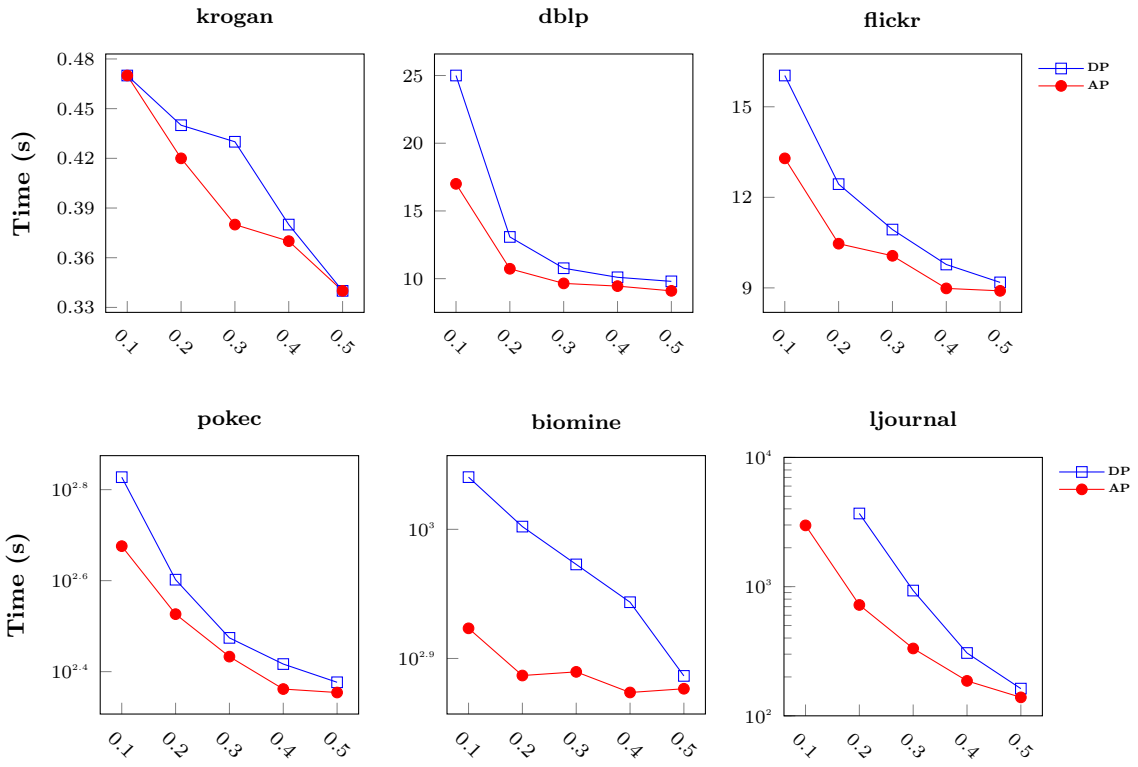


Figure 5.2: Run times of DP and AP for varying θ (x axis). Both perform well on medium datasets. For bigger datasets, biomine and ljournal, the difference is more pronounced. For ljournal, for $\theta = 0.1$, it is only AP that can complete within one day.

In general, the running times of both DP and AP decrease significantly as θ increases. This is because the number of triangles which, (a) exist with probability greater than θ and (b) have a support at least k again with probability greater than θ, decreases. As can be seen, AP is faster than DP on all datasets for different values of θ. In addition to the *ljournal-2008* case for which only AP could complete, when $\theta = 0.1$, the gain of AP over DP is about 24% and 30% for *biomine* and *pokec*, respectively.

We report the running time of FG and WG in Figure 5.3 along with the running time of local (denoted by L in the figure) nucleus decomposition for $\theta = 0.1$ (which as explained above is more difficult than $\theta = 0.2, \ldots, 0.5$). Note that the global and weakly-global nuclei are obtained from the local ones using Algorithms 11 and 12. Therefore, their running time includes the time required for obtaining local nuclei. For local decomposition, we use DP to obtain the probabilistic support of the triangles, except for *ljournal-2008* for which we use AP since DP does not scale for this

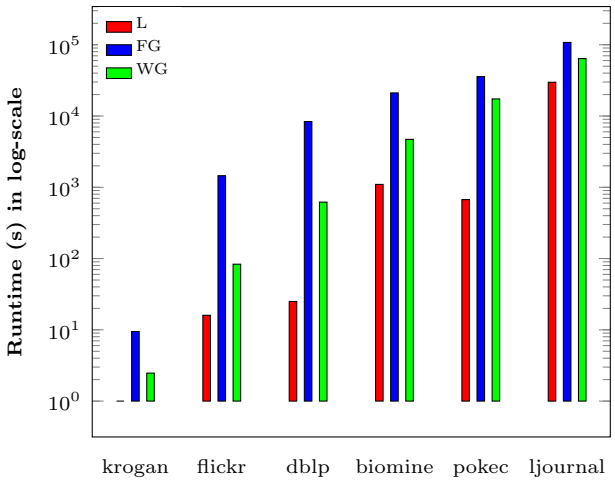


Figure 5.3: Run time of L, FG, and WG. FG and WG include the time for L. WG is faster because it performs deterministic decomposition only on a fixed number of sample graphs while FG does so each time a candidate graph is discovered.

threshold.

In general WG is faster than FG. This is because WG performs deterministic nucleus decomposition only on a fixed number of sample graphs while FG does the decomposition every time that a candidate graph is detected. We also note that as the graph becomes larger, WG will have to perform nucleus decomposition on larger sample graphs leading to increased running time. For FG, usually candidate graphs are small even for large graphs. So, when the graph becomes lager, the runtime of WG increases more compared to FG.

5.7.2 Accuracy Evaluation

To evaluate the accuracy of the AP algorithm, we compare the final nucleus scores obtained by DP and AP algorithms. We report the results in Table 5.2. We show the results for θ equal to 0.2 and 0.4, since for the remaining values the error results do not differ significantly. The second column shows the average difference (error) from true value over the total number of triangles. The last column shows the percentage of triangles whose value is different from their exact value.

As can be seen, the average error is quite small for all the datasets we consider. Particularly, for *flickr* with $\theta = 0.4$ and *biomine* with $\theta = 0.2$ and $\theta = 0.4$ we have that AP computes nucleus decomposition with *zero error*. Also, the percentage of triangles with an error score is very small, namely less than 1% for all the datasets,

Dataset	Avg Error	% of $\triangle$ with Error
	$\theta = 0.2/\theta = 0.4$	
krogan	0.0524/0.0209	5.24/2.08
dblp	0.0069/0.0041	0.69/ 0.41
flickr	0.0031/0.0	0.31/0.0
pokec	0.0014/4.15e-5	0.14/0.004
biomine	0.0/0.0	0.0/0.0
ljournal-2008	0.0179/0.0070	1.79/0.69

Table 5.2: Avg difference (error) of AP scores from true DP scores and percentage of triangles with error. Errors are very small.

except *krogan* and *ljournal-2008*. For these two, the percentages are still small, 5.24% and 1.79%, respectively. These results show that the output of AP is very close to that of exact computation by DP, and thus, AP is a reliable approximation methodology.

5.7.3 Quality Evaluation of Nucleus Subgraphs

Here we compare the cohesiveness of ℓ-(k,θ)-nucleus with the cohesiveness of local (k,γ)-truss [34] and (k,η)-core [9]. We use two metrics. The first metric is the probabilistic density (PD) of a graph $\mathcal{G}$, which we denote by $\mathrm{PD}(\mathcal{G})$ and is defined as follows [34]:

$$\mathrm{PD}(\mathcal{G}) = \frac{\sum_{e\in E} p(e)}{\frac{1}{2}|V|\cdot(|V|-1)}. \tag{5.18}$$

In words, PD of a probabilistic graph is the ratio of the sum of edge probabilities to the possible number of edges in a graph.

The second metric is probabilistic clustering coefficient (PCC). It measures the level of tendency of the nodes to cluster together. Given a probabilistic graph $\mathcal{G}$, its PCC is defined as follows [34, 59]:

$$\mathrm{PCC}(\mathcal{G}) = \frac{3\sum_{\triangle_{uvw}\in\mathcal{G}} p(u,v)\cdot p(v,w)\cdot p(u,w)}{\sum_{(u,v),(u,w),v\neq w} p(u,v)\cdot p(u,w)}. \tag{5.19}$$

For probabilistic nucleus, probabilistic truss and probabilistic core subgraphs, we use the same threshold $\theta = \gamma = \eta$, set to 0.1 and 0.3. (γ is used as threshold in the truss case [34], and η is used as threshold in the core case [9]). Table 5.3 reports results on *dblp*, *pokec*, and *biomine*. Results for the other datasets are similar. For a given threshold, we report the statistics of local (k_{Nmax},θ)-nucleus, (k_{Tmax},γ)-truss,

Graph	θ	$\lvert V_N\rvert / \lvert V_T\rvert / \lvert V_C\rvert$	$\lvert E_N\rvert / \lvert E_T\rvert / \lvert E_C\rvert$	$k_{Nmax}/k_{Tmax}/k_{Cmax}$	$\text{PD}_N/\text{PD}_T/\text{PD}_C$	$\text{PCC}_N/\text{PCC}_T/\text{PCC}_C$	$\text{Time}_N/\text{Time}_T/\text{Time}_C$
dblp	0.1	19/34/115	171/561/6555	9/14/26	0.800/0.611/0.264	0.790/0.620/0.317	25/100/15.86
dblp	0.3	14/26/138	108/366/6693	7/11/23	0.9917/0.785/0.277	0.9918/0.789/0.384	11/30/16.99
pokec	0.1	13/72/288	121/1335/10592	3/8/27	0.678/0.341/0.129	0.636/0.393/0.170	672/1162/4401
pokec	0.3	6/71/278	21/1031/10142	2/6/25	0.815/0.321/0.132	0.793/0.406/0.172	298/980/4349
biomine	0.1	103/102/430	5231/5127/92200	18/33/79	0.540/0.538/0.211	0.540/0.538/0.217	1098/7642/5792
biomine	0.3	7/102/431	23/5125/92625	2/28/73	0.714/0.538/0.212	0.701/0.539/0.218	939/1563/5685

Table 5.3: Cohesiveness statistics of l-(k,θ)-nucleus N, (k,θ)-truss, and (k,θ)-core, C on dblp, pokec, and biomine. The number of vertices ($\lvert V_N\rvert / \lvert V_T\rvert / \lvert V_C\rvert$), the number of edges ($\lvert E_N\rvert / \lvert E_T\rvert / \lvert E_C\rvert$), maximum nucleus/truss/core score ($k_{Nmax}/k_{Tmax}/k_{Cmax}$), the probabilistic density ($PD_N/PD_T/PD_C$), and the probabilistic clustering coefficient (PCC_N, PCC_T, PCC_C), respectively.

and (k_{Cmax},η)-core, where k_{Nmax}, k_{Tmax}, and k_{Cmax} are maximum nucleus, truss and core scores, respectively. Also, for k_{Nmax}, k_{Tmax}, and k_{Cmax}, we might obtain more than one connected component; we report the average statistics over such components. We denote by V_N, V_T, V_C, the sets of nodes, by E_N, E_T, E_C, the sets of edges, by PD_N, PD_T, PD_C, the PD's and by PCC_N, PCC_T, PCC_C, the PCC's of nucleus, truss, and core components, respectively. The last column shows the running time for computing each decomposition. We observe that sometimes nucleus decomposition is faster than truss decomposition. This is because in nucleus decomposition there could be fewer triangles that survive the specified threshold in terms of support than edges in truss decomposition.

As can be seen in the table, (k_{Nmax},θ)-nucleus produces high quality results in terms of PD and PCC. We achieve a significantly higher PD and PCC for nucleus compared to truss and core. For instance, for *dblp* the PD for nucleus is 0.8 versus 0.611 and 0.264 for truss and core, which translates for nucleus being about 30% and 200% more dense than truss and core. Similar conclusions can be drawn for PCC as well.

Moreover, Figure 5.4 reports the average PD, average PCC, average edges in each $\boldsymbol{\ell}$-(k,θ)-nucleus, and number of connected components ($\boldsymbol{\ell}$-(k,θ)-nuclei) for an example dataset *flickr* with fixed $\theta = 0.3$ and varying k. We see that even for small values of k, PD and PCC are considerably high (above 70-80%). In general, PD and PCC become larger as k increases, since denser nuclei will be detected by removing triangles having low support probability to be part of a 4-clique. This causes the final subgraphs to have edges with high probability only. Furthermore, since $\boldsymbol{\ell}$-(k,θ)-nucleus implies connectivity, the number of connected components increases as k decreases. It results in an increase in the average number of edges in each $\boldsymbol{\ell}$-(k,θ)-nucleus.

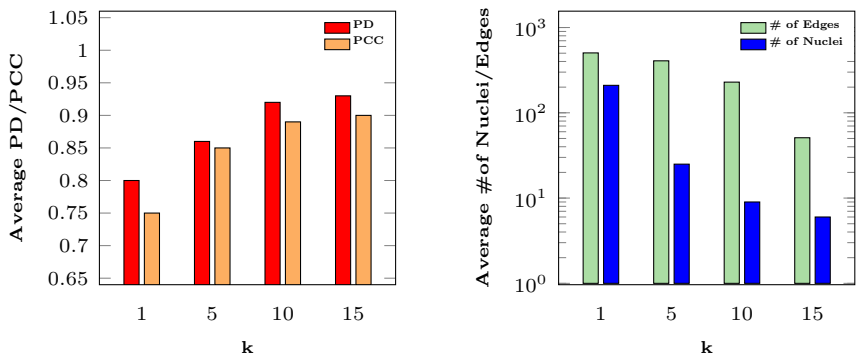


Figure 5.4: Average PD and PPC, average number of edges, average number of ℓ-(k,θ)-nuclei for flickr with $\theta = 0.3$.

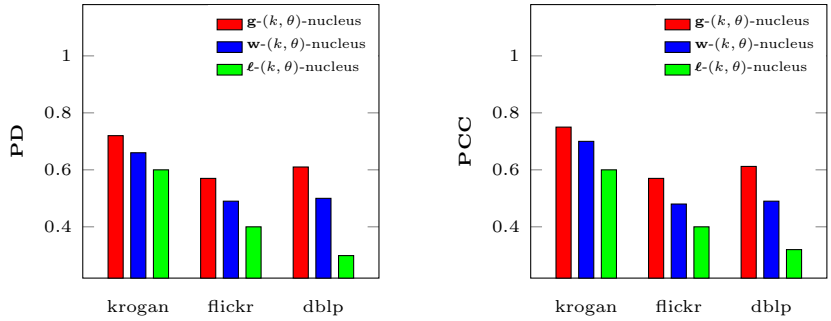


Figure 5.5: PD and PCC for **g**-(k,θ), **w**-(k,θ), *and* ℓ-(k,θ) *nuclei on krogan, flickr, and dblp.*

Finally, we compare the PD and PCC values of **g**-(k,θ)-nucleus, **w**-(k,θ)-nucleus, and ℓ-(k,θ)-nucleus, for *krogan*, *flickr*, and *dblp* datasets using $\theta = 0.001$, and averaging over all the possible values of k. The results are shown in Figure 5.5. We see that **g**-(k,θ)-nucleus achieves higher cohesiveness as expected. In addition, **w**-(k,θ)-nucleus exhibits quite good PD and PCC values higher than those for ℓ-(k,θ)-nucleus.

Effect of ϵ and δ. We consider *krogan* dataset with $\theta = 0.1$. The choice of ϵ and δ influence the number n of possible worlds we sample. For $\epsilon = 0.1$ and $\delta = 0.1$ we obtain $n = 150$. In order to see the fidelity of our results, we experiment by increasing n to higher values, namely $300, 500, 1000, 2000$. As the results in Table 5.4 show, the following metrics about global and weakly-global nuclei: average PD, average PCC, average number of vertices, and average number of edges change very little. Specifically, the first two metrics are dispersed by not more than 0.4% around their mean over the different values of n, and the last two metrics are dispersed by not more than 1.8%. There can be many ϵ and δ values corresponding to a given sample

n	AV(PD)		AV(PCC)		AV(Edge)		AV(Vertex)		ϵ	δ
150	.905	.726	.903	.770	12.744	55.631	5.336	11.971	0.1	0.1
300	.906	.733	.903	.773	12.725	52.960	5.334	11.543	0.07	0.05
500	.906	.729	.903	.767	13.005	53.883	5.383	11.703	0.05	0.06
1000	.905	.725	.902	.766	12.823	53.772	5.356	11.745	0.05	0.01
2000	.906	.727	.903	.768	12.782	54.264	5.350	11.792	0.03	0.05
AV	.906	.728	.903	.769	12.816	54.102	5.352	11.751		
SD	.0004	.003	.0003	.002	.112	.978	.020	.155		

Table 5.4: Effect of sample size (n), ϵ, and δ on different average metrics, average PD, average PCC, average number of edges, and average number of vertices for global and weakly-global nuclei. The first and second columns for each metric are for global and weakly-global nuclei, respectively. The results shown here are on krogan with $\theta = 0.1$. Observe that standard deviation (SD) is not more than 1.8% of the average for all columns. For some of the columns SD is much smaller, e.g. for average PD (first column) it is only 0.05%.

size; for illustration, for $n = 150$, we can have $\epsilon = 0.1$, $\delta = 0.1$, whereas for $n = 2000$, we can have $\epsilon = 0.03$, $\delta = 0.05$, i.e. we see that even though in the latter case the ϵ and δ decrease by a factor of 3 and 2, respectively, still the nuclei results in terms of the aforementioned metrics are almost the same. This validates the choice of ϵ and δ to 0.1 since lower values do not offer significant improvement in the quality of results.

5.7.4 Case Study

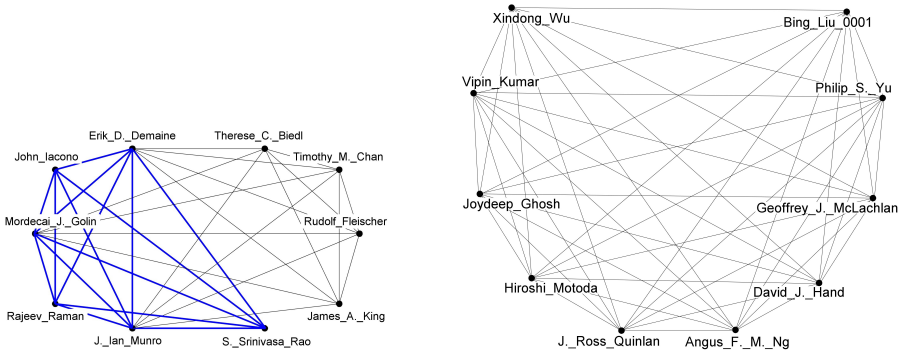
DBLP. To show the usefulness of nucleus decomposition in probabilistic graphs, we apply our decomposition algorithms to solve the *task-driven team formation* problem for a DBLP network. In task-driven team formation [9], we are given a probabilistic graph $\mathcal{G}^T = (V, E, p^T)$, which is particularly obtained for task T. Vertices in $\mathcal{G}^T$ are individuals and edge probabilities are obtained with respect to task T as described in [9]. Given a query $\langle Q, T\rangle$, where $Q \subset V$, and T is a set of keywords describing a task, the goal is to find a set of vertices that contain Q and make a good team to perform the task described by the keywords in T. By a good team we mean a good affinity among the team members in terms of collaboration for the given task. To solve task-driven team formation using nucleus decomposition, we extend the definition of [9] to employ probabilistic nucleus: Given a probabilistic graph $\mathcal{G}^T = (V, E, p^T)$ with respect to a task T, a query set Q of vertices, and a threshold θ, apply nucleus decomposition on $\mathcal{G}^T$ and find a (k, θ)-nucleus (local/weakly-global/global) which **(1)**

contains the vertices in Q, and **(2)** has the highest value of k for the given θ, and return the obtained subgraph as a solution.

In our experiment, we use a DBLP collaboration network from [9], where vertices are authors, and edges represent collaboration on at least one paper. The dataset has $1,089,442$ vertices and $4,144,697$ edges. For each edge, we take the bag of words of the title of all papers coauthored by the two authors connected by the edge and apply Latent Dirichlet Allocation (LDA) [6, 9] to infer its topics and calculate the edge probability. Given a task T with keywords, and the input collaboration network, we obtain a probabilistic graph $\mathcal{G}^T$, in which $p(u, v)$ represents the collaboration level in the papers co-authored by u and v related to task T ([9, 34]).

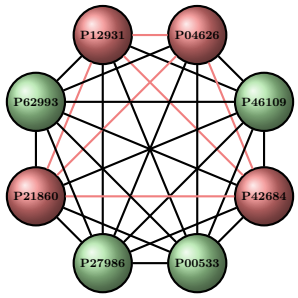
The first sample query we consider is $\langle$\{"algorithm"\}, \{"Erik_D._Demaine", "J._Ian_Munro", "John_Iacono"\}$\rangle$. Figure 5.6a shows the subgraph obtained by $\boldsymbol{\ell}$-(k, θ)-nucleus and $\mathbf{w}$-(k, θ)-nucleus decompositions, where $k = 2$ and $\theta = 10^{-11}$. The threshold is the same as in the case studies of previous works (on truss and core). As discussed in [9], the edge probabilities in the dataset are very small, which requires setting threshold θ to a small value. More systematically, picking an appropriate value for the threshold can be done using binary search over $(0, b]$, where $b \leq 1$. The subgraph contains all the three authors in the query. It has 10 vertices and 33 edges. As can be seen, the obtained subgraph is quite good for task-driven team formation. All the authors in the subgraph are well-known and have strong collaboration affinity to work on a research paper related to algorithms. A $\mathbf{g}$-(k, θ)-nucleus (same k and θ) that contains the query vertices is shown with thick blue edges in the same figure. As expected, this subgraph is more cohesive and it happens to be a clique of size 6. Its density and clustering coefficient (PCC) is 0.138 and 0.140 as opposed to 0.099 and 0.110 for the local and weakly-global subgraphs. From a research perspective the collaborations of the academicians in the blue subgraph are more focused on designing efficient data-structures.

The second query we use shows the usefulness of the weakly-global notion. It has keyword \{"algorithm"\} and vertices \{"Xindong_Wu", "Bing_Liu_0001", "Vipin_Kumar"\}. Figure 5.6b shows the $\mathbf{w}$-(k, θ) nucleus for this query, where the threshold is the same as before, and $k = 1$. The local nucleus containing the query authors had more than 100 nodes while the global nucleus containing these three query authors was empty. This example shows that the weakly global notion can discover interesting teams when the other two notions produce teams that are too big or too small (or empty). In particular, all the authors in the resulting subgraph are very well-know and



(a) (b)

Figure 5.6: ***a)*** *A case study of task-driven team formation with keyword {"algorithm"}, and query vertices {"Erik_D._Demaine", "J._Ian_Munro", "John_Iacono"}, $k = 2$, and $\theta = 10^{-11}$. The depicted graph with thick blue edges corresponds to a* $\mathbf{g}$*-(k, θ) nucleus. The whole graph (of 10 vertices) is a* $\boldsymbol{\ell}$*-(k, θ) nucleus which coincides with a* $\mathbf{w}$*-(k, θ) nucleus in this example.* ***b)*** *A weakly-global* $\mathbf{w}$*-(k, θ) nucleus for task-driven team formation with query nodes {"Xindong_Wu", "Bing_Liu_0001", "Vipin_Kumar"}, and keyword {"algorithm"}. $k = 1$, and $\theta = 10^{-11}$.*



(a)

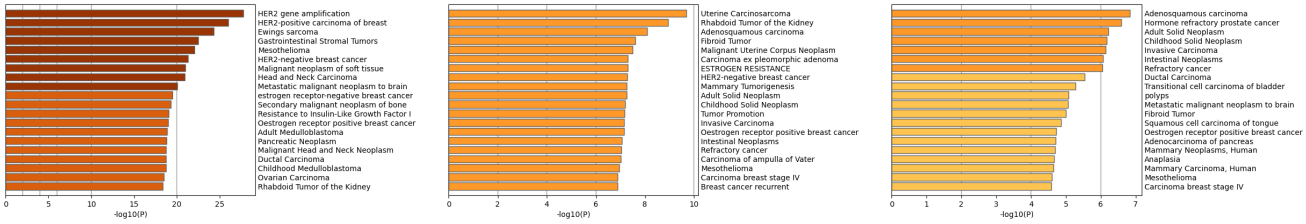
Figure 5.7: $\mathbf{w}$*-(k, θ) nucleus (green and pink nodes) and a* $\mathbf{g}$*-(k, θ)-nucleus (pink nodes) that contain the proteins of interest P04626, P12931, P42684. $k = 1$ and $\theta = 0.001$.*

have similar research area which can form a good team related to keyword algorithm (query keyword).

Human Biomine. We use the human biomine dataset [60], which has 861,812 nodes and 8,666,287 edges. This dataset is different from the biomine dataset we used for our efficiency evaluation. We consider how our notions perform in detecting proteins/genes that interact with the **SARS-CoV-2** coronavirus. Bouhaddou et al. [11] found that during the **SARS-CoV-2** virus infection, changes in activities can happen for human kinases. We select three proteins, P04626, P12931 and P42684; they are tyrosine kinase-related proteins and come from UniProt, which is a freely accessible database of protein sequences and functional information. The gene names

Notion	Max k	Nodes	Density
l-core	88	2408	0.04
g-core	31	10026	0.01
l-truss	4	5787	0.01
l-nucleus	1	95	0.06
g-truss	2	10	**0.44**
w-nucleus	1	8	**0.51**
g-nucleus	1	4	**0.56**

Table 5.5: Comparison of different dense subgraph notions with respect to (1) max k for which the subgraph contains the proteins of interest, (2) number of nodes in the subgraph, and (3) density of the subgraph. $\theta = 0.001$.



(a) (b) (c)

*Figure 5.8: Top enriched terms related to diseases in the detected subgraphs by local, weakly-global, and global nucleus decompsoitions. Variable **P** on the x-axis refers to p-value.*

associated with these proteins are SRC, ERBB2, and ABL2. These proteins have received literature support for interaction with **SARS-CoV-2** coronavirus [53, 96, 75, 30, 94, 21, 11]. We refer to them as proteins of interest. We find the subgraphs obtained by local, weakly-global, and global nucleus decomposition which contain these three nodes. Moreover, at the same time we compare these graphs with their counterparts, truss and core in terms of density and size of the subgraph. For all the notions we set threshold $\theta = 0.001$.

Table 5.5 shows the comparison of different dense subgraph notions with respect to (1) largest k for which the subgraph contains the proteins of interest, (2) number of nodes in the subgraph, and (3) density of the subgraph. We see that l-nucleus is denser than l-truss and both l-core and g-core. Also, w-nucleus and g-nucleus are denser than g-truss. In terms of nodes, l-nucleus gives a subgraph which is much smaller than the subgraphs of l-core, g-core, and l-truss. More precisely, with respect to l-nucleus, the three proteins of interest appear in a nucleus of 95 vertices and 509 edges. To see which kind of biology function/process our detected community represent, we use *Metascape*

(`https://metascape.org/gp/index.html#/main/step1`). Metascape [97] is a web-based portal that provides comprehensive gene list annotation and analysis resources. Using Metascape, we find that the proteins in the local nucleus are associated with several diseases, most of them being forms of cancer (16 out 20). The p-values of the association are less than 10^{-18}, which is statistically very significant. Please see Figure 5.8 for more details.

Figure 5.7 shows the weakly global and global nuclei which contain the proteins of interest. All the nodes (green and pink) comprise the weakly global subgraph. The pink nodes comprise the global nucleus subgraph. Using Metascape, we find that the proteins in our weakly-global and global subgraphs are associated with some more specific forms of cancer such as *Uterine Carcinosarcoma* and *Hormone Refractory Prostate Cancer*, respectively, with p-values less than 10^{-6}, which are statistically quite significant, especially given the fact that these subgraphs are much smaller than the local nucleus (in general, the more observations we have, the smaller the p-values become). These findings are useful to biologists in order to perform targeted tests for checking whether drugs for the treatment of these diseases can also be repurposed for treating COVID-19 [30]. There are over 250 anticancer drugs approved by the FDA, but far fewer for specific kinds of cancer. Thus, showing connections to specific forms helps narrow the choice of drugs to repurpose.

In summary, it is running all the three versions of nucleus decomposition on the Biomine dataset that gives surprising subgraphs pointing to potentially useful further investigation by biologists. Running only local nucleus decomposition will miss such interesting groups, no matter how we set k and θ.

5.8 Conclusions

In this chapter, we presented several key contributions that we made. We introduced the notion of local, weakly-global and global nuclei for probabilistic graphs. We showed that computing weakly-global and global nuclei is intractable. We complemented these hardness results with effective algorithms to approximate them using techniques from Monte-Carlo sampling.

We designed a polynomial time, peeling based algorithm for computing local nuclei based on dynamic programming and showed that its efficiency can be much improved using novel approximations based on Poisson, Binomial and Normal distributions. Finally, using an in-depth experimental study, we demonstrated the efficiency, scalability and accuracy of our algorithms for nucleus decomposition on real world datasets.

Chapter 6

Conclusions and Future Work

Dense subgraph mining is an important task in analyzing the structure of network graphs. Many real-world networks, such as social, trust, communication, and biological networks, are associated with uncertainty, and as a result, can be modeled as probabilistic graphs. Core, truss and nucleus decompositions are popular notions of dense subgraphs with a wide range of applications. For instance, they can be used in describing biological functions of proteins in protein protein interaction networks, computing task-driven team formation, and speeding up computation of other definitions of dense graphs such as maximum cliques. In this dissertation, we have considered core, truss, and nucleus decomposition over probabilistic graphs. Finding these subgraphs in probabilistic context is challenging due to uncertainty in such graphs which causes combinatorial nature for many mining task in these graphs. Thus, we have proposed well-suited efficient algorithmic approaches. In particular, the contributions of dissertation were as follows.

- We proposed efficient algorithms for probabilistic core decomposition. Our first algorithm was based on the idea of recursively removing vertex of the smallest degree. Our contribution was in designing efficient arrays for storing important bookkeeping information. Also, we obtained easy-to-compute lower-bounds on probabilistic degree of vertices using central limit theorem. These lower-bounds were used in the peeling process. We also proposed a sequential algorithm that was able to handle the input graphs which do not fill in memory. The performance of the proposed algorithms was evaluated on real-world datasets. The details of our proposed approaches were discussed in Chapter 2.

- We considered truss decomposition over probabilistic graphs. We designed an

efficient algorithm which was based on recursive edge deletion and was able to to calculate truss decomposition in big probabilistic graphs not possible with the pure DP approach. In the peeling process, tail probability of support of edges were estimated using statistical approximation. In addition, we proposed an algorithm based on h-index updating for probabilistic graphs. We proved the correctness of our algorithm and showed that convergence to true truss values can be obtained after a few number of iterations. Finally, the proposed algorithms were evaluated on real datasets. In Chapters 3 and 4, details of these algorithms were included.

- We extended nucleus decomposition in deterministic graphs to probabilistic context. We proposed three notions of nucleus decomposition in probabilistic graphs; local, global, and weakly-global. We proved that computing global and weakly-global decompositions is intractable, namely #P-hard and NP-hard, respectively. We proposed efficient algorithms based on Monte-Carlo sampling for these two cases. For local case, we showed that the computation can be done in polynomial time. However, the main challenge is efficient computation of the probability of each triangle to be contained in k 4-cliques. We presented an exact method and a variety of approximation methods for this task, which combined with a triangle peeling approach, finds local nucleus decomposition efficiently. Finally, we demonstrated efficiency of the proposed algorithms on real datasets in Chapter 5.

6.1 Future Work

In the following we discuss some ideas and future research directions that can be beneficial to investigate for future work and plans beyond this dissertation.

While in the literature, the most common probabilistic model is a graph in which edges exist independently of each other, one area of research is to consider dependency of the existence probability of a portion or all the edges in the graph. In this case, the proposed algorithms for core, truss, nucleus decompositions need to be engineered to handle dependency of edges.

Moreover, in this thesis, we considered undirected probabilistic graphs. However, complex networks in real-world have often rich information. For instance,

their edges can be asymmetric, as well as having nodes with categorical and numeric attributes [68]. Finding dense subgraphs based on motifs with the presence of this rich information can be challenging particularly for probabilistic graphs, and it can be further investigated in the future.

As another area for future work, in the literature, the assumption for connectivity of the probabilistic dense notions is simple deterministic connectivity which ignores edge probabilities and clique structure. As [69] argue, clique-based connectivity needs to be systematically addressed. So, it is of great importance to define clique-based connectivity based on the probabilistic nature of these graphs and propose efficient algorithms that can find dense subgraphs that are connected with high probability.

Another area of research is designing algorithms that given k, can find (k, η)-core, (k, η)-truss, and (k, θ)-nucleus for various possible η and θ, respectively. It should be noted that this problem is well-defined since there exist a finite number of k-core, k-truss, and k-nucleus in a (probabilistic) graph, each of which must be a (k, η)-core, (k, η)-truss, and (k, θ)-nucleus for some maximum possible η and θ, respectively. Last but not least, the proposed algorithms for global and weakly-global nucleus subgraphs, can be further improved to speed up computation of these subgraphs.

Bibliography

[1] L. Antiqueira, O. N. Oliveira Jr, L. da Fontoura Costa, and M. d. G. V. Nunes. A complex network approach to text summarization. *Information Sciences*, 179(5):584–599, 2009.

[2] S. Aridhi, M. Brugnara, and Y. Montresor, A.and Velegrakis. Distributed k-core decomposition and maintenance in large dynamic graphs. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 161–168. ACM, 2016.

[3] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[4] V. Batagelj and M. Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2):129–145, 2011.

[5] P. Billingsley. *Mathematical Methods of Statistics*. John Wiley & Sons, 3 edition, 1995.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[7] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*. ACM Press, 2011.

[8] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004.

[9] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1316–1325. ACM, 2014.

[10] Ch. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier. Influence maximization in social networks: Towards an optimal algorithmic solution. *CoRR, abs/1212.0884*, 2012.

[11] M. Bouhaddou, D. Memon, B. Meyer, K. M. White, V. V. Rezelj, M. C. Marrero, B. J. Polacco, J. E. Melnyk, S. Ulferts, R. M. Kaake, et al. The global phosphorylation landscape of sars-cov-2 infection. *Cell*, 182(3):685–712, 2020.

[12] C. Budak, D. Agrawal, and A. El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World wide web*, pages 665–674. ACM, 2011.

[13] G. Cavallaro. Genome-wide analysis of eukaryotic twin cx 9 c proteins. *Molecular BioSystems*, 6(12):2459–2470, 2010.

[14] Y. Che, Zh. Lai, Sh. Sun, Y. Wang, and Q. Luo. Accelerating truss decomposition on heterogeneous processors. *Proceedings of the VLDB Endowment*, 13(10):1751–1764, 2020.

[15] L. Chen and X. Lian. Query processing over uncertain databases. *Synthesis Lectures on Data Management*, 4(6):1–101, 2012.

[16] Shu Chen, Ran Wei, Diana Popova, and Alex Thomo. Efficient computation of importance based communities in web-scale networks using a single machine. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1553–1562, 2016.

[17] J. Cheng, Y. Ke, Sh. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 51–62. IEEE, 2011.

[18] H. Cramér. *Mathematical Methods of Statistics*. Princeton University Press, 1946.

[19] M. T. Dittrich, G. W. Klau, A. Rosenwald, Th. Dandekar, and T. Müller. Identifying functional modules in protein–protein interaction networks: an integrated exact approach. *Bioinformatics*, 24(13):i223–i231, 2008.

[20] J. Dong and S. Horvath. Understanding network concepts in modules. *BMC systems biology*, 1(1):24, 2007.

[21] K. H. Ebrahimi, J. Gilbert-Jaramillo, W. S. James, and J. S. McCullagh. Interferon-stimulated gene products as regulators of central carbon metabolism. *The FEBS journal*, 288(12):3715, 2021.

[22] W. Ehm. Binomial approximation to the poisson binomial distribution. *Statistics & Probability Letters*, 11(1):7–16, 1991.

[23] F. Esfahani, M. Daneshmand, V. Srinivasan, A. Thomo, and K. Wu. Truss decomposition on large probabilistic networks using h-index. In *33rd International Conference on Scientific and Statistical Database Management*, pages 145–156, 2021.

[24] F. Esfahani, V. Srinivasan, A. Thomo, and K. Wu. Efficient computation of probabilistic core decomposition at web-scale. In *EDBT*, pages 325–336, 2019.

[25] F. Esfahani, J. Wu, V. Srinivasan, A. Thomo, and K. Wu. Fast truss decomposition in large-scale probabilistic graphs. In *Proceedings of the 22nd International Conference on Extending Database Technology*, pages 722–725, 2019.

[26] Fatemeh Esfahani, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. Nucleus decomposition in probabilistic graphs: Hardness and algorithms. *arXiv preprint arXiv:2006.01958*, 2021.

[27] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *The VLDB Journal*, 29(1):353–392, 2020.

[28] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[29] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010.

[30] Y. Guo, F. Esfahani, X. Shao, V. Srinivasan, A. Thomo, L. Xing, and X. Zhang. Integrative covid-19 biological network inference with probabilistic core decomposition. *Briefings in Bioinformatics, doi: 10.1093/bib/bbab455*, 2021.

[31] F. A. Haight. Handbook of the poisson distribution. 1967.

[32] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[33] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1311–1322. ACM, 2014.

[34] X. Huang, W. Lu, and L. V. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proceedings of the 2016 International Conference on Management of Data*, pages 77–90. ACM, 2016.

[35] R. Jin, L. Liu, and Ch. C. Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 992–1000. ACM, 2011.

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

[37] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[38] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD'03*, 2003.

[39] A. Khan, F. Bonchi, F. Gullo, and A. Nufer. Conditional reliability in uncertain graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[40] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo. K-core decomposition of large networks on a single pc. In *Proceedings of the 41st International Conference on Very Large Data Bases*, pages 13–23. ACM, 2015.

[41] H. Kobayashi, B.L. Mark, and W. Turin. *Probability, Random Processes, and Statistical Analysis*. Cambridge University Press, 2011.

[42] N. Korovaiko and A. Thomo. Trust prediction from user-item ratings. *Social Network Analysis and Mining*, 3(3):749–759, 2013.

[43] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, Sh. Pu, N. Datta, A. P. Tikuisis, et al. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084):637, 2006.

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

[45] L. Le Cam. An approximation theorem for the poisson binomial distribution. *Pacific Journal of Mathematics*, 10(4):1181–1197, 1960.

[46] V. E. Lee, N. Ruan, R. Jin, and Ch. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[47] R. Li, L. Qin, F. Ye, G. Wang, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Finding skyline communities in multi-valued networks. *The VLDB Journal*, pages 1–26, 2020.

[48] X. Li, M. Wu, Ch. K. Kwoh, and S.K. Ng. Computational approaches for detecting protein complexes from protein interaction networks: a survey. *BMC genomics*, 11(1):S3, 2010.

[49] Y. Liu, J. Lu, H. Yang, X. Xiao, and Zh. Wei. Towards maximum independent sets on massive graphs. *Proceedings of the VLDB Endowment*, 8(13):2122–2133, 2015.

[50] A. Lyapunov. Sur une proposition de la théorie des probabilités. *Bulletin de l'Academie Impériale des Sciences de St. Petersbourg*, 13:359–386, 1900.

[51] A. Lyapunov. Nouvelle forme de la théoreme dur la limite de probabilité. *Mémoires de lAcademie Impériale des Sciences de St. Petersbourg*, 12:1–24, 1901.

[52] X. Ma, G. Zhou, J. Shang, J. Wang, J. Peng, and J. Han. Detection of complexes in biological networks through diversified dense subgraph mining. *Journal of Computational Biology*, 24(9):923–941, 2017.

[53] M. Marchetti. Covid-19-driven endothelial damage: complement, hif-1, and abl2 are potential pathways of damage and targets for cure. *Annals of hematology*, pages 1–7, 2020.

[54] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Transactions on parallel and distributed systems*, 24(2):288–300, 2013.

[55] N. Mukhopadhyay. *Probability and statistical inference*. CRC Press, 2000.

[56] A. Papoulis and S. U. Pillai. *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.

[57] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The pursuit of a good possible world: extracting representative instances of uncertain graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 967–978, 2014.

[58] P. Parchas, N. Papailiou, D. Papadias, and F. Bonchi. Uncertain graph sparsification. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2435–2449, 2018.

[59] J. J. Pfeiffer and J. Neville. Methods to determine node centrality and clustering in graphs with uncertain structure. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[60] V. Podpean, . Ramak, K. Gruden, H. Toivonen, and N. Lavra. Interactive exploration of heterogeneous biological networks with biomine explorer. *Bioinformatics*, 06 2019.

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

[62] A. Röllin. Translated poisson approximation using exchangeable pair couplings. *The Annals of Applied Probability*, 17(5/6):1596–1614, 2007.

[63] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K. L. Wu, and Ü. V. Çatalyürek. Incremental k-core decomposition: algorithms and evaluation. *The VLDB Journal*, 25(3):425–447, 2016.

[64] A. E. Sarıyüce, C. Seshadhri, and A. Pinar. Local algorithms for hierarchical dense subgraph discovery. *Proc. of the VLDB Endowment*, 12(1):43–56, 2018.

[65] A. E. Sarıyüce, C. Seshadhri, and A. Pinar. Local algorithms for hierarchical dense subgraph discovery. *VLDB*, 2019.

[66] A. E. Sarıyüce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings of the 24th International Conference on World Wide Web*, pages 927–937, 2015.

[67] A. E. Sarıyüce, C Seshadhri, Ali Pinar, and Ümit V Çatalyürek. Nucleus decompositions for identifying hierarchy of dense subgraphs. *ACM Transactions on the Web (TWEB)*, 11(3):1–27, 2017.

[68] Ahmet Erdem Sarıyüce. Motif-driven dense subgraph discovery in directed and labeled networks. In *Proceedings of the Web Conference 2021*, pages 379–390, 2021.

[69] Ahmet Erdem Sariyüce and Ali Pinar. Fast hierarchy construction for dense subgraphs. *Proc. VLDB Endow.*, 10(3):97–108, 2016.

[70] R. Saxena, Sh. Kaur, and V. Bhatnagar. Social centrality using network hierarchy and community structure. *Data Mining and Knowledge Discovery*, 32(5):1421–1443, 2018.

[71] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[72] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular systems biology*, 3(1):88, 2007.

[73] I. G. Shevtsova. An improvement of convergence rate estimates in the lyapunov theorem. *Doklady Mathematics*, 82(3):862864, 2010.

[74] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 75–86. ACM, 2014.

[75] K. Taniguchi-Ponciano, E. Vadillo, Hé. Mayani, Cé. R. Gonzalez-Bonilla, J. Torres, A. Majluf, G. Flores-Padilla, N. Wacher-Rodarte, J. C. Galan, E. Ferat-Osorio, et al. Increased expression of hypoxia-induced factor 1α mRNA and its related genes in myeloid blood cells from critically ill COVID-19 patients. *Annals of Medicine*, 53(1):197–207, 2021.

[76] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences*, 109(16):5962–5966, 2012.

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

[78] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5(9):812–823, 2012.

[79] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu. I/o efficient core graph decomposition at web scale. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 133–144. IEEE, 2016.

[80] T. Wolf, A. Schröter, D. Damian, L. D. Panjer, and Th. H. Nguyen. Mining task-based social networks to explore collaboration in software teams. *IEEE Software*, 26(1):58–66, 2009.

[81] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu. Core decomposition in large temporal graphs. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 649–658. IEEE, 2015.

[82] J. Wu, A. Goshulak, V. Srinivasan, and A. Thomo. K-truss decomposition of large networks on a single consumer-grade machine. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 873–880. IEEE, 2018.

[83] Jian Wu, Venkatesh Srinivasan, and Alex Thomo. Graph-xll: a graph library for extra large graph analytics on a single machine. In *2019 10th International Con-*

ference on Information, Intelligence, Systems and Applications (IISA), pages 1–7. IEEE, 2019.

[84] Q. Wu, X. Huang, A. Culbreth, J. Waltz, L. E. Hong, and Sh. Chen. Extracting brain disease-related connectome subgraphs by adaptive dense subgraph discovery. *bioRxiv*, 2020.

[85] Y. Xing, N. Xiao, Y. Lu, R. Li, S. Yu, and S. Gao. Fast truss decomposition in memory. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 719–729, 2017.

[86] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient keyword search on uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2767–2779, 2013.

[87] Y. Yuan, G. Wang, H. Wang, and L. Chen. Efficient subgraph search over large uncertain graphs. *Proceedings of the VLDB Endowment*, 4(11):876–886, 2011.

[88] B. Zhang and S. Horvath. A general framework for weighted gene co-expression network analysis. *Statistical applications in genetics and molecular biology*, 4(1), 2005.

[89] S. Zhang, D. Zhou, M. Y. Yildirim, S. Alcorn, J. He, H. Davulcu, and H. Tong. Hidden: hierarchical dense subgraph detection with application to financial fraud detection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 570–578. SIAM, 2017.

[90] Y. Zhang and S. Parthasarathy. Extracting, analyzing and visualizing triangle k-core motifs within networks. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1049–1060. IEEE, 2012.

[91] Zh. Zhang, J. X. Yu, L. Qin, L. Chang, and X. Lin. I/o efficient: computing sccs in massive graphs. *The VLDB Journal*, 24(2):245–270, 2015.

[92] Zh. Zhang, J. X. Yu, L. Qin, and Z. Shang. Divide & conquer: I/o efficient depth-first search. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 445–458. ACM, 2015.

[93] F. Zhao and A. K. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. In *Proceedings of the VLDB Endowment*, volume 6, pages 85–96. VLDB Endowment, 2012.

[94] H. Zhao, M. Mendenhall, and M. W. Deininger. Imatinib is not a potent anti-sars-cov-2 drug. *Leukemia*, 34(11):3085–3087, 2020.

[95] Y. Zhao, X. Dong, and Y. Yin. Effective and efficient dense subgraph query in large-scale social internet of things. *IEEE Transactions on Industrial Informatics*, 16(4):2726–2736, 2019.

[96] W. J. Zheng, Q. Yan, Y. Ni, Sh. F Zh., L. Yang, H. Zhuang, X. Liu, and Y. Jiang. Examining the effector mechanisms of Xuebijing injection on COVID-19 based on network pharmacology. *BioData mining*, 13:17, 2020.

[97] Y. Zhou, B. Zhou, L. Pache, M. Chang, A. H. Khodabakhshi, O. Tanaseichuk, Ch. Benner, and S. K. Chanda. Metascape provides a biologist-oriented resource for the analysis of systems-level datasets. *Nature communications*, 10(1):1–10, 2019.

[98] Zh. Zou, H. Gao, and J. Li. 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*, pages 633–642. ACM, 2010.

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

[100] D. Zwillinger and S. Kokoska. *CRC Standard probability and statistics tables and formulae*. Chapman and Hall/CRC, Boca Raton, Florida, USA, 2000.

[101] D. Zwillinger and S. Kokoska. *Probability Theory*. Springer-Verlag, London, 2013.