Towards Graph Analytic and Privacy Protection

Dongqing Xiao

A Proposal for a PhD Dissertation in Computer Science

Worcester Polytechnic Institute, Worcester, MA

Dec 2016

Committee Members:

Dr. Mohamed Y. Eltabakh, Assistant Professor, Worcester Polytechnic Institute. Advisor.

Dr. Elke A. Rundensteiner, Professor, Worcester Polytechnic Institute.

Dr. Xiangnan Kong, Assistant Professor, Worcester Polytechnic Institute.

Dr. Yuanyuan Tian, Researcher, IBM Almaden, External member.

Abstract

In the real world, graph structured data is ubiquitous. For instance, social networks, communication networks, biological networks, etc. can all be modeled as graphs. Various graph analysis mining methods have been proposed to deepen the understanding of the graph data, in particular in sequential platforms. However, the graph size continues to increase and becomes massive involving billions of vertices and edges. Such massive graphs can easily exceed the available memory of a single commodity computer and pose a new challenge for graph mining tasks. One of the methods to deal with large graphs is to exploit parallel programming paradigms including MapReduce. Within this scope, we study how to design an efficient distributed triangle listing algorithm for web-scale graphs with MapReduce.

• *Distributed triangle listing algorithm* Inspired by the theoretical and practical significance, the triangle listing problem has then been studied in MapReduce. Triangle listing requires accessing the neighbors of the neighbor of a vertex, which may appear arbitrarily in different graph partitions (poor locality in data access). Existing algorithms suffer from generating and shuffling huge amounts of intermediate data, where interestingly, a large percentage of this data is redundant. Inspired by this observation, we present the *"Bermuda"* method that effectively reduces the size of the intermediate data via redundancy elimination and sharing of messages whenever possible.

Besides, in many real-world applications, graphs are not deterministic but probabilistic in nature for a variety of reasons. In these cases, data is represented as an uncertain graph whose edges are accompanied with a probability of existence. Uncertain graphs have been shown to be invaluable dataset for a variety of real applications and analytic tasks. However, the raw uncertain graph is particularly sensitive: it contains personal details and sensitive connections between them which are not public and should not be revealed. Thus, before such uncertain graphs can be released for research purposes, the data needs to be anonymized to prevent potential re-identification attacks. However, previous graph anonymization approaches are all geared towards deterministic graphs. It calls for novel anonymization techniques that are able to cope with the additional uncertainty explicitly. Within this scope, we study uncertain graph anonymization techniques for resisting privacy attacks depend on the node degree statistic and degree probabilistic distribution.

• *Degree Anonymization over Uncertain Graphs* In this work, we consider that the adversary relies upon the knowledge of node degree statistic (the expected value) in devising a matching attack for nodes in the released uncertain graph, which is a well-known privacy attack while in the new context–uncertain graph. However, applying existing methods straightforward can not provide efficient privacy protection meanwhile incurring a large amount of utility loss. To prevent such matching attack, we consider (k, ϵ)-*obf*, a popular privacy notation, to protect such degree-based matching attack. We extend the existing (k, ϵ)-*obf* framework to work over the uncertain graph. We also design a set of heuristic-based techniques to make the anonymization mechanism efficient meanwhile preserving data utilities, especially the graph reliability. Extensive experiments on large real datasets show the satisfactory performance of our methods in terms of privacy protection, efficiency, and practical utilities.

• *Probabilistic Degree Anonymization over Uncertain Graphs* In this work, we consider that the adversary relies upon the knowledge of node degree probability distribution rather than aggregated statistic in devising a matching attack for nodes in the released uncertain graph. We formally present the definition of probabilistic degree-based node re-identification attack, which is a newly identified privacy attack in the uncertain graph. Similarly, we extend the concept of (k, ϵ)-*obf* to protect such matching attack. We develop a general framework which injects uncertainty to edges in the uncertain graph for obtaining (k, ϵ)-*obf*. In particular, we propose to utilize the clustering result of probabilistic degree for guiding the perturbation which maximizes the likelihood of providing higher privacy guarantee. We also consider utilizing the genetic algorithm for the optimization of edge perturbation strategy. Extensive studies including privacy protection, efficiency, and practical utility evaluation will be conducted on large real datasets to evaluate the effectiveness and efficiency of my proposed approaches.

Contents

Acknowledgments

I would like to express my sincere thanks to my advisor Prof. Mohamed Eltabakh. Without his guidance and support, this work would not have been possible. I am grateful to Prof. Xiangnan Kong for always being patient and being there for our discussion. My thank you also goes to Prof. Elke rundensteiner and Dr. YuanYuan Tian for serving on my dissertation committee and their insightful comments. I owe much to my friends and all the DSRG members for having the valuable conversations on my works and sharing many laughs as well as frustrations.

Chapter 1

Introduction

1.1 Motivation

1.1.1 Distributed Triangle Listing for Massive Graphs

Graphs arise naturally in many real-world applications such as social networks, biomedical networks, and communication networks. In these applications, the graph can often be massive involving billions of vertices and edges. For example, Facebook's social network involves more than 1.23 billion users (vertices), and more than 208 billion friendships (edges). Such massive graphs can easily exceed the available memory of a single commodity computer. That is why distributed analysis on massive graphs has become an important research area in recent years [35, 21].

Triangle listing—which involves listing all triangles in a given graph—is well identified as a building-block operation in many graph analysis and mining techniques [19, 30]. First, several graph metrics can be directly obtained from triangle listing, e.g., *clustering coefficient* and *transitivity*. Such graph metrics have wide applications including quantifying graph density, detecting spam pages in web graphs, and measuring content quality in social networks [8]. Moreover, triangle listing has a broad range of applications including the discovery of dense sub-graphs [30], study of motif occurrences [36], and uncovering of hidden thematic relations in the web [19]. There is another well-known and closely-related problem to triangle listing, which is the *triangle counting* problem. Clearly, solving the triangle listing problem would automatically solve triangle counting, but not vice versa. Compared to triangle counting, triangle listing serves a broader range of applications. For example, Motif identification [36], community detection [9], and dense subgraphs [30] are all dependent on the more complex *triangle listing* problem.

Several techniques have been proposed for processing web-scale graphs including streaming algorithms [8, 13], external-memory algorithms [25, 31, 33], and distributed parallel algorithms [3, 48]. The streaming algorithms are limited to the *approximate triangle counting* problem. External-memory algorithms exploit asynchronous I/O and multi-core parallelism for efficient triangle listing [33, 22, 25]. In spite of achieving an impressive performance, external-memory approaches assume that the input graphs are in a centralized storage, which is not the case for many emerging applications that generate graphs distributed in nature. Even more seriously, external-memory approaches cannot easily scale up in terms of computing resources and parallelization degree. Algorithm [48] presents a parallel algorithm for exact triangle counting using the MapReduce framework. The algorithm proposes a partitioning scheme that improves the memory requirements to some extent, yet it still suffers from a huge communication cost. Algorithm [3] presents an efficient MPI-based distributed memory algorithm on the basis of [48] with load balancing techniques. However, as a memory-based algorithm, it suffers from memory limitations.

In addition to these techniques, several distributed and specialized graph frameworks have been recently proposed as general-purpose graph processing engines [35, 21, 20]. However, most of these frameworks are customized for iterative graph processing where distributed computations can be kept in-memory for faster subsequent iterations. However, the triangle listing algorithms are not iterative and would not make use of these optimizations.

1.1.2 Uncertain Graph Anonymization

Network data has become increasingly important in recent years of the greater importance of various application domains such as Web, social network, biological networks, and communication networks. The semantics and the interpretation of the nodes and the links may vary with application domain, *e.g.*, in a social network node can represent individuals and their connection captures friendship, while in a protein-to-protein network (PPI) nodes are proteins and connections refer to their interactions. These graph data carries valuable information and are analyzed in various ways to exact knowledge. For example, social networks provide significant insight into the psychological behavior of individuals or PPI networks are widely studied to elucidate the mechanism of diseases.

In the same time, the existence of the relationship between two entities is uncertain due to noisy measurements and inference model. For instance, in PPI networks, since the interactions (edges) are derived through noisy and error-prone experiments, each edge is associated with an uncertainty value [32]. In large social networks, uncertainty arises for various reasons [1, 28]. The edge probability may denote the *influence* of one person on another [28]. Incorporating uncertainty to graphs lead to uncertain graphs (also called probabilistic graphs), whose edges are labeled with a probability of existence. This probability represents the confidence that the relation (the edge) holds in reality. The use of such information can enhance the effectiveness of graph mining algorithms because the uncertainty provides a guidance in the use of different possible worlds during the mining process. Given the wide spectrum of application domains, querying and mining uncertain graphs has received considerable attention recently.

Various agencies and data owners are increasingly sharing their uncertain graphs with third-party analysts, which immediately raises the big concern of data privacy. An adversary may leverage any available auxiliary information, e.g., node degrees, neighborhoods of breached nodes, to reveal the identities of the nodes in the release graph [24]. Such privacy vulnerability is a big barrier preventing the sharing, analytics, and the extraction of useful insights from these graphs.

Before publishing, the graph needs to be anonymized to prevent potential re-identification attacks. Numerous graph anonymization techniques have been proposed in the literature to address this challenge [34, 11, 37, 40]. These anonymization techniques aim at minimizing the disclosure risk of private information by modifying graph at some level. However, these conventional methods have mainly focused on *deterministic graphs*, where the edges' presence is known with certainty. In contrast, in uncertain graphs, the edges carry probabilities of their existence, which makes the privacy-preservation problem far more different because of the incorporation of uncertainty in the de-anonymization process.

Compared to deterministic graphs, the publishing of uncertain graphs reveals **additional** information–associated edge uncertainty which can be used by the adversary to re-identify some entities in the released uncertain graph. To gain more intuition on uncertain graph model and the difference in structural privacy attack, we present a simple example. Consider the release uncertain graph shown in Figure 1.1. Each edge is associated with the probability of being present. Possible instantiations of this graph are commonly referred to as *worlds*. In our example, there are $2^5 = 32$ possible worlds. The probability of a world can be calculated based on the probability of its edges as shown in the example in Figure 1.1. Suppose the adversary has the external information, *i.e.*, *Ana* has a degree of 3 with high probability in the original graph. After the calculation of

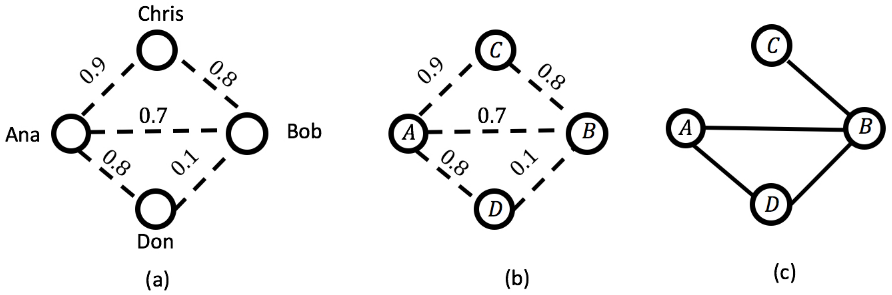


Figure 1.1: (a) The original uncertain graph (b) The naive anonymized version of uncertain graph with 5 edges and $2^5 = 32$ possible world (c) One possible world: $G = \{(AB), (AD), (BC), (BD)\}$. The probability of G is $Pr(G) = p(A, B)(1 - p(A, c))p(A, D)p(B, C)p(B, D) = 0.00448$.

the probability that nodes has the degree of 3 in the release uncertain graph as shown in the example in Figure 1.1, the adversary could know that posterior probability that *Ana* is possible node a is around 0.9 and is node d is around 0.1. Clearly, it leads to the disclosure of node identity. To best of our knowledge, uncertain graph anonymization problem remains unexplored.

1.2 State-Of-the-Art

1.2.1 Distributed Triangle Listing Algorithms

Triangle listing is a basic operation of the graph analysis. Many research works have been conducted on this problem, which can be classified into three categories: in-memory algorithms, external-memory algorithms and distributed algorithms. Here, we briefly review these works.

In-Memory Algorithm. The majority of previously introduced triangle listing algorithms are the In-Memory processing approaches. Traditionally, they can be further classified as Node-Iterator[2, 7, 46] and Edge-Iterator ones[26, 15] with the respect to iterator-type. Authors [26, 15, 46]improved the performance of in-memory algorithms by adopting degree-based ordering. Matrix multiplication is used to count triangles [2]. However, all these algorithms are inapplicable to massive graphs which do not fit in memory.

External-Memory Algorithms. In order to handle the massive graph, several external-memory approaches were introduced [25, 31, 33]. Common idea of these methods is: (1) Partition the input graph to make each partition fit into main-memory, (2) Load each partition individually into main-memory and identify all its triangles, and then remove edges which participated in the identified triangle, and (3) After the whole graph is loaded into memory buffer once, the remaining edges are merged, then repeat former steps until no edges remain. These Algorithms require a lot of disk I/Os to perform the reading and writing of the edges. Authors [25, 31] improved the performance by reducing the amount of disk I/Os and exploiting multi-core parallelism. External-Memory Algorithms show great performance in time and space. However, the parallelization of external-memory algorithms is limited. External-memory approaches cannot easily scale up in terms of computing resources and parallelization degree.

Distributed Algorithms. Another promising approach to handle triangle listing on large-scale graphs is the distributed computing. Suri *et al.* [48] introduced two MapReduce adaptions of NodeIterator algorithm and the well-known Graph Partitioning (GP) algorithm to count triangles. The Graph Partitioning algorithm utilizes one universal hash partition function over nodes to distribute edges into overlapped graph partitions, then identifies triangles over all the partitions. Park *et al.* [43] further generalized Graph Partitioning algorithm into multiple rounds, significantly increasing the size of the graphs that can be handled on a given system. The authors compare their algorithm with GP algorithm [48] across various massive graphs then show that they get speedups ranging from 2 to 5. In this work, we show such large or even larger speedup (from 5 to 10) can also be obtained by reducing the size intermediate result directly via our methods. Teixeira *et al.* [49] presented Arabesque, one distributed data processing platform for implementing subgraph mining algorithms on the basis of MapReduce framework. Arabesque automates the process of exploring a very large number of subgraphs, including triangles. However, these MapReduce algorithms must generate a large amount of intermediate data that travel over the network during the shuffle operation, which degrade their performance. Arifuzzaman *et al.* [3] introduced an efficient MPI-based distributed memory parallel algorithm (Patric) on the basis of NodeIterator algorithm. The Patric algorithm introduced degree-based sorting preprocessing step for efficient set intersection operation to speed up execution. Furthermore, several distributed solutions designed for subgraph mining on large graph were also proposed [35, 47]. Shao *et al.* introduced the PSgl framework to iteratively enumerate subgraph instance. Different from other parallel approaches, the PSgl framework completes relies on the graph traversal and avoids the explicit join operation. These distributed memory parallel algorithms achieve impressive performance over large-scale graph mining tasks. These methods distributed the data graph among the worker's memory, thus they are not suitable for processing large-scale graph with small clusters.

1.2.2 Graph Anonymization

Uncertain graph anonymization problem is related to the conventional graph anonymization problem. Several definitions and methods have been proposed to protect users' privacy when publicly releasing graph data. Here, we focus on graph-modification techniques which alter graph's structure and release the entire anonymous network, allowing researchers and third parties to apply all graph-mining process on anonymous data, from local to global knowledge extraction.

Note that for defining the problem of privacy preserving graph publishing, we need to formulate the following issue: firstly, we need to identify information to be preserved. Secondly, we need to model the background knowledge that an adversary may use to attack the privacy. And thirdly, we need to specify the usage of the published graph data so that an anonymization method can try to retain the utility as much as possible while the privacy information is fully preserved.

Regarding the privacy information to be preserved in the graph data, three main categories of privacy threats have been identified:

1. *Identity disclosure* occurs when the identity of an individual who is associated with a vertex is revealed. It includes sub-categories such as vertex existence, vertex properties, and graph metrics.

2. *Attribute disclosure* which seeks not necessarily to identify a vertex, but to reveal sensitive labels of the vertex.

3. *Link disclosure* when the sensitive relationship between two individuals is disclosed. Depending on graph's type, we can refine this category as link relationships, link weight, and sensitive edge labels.

Identify disclosure often lead to attribute disclosure due to that fact that identity disclosure occurs when an individual is identified within a dataset, whereas attribute disclosure occurs when sensitive information that an individual wished to keep private is identified.

Determining the knowledge of the adversary is a challenging problem. A variety of adversaries' knowledge has been proposed in conjunction with their attack and a protection method. Attacks on naively anonymized network data have been developed, which can reidentify vertices, disclose edges between vertices. These attacks include matching attacks, which use external knowledge of vertex features [34, 53, 11, 58]; injection attacks which alter the network prior to publication [4]; and auxiliary network attacks which use publicly available networks as an external information source [39]. To solve these problems, methods which introduce noise to the original data have been developed in order to hinder the potential process of re-identification.

Graph modification techniques

From a high-level view, there are three general families of graph-modification techniques to mitigate graph data privacy:

- *Generalization or clustering-based approaches* which can be essentially regarded as grouping vertices and edges into partitions called super-vertices and super-edges. The details about individuals can be hidden properly, but the graph may be shrunk considerably after anonymization, which may not be desirable for analyzing local structures.
- *Edge and vertex modification* approaches first transform the data by edges or vertices modification (adding and/or deleting) and then release the perturbed data. The data is thus made available for unconstrained analysis with existing graph mining techniques.
- *Uncertain graphs* are approaches based on adding or removing edges “partially" by assigning a probability to each edge in the anonymous network. Instead of creating or deleting edges, the set of all possible edges is considered and a probability is assigned to each edge.

All mentioned methods first transform the data into different types of graph’s modifications and then release the perturbed data. The data is thus made available for unconstrained analysis. On the contrary, there is “privacy preserving graph mining" methods, which do not release data, but only the output of an analysis task. For instance, differential privacy [27] is a well-known privacy preserving graph mining method. In our work, we do not consider this method for anonymizing uncertain graphs, since they do not allow us to release the entire network, which provides the widest range of applications for data mining and knowledge extraction.

- **Generalization approaches**

Generalization approaches can be essentially regarded as grouping vertices and edges into partitions called super-vertices and super-edges. The details about individuals can be hidden properly, but the graph may be shrunk considerably after anonymization, which may be not desirable for analyzing local structures. All methods developed, therefore, need the whole graph to be applied to. Consequently, they are not able to deal with the streaming graph data. Here, we remind the reader new methods can be developed using this core idea to generate anonymous graph dataset. The first approach of this category was proposed by Hay *et al.* [24]. It uses the size of the partition to ensure node anonymity. After grouping, each super-vertex represents at least k nodes and each super-edge represents all the nodes between nodes in two super-vertices. Only the edge density is published for each partition, so it will be hard to distinguish between individuals in a partition. A similar idea was applied for the complex network, *i.e.*, labeled network [10].The clustering problem is known to be NP-hard. Researchers ever present different methods for optimization. For instance, Sihag *et al.* chose the genetic algorithm to optimize this NP-hard problem. It does achieve a better result in terms of information loss. Unfortunately, this method does not seem scalable for large networks.

- **Edge and vertex modification approaches**

Edge and vertex modification approaches anonymize a graph by modifying edges or vertices in the graph. These modifications can be made at random (referred to as *randomization, random perturbation*). *random perturbation* techniques are generally the simplest and present the lowest complexity. Thus, they are able to deal with large networks. The first method was proposed by Hay *et al.*, called *Random perturbation* which anonymizes unlabelled graphs using Rand add/del strategy, *i.e.*, randomly removing p edges and then randomly add p fake edges without change the set of vertices and the total number of edges. On this basis, Ying and Wu [56, 57] developed two algorithms designed to preserve spectral characteristics of the original graph called *Spctr Add/Del* and *Spctr Switch*. Following the path, Stokes and Torra states an appropriate selection of the eigenvalues in the spectral method can perturbation the graph while keeping its most significative edges. The generic strategy which aims to preserve the most important edges in the network trying to maximize utility while achieving the desired privacy level. Generally, such utility-aware methods achieve lower information loss, but at a cost of increasing complexity. Another improved variation of *Random perturbation* was proposed by Ying *et al.* [56], called *Blockwise Random Add/Del*. This method divides the graph into blocks according to the degree sequence and implements edge modifications on the vertices at high risk of re-identification, not at random over the entire set of vertices. However, the ever-mentioned random perturbation techniques do not offer privacy guarantee.

The modification can be performed in order to fulfill some desired constraints (referred to as *constrained perturbation methods*). Among them, the k-anonymity model is the most well-known privacy notation which imported from relational data anonymization. The k-anonymity model indicates that an attacker can not distinguish between k records although he manages to find a group of quasi-identifiers. Therefore, the attacker can not re-identify an individual with a probability greater than $\frac{1}{k}$. The concept can be used as quasi-identifier to extend k-anonymity on the graph data such as k-degree anonymity [34], k-symmetry [53], and k-automorphism [59].

Constrained graph modification based on modifying the graph structure (by edge modifications) to ensure all the vertices satisfy k-anonymity. The first method was proposed by Liu and Terzi [34] which based on integer linear programming and edge switch to construct a new anonymous graph which is k-degree anonymous. Hartung *et al.* [23] showed k-degree anonymity becomes NP-hard on graphs with H-index three, which is a quite common case for large networks. Different kinds of heuristics were proposed to improve over Liu and Terzi's work in terms of speed and scalability [38, **?**]. For instance, Nagle *et al.* [38] proposed a local anonymization algorithm based on k-degree anonymity that focuses on obscuring structurally important vertices that are not well anonymized, thereby reducing the cost of the overall anonymization procedure. However, results are similar to Liu and Terzi's algorithm in terms of information loss. Namely, they all suffer from the high utility low bound.

- **Uncertain graphs**

Rather than anonymization graphs by generalized them or adding/removing edges to satisfy privacy parameter, recent methods have explored the semantics of uncertain graphs to achieve privacy protection. The first approach was proposed by Boldi *et al.* [11]. It is based on injecting uncertainty in deterministic graphs and publishing the resulting uncertain graphs. The authors notice that from a probabilistic perspective, adding a non-existing edge corresponds to changing its existence probability from 1 to 0 vice versa. In their method, instead of considering only binary edge probabilities, they allow probabilities to take any value in the range $[0, 1]$. From the perspective of graph modification, they provide more gained way “partial Add/Del Edge" to transform the input graph to the anonymous one thereby reduce the information loss in the anonymization procedure. However, the specific method ignores several opportunities for further reducing information loss in the anonymization procedure. Nguyen *et al.* [40] proposed a generalized obfuscation model based on uncertain adjacency matrices that keep expected node degrees equals to those in the original graph, and a generic framework for privacy and utility quantification of anonymization methods. The same authors present another method based on maximum variance to achieve better trade-off privacy and data utility (referred to as *MaxVar*). In particular, they transform the optimization problem into independent quadratic optimization problems by dividing the large input graph into subgraphs. From the view of graph modification, they provide more subtle way “partial Switch Edge" for anonymizing the input graph thereby achieve the better trade-off between privacy and utility. However, *MaxVar* fails to provide meaning privacy guarantee for user tunable purpose. What’s more, these two methods assumes each edge modification has the equal impact over the graph. However, it is not always the case, especially in large networks.

1.3 Challenges

Although many effective graph anonymization techniques have been proposed for deterministic graphs, it is challenging to shift existing graph anonymization techniques to work over uncertain graphs.

It is challenging to develop a proper metric for quantifying the information loss for uncertain graph anonymization. Fundamentally, the graph anonymization techniques required modifying graph structure at some level. The intermediate goal of graph anonymization is to balance the utility and privacy. The first question we need to solve is to develop proper metrics for quantifying loss of information. In the context of deterministic graphs, this problem has been extensively studied. Most of the previous works use the total number of modified edges to measure the utility loss [34, 11]. Researchers argued this measure is not effective as it assumes each edge modification has an equal impact on the original graph properties [51]. They suggest studying the change on other structural properties such as the spectrum [57], community structure [51], shortest path length and the neighborhood overlap [41]. However, above-mentioned metrics are all designed for comparing deterministic graphs and can not be used to handle uncertain graphs directly. Thus, we need to investigate other utility metrics suitable for uncertain graphs which are able to capture the essence of structural properties for better serving a wide range of analytics.

It is challenging to define the adversary knowledge and privacy protection model which explicitly incorporates edge uncertainty. Compared to deterministic graphs, the publishing of uncertain graphs reveals additional information–associated edge uncertainties which can be used by the adversary to re-identify some entities in the released uncertain graph. The available edge uncertainty in the released uncertain graph can be used to enhance various kinds of de-anonymization attack. Together with edge uncertainty, they become newly identified attacks which are different from existing ones. The theoretical formulation is needed. The second question we need to solve is to model how the adversary incorporates edge uncertainty into the de-anonymization process, namely Attack Model. In this work, we focus on structural attack. Usually, structural attacks perform in the following way. Let G be a graph and $\acute{G}$ be the anonymized version of the given graph. The adversary locates the match vertices in $\acute{G}$ according to the structural information about the target in G. If there is a limited number of answers, it may lead to target node re-identification and to privacy infringement. In the case of deterministic graphs, the structural information of a target in G is an assertion with certainty, such as "Ana has 3 neighbors" and the matching assertion can be evaluated as True or False with certainty. In the case of uncertain graphs, it becomes more complex. The structural property of a target node v in an uncertain graph is defined as a set of observations over all the possible worlds. First, depending on the domain of real applications, the adversary may have a complete and exact knowledge or only the aggregated statistics with respect to the structural property. Let the property be node degree, the adversary may assess the exact degree distribution or only the global statistic. Different adversary knowledge triggers different matching attack. Second, the matching process which links the nodes in the perturbed graph with the collected structural information (complete distribution or expected value) performs in a different way. We need to extend the concept of matching attacks in uncertain graphs with different kinds of adversary knowledge, and then design a proper privacy model on the basis of the matching evaluation.

It is challenging to design an effective and efficient uncertain graph techniques. Given an input graph G and allowed operations O, the task of graph anonymization is to transform G into the anonymous one by performing as few operations as possible. The problem is known to be NP-hard when the input graph G is deterministic one and graph modification operations includes edge addition and edge deletion [23]. The complexity of uncertain graph anonymization problem falls into the same category. Although many graph anonymization methods have been proposed, they can not be used for uncertain graph directly. The shifting of existing methods need consider uncertainty explicitly in each aspect. For example, we need to design a set of heuristic-based techniques to make the anonymization mechanism efficient meanwhile preserving data utilities in the context of uncertain graphs.

Chapter 2

Preliminaries

In my dissertation, I will study triangle listing problem for web-scale graphs and anonymization techniques in the context of uncertain graphs. For the later topics, we focus on resisting the degree-based node re-identification. Here, we give the definitions of uncertain graph model and attack model, privacy criteria related with uncertain graph anonymization problem. We also give the formal formulation of proposed problems.

2.1 Distributed Triangle Listing

2.1.1 Triangle Listing Problem

Suppose we have a simple undirected graph $G(V, E)$, where V is the set of vertices (nodes), and E is the set of edges. Let $n = |V|$ and $m = |E|$. Let $N_v = \{u|(u, v) \in E\}$ denote the set of *adjacent nodes* of node v, and $d_v = |N_v|$ denote the degree of node v. We assume that G is stored in the most popular format for graph data, *i.e.*, the adjacency list representation. Given any three distinct vertices $u, v, w \in V$, they form a triangle $\triangle_{uvw}$, iif $(u, v), (u, w), (v, w) \in E$. We define the set of all triangles that involve node v as $\triangle(v) = \{\triangle_{uvw} |\, (v, u), (v, w), (u, w) \in E\}$. Similarly, we define $\triangle(G) = \bigcup_{v \in V} \triangle(v)$ as the set of all triangles in G.

Problem 1 ***Triangle Listing Problem****: Given a large-scale distributed graph $G(V, E)$, our goal is to report all triangles in G, i.e., $\triangle(G)$, in a highly distributed way.*

2.1.2 Sequential Triangle Listing

Algorithm 1 NodeIterator++

```
Preprocessing step
for all $(u, v) \in E$ do
    if $u \succ v$, store $u$ in $N_v^H$
    else store $v$ in $N_u^H$
end for
Triangle Listing
$\triangle(G) \leftarrow \emptyset$
for all $v \in V$ do
    for all $u \in N_v^H$ do
        for all $w \in N_v^H \bigcap N_u^H$ do
            $\triangle(G) \leftarrow \triangle(G) \bigcup \{\triangle_{vuw}\}$
        end for
    end for
end for
```

Sequentail triangle listing algorithms have been extensively stuidied. Here, we present a sequential triangle listing algorithm which is widely used as the basis of parallel approaches [48, 3, 43]. In this work, we also use it as the basis of our distributed approach.

A naive algorithm for listing triangles is as follows. For each node $v \in V$, find the set of edges among its neighbors, i.e., pairs of neighbors that complete a triangle with node v. Given this simple method, each triangle (u, v, w) is listed six times—all six permutations of u, v and w. Several other algorithms have been proposed to improve on and eliminate the redundancy of this basic method, e.g., [46, 8].

One of the algorithms, known as *NodeIterator++* [46], uses a total ordering over the nodes to avoid duplicate listing of the same triangle. By following a specific ordering, it guarantees that each triangle is counted only once among the six permutations. Moreover, the NodeIterator++ algorithm adopts an interesting node ordering based on the nodes' degrees, with ties broken by node IDs, as defined blow:

$$u \succ v \Longleftrightarrow d_u > d_v \; or \; (d_u = d_v \; and \; u > v)$$

This degree-based ordering improves the running time by reducing the diversity of the effective degree $\hat{d_v}$. The running time of NodeIterator++ algorithm is $O(m^{3/2})$. A comprehensive analysis can be found in [46].

The standard *NodeIterator++* algorithm performs the degree-based ordering comparison during the final phase, i.e., the triangle listing phase. The work in [3] and [48] further improves on that by performing the comparison $u \succ v$ for each edge $(u, v) \in E$ in the preprocessing step (Lines 1-3, Algorithm 1). For each node v and edge (u, v), node u is stored in the effective list of v (N_v^H) if and only if $u \succ v$, and hence $N_v^H = \{u : u \succ v \; and \; (u, v) \in E\}$. The preprocessing step cuts the storage and memory requirement by half since each edge is stored only once. After the preprocessing step, the effective degree of nodes in G is $O(\sqrt{m})$ [46]. Its correctness proof can be found in [3]. The modified NodeIterator++ algorithm is presented in Algorithm 1.

2.1.3 MapReduce Overview

MapReduce is a popular distributed programming framework for processing large datasets [18]. MapReduce, and its open-source implementation Hadoop [52], have been used for many important graph mining tasks [48, 43]. In this paper, our algorithms are designed and analyzed in the MapReduce framework.

Computation Model. An analytical job in MapReduce executes in two rigid phases, called the *map* and *reduce* phases. Each phase consumes/produces records in the form of *key-value* pairs—We will use the keywords *pair*, *record*, or *message* interchangeably to refer to these key-value pairs. A pair is denoted as $\langle k; val \rangle$, where k is the key and val is the value. The *map* phase takes one key-value pair as input at a time, and produces zero or more output pairs. The *reduce* phase receives multiple key-listOfValues pairs and produces zero or more output pairs. Between the two phases, there is an implicit phase, called *shuffling/sorting*, in which the mappers' output pairs are shuffled and sorted to group the pairs of the same key together as input for reducers.

Our proposed solution will leverage and extend some of the basic functionality of MapReduce, which are:

- **Key Partitioning:** Mappers employ a *key partitioning function* over their outputs to partition and route the records across the reducers. By default, it is a hash-based function, but can be replaced by any other user-defined logic.

- **Multi-Key Reducers:** Typically, the number of distinct keys in an application is much larger than the number of reducers in the system. This implies that a single reducer will sequentially process multiple keys—along with their associated groups of values—in the same reduce instance. Moreover, the processing order is defined by *key sorting function* used in shuffling/sorting phase. By default, a single reduce

Algorithm 2 MR-Baseline

```
    **Map**: Input: $\langle v; N_v^H \rangle$
 1: emit $\langle v; (v, N_v^H) \rangle$
 2: **for all** $u \in N_v^H$ **do**
 3:     emit $\langle u; (v, N_v^H) \rangle$
 4: **end for**

    **Reduce**:Input:$[\langle u; (v, N_v^H) \rangle]$
 5: initiate $N_u^H$
 6: **for all** $\langle u; (v, N_v^H) \rangle$ **do**
 7:     **for all** $w \in N_u^H \cap N_v^H$ **do**
 8:         emit $\triangle_{vuw}$
 9:     **end for**
10: **end for**
```

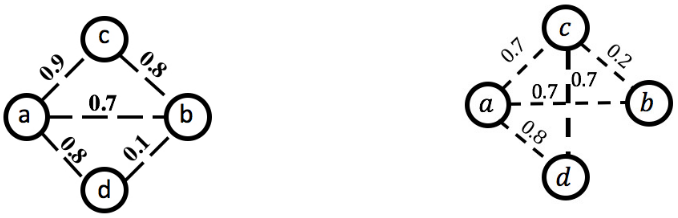
instance processes each of its input groups in total isolation from the other groups with no sharing or communication.

2.1.4 Triangle Listing in MapReduce

Both [48] and [3] use the NodeIterator++ algorithm as the basis of their distributed algorithms.[48] identifies the triangles by checking the existence of pivot edges, while [3] uses set intersection of effective adjacency list (Line 7, Algorithm 1). In this section, we present the MapReduce version of the NodeIterator++ algorithm similar to the one presented in [3], referred to as *MR-Baseline* (Algorithm 2).

The general approach is the same as in the NodeIterator++ algorithm. In the map phase, each node v needs to emit two types of messages. The first type is used for the initiation its own effective adjacency list in the reduce side, referred to as *a core message* (Line 1, Algorithm 2). The second type is used for identifying triangles, referred to as *pivot messages* (Lines 2-3, Algorithm 2). All pivot messages from v to its effective adjacent nodes are identical. In the reduce phase, each node u will receive a core message from itself, and a pivot message from adjacent nodes with the lower degree. Then, each node identifies the triangles by performing a set intersection operation (Lines 5-6, Algorithm 2).

We omit the code of the pre-processing procedure since its implementation is straightforward in MapReduce. In addition, we will exclude the pre-processing cost for any further consideration since it is typically dominated by the actual running time of the triangle



(a) An *original* graph. (b) An *anonymization*.

Figure 2.1: Uncertain graph anonymization.

listing algorithm, plus it is the same overhead for all algorithms.

2.2 Privacy Protection on Uncertain Graphs

2.2.1 Uncertain Graph

Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, where V is the set of nodes, E is the set of edges, and function $p : E \rightarrow [0, 1]$ assigns a probability of existence to each each edge, denoted as $p(e)$. We consider the edge probabilities are independent, and we assume *possible world* semantics, consistent with existing literature [45, 1, 28]. Specifically, the *possible world* semantics interprets $\mathcal{G}$ as a set of possible deterministic graphs $W(\mathcal{G})$, each deterministic graph $G \in W(\mathcal{G})$ including all vertices of $\mathcal{G}$ and a subset of edges $E_G \subset E$. The probability of observing any possible world $G = (V, E_G) \in W(\mathcal{G})$ is

$$Pr[G] = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e))$$

We can think of the uncertain graph $\mathcal{G}$ as a world generator process, and each graph $G \in W(\mathcal{G})$ as a possible world.

2.2.2 Attack Model

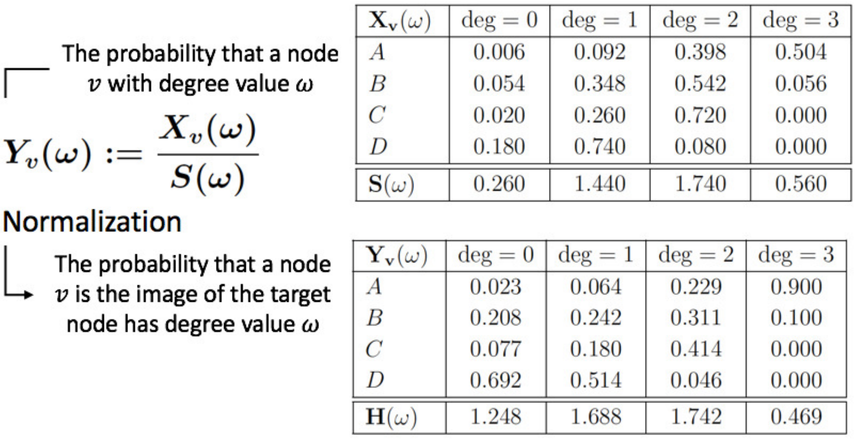
Uncertain graph anonymization problem is related to the conventional graph anonymization problem, which has been extensively studied. As Hay *et al.* pointed out, the structural information such as node degree, 1-neighbood can be utilized to breach user privacy. For example, [11, 34, 50] consider the case that a vertex can be re-identified by its degree, while [17, 53] consider the case that a vertex can be re-identified by degrees of it and its neighbood. Different anonymization techniques are designed to resist different kinds of privacy attack.

The first step to anonymization is to know what external information of a graph may be acquired by an adversary. In our work, we assume that the information of victim node degree is easy to be collected by the adversary, consistent to the literture [34, 50, 11, 53, 17]. In fact, degree-based de-anonymization is one of the most serious privacy attacks in the context of deterministic graphs. The knowledge could also be understood as some *assertion* of the individual, which could be evaluated to *true* or *false* based on the topology structure of the network.

In the context of deterministic graphs, such knowledge can be expressed as one assertion with certainty such as *A*na has 3 neighbors. In the context of uncertain graphs, such knowledge can be a little different. Given an uncertain graph $\mathcal{G}$, let the degree of a fixed node v to be d_v, clearly the observation of d_v differs on the possible worlds $G \in W(\mathcal{G})$. We define the d_v as a random variable with a known distribution. We base our degree-related adversary knowledges on standard stastical measures of the distribution.

It is difficult to model the specific degree-related knowledge used by the adversaries by advance. In our work, we first consider the adversary can and only acquire the expected node degree of victim nodes. For example, the adversary knows that the expected value of the number of Ana's neighbors should 3 as shown in Figure 2.1(a). Later, we consider the adversary can acquire more comprehensive knowledge of node degree of victim nodes such as the distribution function. For example, the adversary may know the distribution of the number of Ana's neighbors as $(3 : 0.504), (2 : 0.398), (1 : 0.092), (0 : 0.006)$.

The second step to anonymization is to know how the adversary links the external knowledge such as node degree to a node in the perturbed graph. In the deterministic graph, the process of node de-anonymization with the external node degree information is quite straightforward. Let $\mathcal{G}$ be a graph and $\acute{\mathcal{G}}$ be the anonymized version of the given graph. The adversary can get a number of `match` vertices in $\acute{\mathcal{G}}$, *i.e.*, the nodes U_c with the `match` degree. When $\acute{\mathcal{G}}$ is a deterministic one, for a given node in $\acute{\mathcal{G}}$, the assertion is evaluated either *true* or *false* with certainty. When the anonymized output $\acute{\mathcal{G}}$ is probabilistic one, the matching attack is performed over all the possible worlds. Following the path, the matching event would be observed with an probability value which defined in a continous range $(0, 1)$. For example, the probability that the node a was observed with degree 3 is 0.504 in Figure 2.1(a). We consider the matching attack is performed between a random variable, namely, find the equal random variable. Namely, for a given node u


The probability that a node v with degree value ω
$Y_v(\omega) := \frac{X_v(\omega)}{S(\omega)}$
Normalization
The probability that a node v is the image of the target node has degree value ω

$\mathbf{X_v}(\omega)$	$\deg = 0$	$\deg = 1$	$\deg = 2$	$\deg = 3$
A	0.006	0.092	0.398	0.504
B	0.054	0.348	0.542	0.056
C	0.020	0.260	0.720	0.000
D	0.180	0.740	0.080	0.000
$\mathbf{S}(\omega)$	0.260	1.440	1.740	0.560

$\mathbf{Y_v}(\omega)$	$\deg = 0$	$\deg = 1$	$\deg = 2$	$\deg = 3$
A	0.023	0.064	0.229	0.900
B	0.208	0.242	0.311	0.100
C	0.077	0.180	0.414	0.000
D	0.692	0.514	0.046	0.000
$\mathbf{H}(\omega)$	1.248	1.688	1.742	0.469

Figure 2.2: The degree uncertainty for each node (top) and normalized values for each degree (bottom)

in $\acute{\mathcal{G}}$, it is defined as the probability that the random variable d_u is equal to the adversary knowledge d_v, denotes as $P(d_u = d_v)$. It can be computed as

$$P(d_u = d_v) = \sum_{i \in Z} P(d_u = i) P(d_v = i)$$

2.2.3 Privacy Criteria

As ever discussed, we want to publish the anonymized version of uncertain graphs where node identity is obfuscated enough. We adopt (k, ϵ)-*obf* model, proposed in [12], to quantify the anonymity level achieved by an anonymized graph (uncertain graph). Its formal definition is as follow:

Definition 1 ***(k, ϵ)-obf [12]*** *Let P be a vertex property, $k \geq 1$ be a desired level of anonymity, and $\epsilon > 0$ be a tolerance parameter. The uncertain graph $\tilde{\mathcal{G}}$ is said to k-obfuscate a given vertex $v \in V_{\mathcal{G}}$ with respect to P if the entropy of the distribution $Y_{P(v)}$ over the vertices of $\mathcal{G}$ is greater than or equals to $\log_2 k$:*

$$H(Y_{P(v)}) \geq \log_2 k.$$

The uncertain graph $\mathcal{G}$ is (k, ϵ)-obf with respect to property P if it k-obfuscates at least $(1 - \epsilon)|V|$ vertices in $V_{\mathcal{G}}$. P can be any node properties.

Figure 2.2 gives an example of how to compute degree entropy for the uncertain graph *in Figure 2.1(a). Each row in the top side is the degree distribution for the corresponding node. For example, node a has degree 0 with probability $(1-0.9)(1-0.8)(1-0.7) = 0.006$. The last row in the top side is the sum of probability over all the nodes. For example, $S(0) = 0.006 + 0.054 + 0.020 + 0.180 = 0.26$. The bottom one normalizes values in each column to get distribution $Y_{P(v)}$. For example, $Y_0(a)$= $\frac{X_0(a)}{S(0)} = 0.023$. The entropy $H(Y_d(v))$ for each degree value ω is shown in the last bottom row. Given $k = 3, \log_2 3 = 1.58$, node b, c with expected degree 2 and d with true degree 1 satisfies k-obf while a with expected degree 3 fails, thus $\epsilon = 0.25$.*

2.2.4 Reliability-based Utility Criteria

Inspired by the importance of reliability, we suggest to quantify the utility loss (anonymization cost) as the reliability difference between the anonymized result $\tilde{\mathcal{G}}$ and the original one $\mathcal{G}$, reliability discrepancy. We proceed to explain its formal definition.

In the context of uncertain graphs, *reliability* generalizes the concept of connectivity by capturing the probability that two given (sets of) nodes are reachable over all possible worlds. We focus on two-terminal reliability.

Definition 2 ***Two-Terminal Reliability [16]*** *Given an uncertain graph $\mathcal{G}$, and two distinct nodes u and v in V, the reliability for two nodes (u, v) is defined as follows:*

$$R_{u,v}(\mathcal{G}) = \sum_{G \subseteq W(\mathcal{G})} \mathcal{I}_G(u, v) Pr[G]$$

where $\mathcal{I}_G(u, v)$ is 1 when u and v are contained in a connected component in G, and 0 otherwise.

Definition 3 ***Two-Terminal Reliability Discrepancy*** *The reliability discrepancy of two distinct nodes (u, v) is defined as the absolute difference of corresponding reliability values in the original graph $\mathcal{G}$ and the anonymized result $\tilde{\mathcal{G}}$. Its formal definition is as follows:*

$$RD_{u,v}(\tilde{\mathcal{G}}) = |R_{u,v}(\mathcal{G}) - R_{u,v}(\tilde{\mathcal{G}})|$$

Thus, we come up the definition of reliability discrepancy of the overall anonymized result against the true graph.

Definition 4 ***Reliability Discrepancy*** *Compared to the input uncertain graph $\mathcal{G} = (V, E, p)$, the reliability discrepancy of one anonymized instance $\tilde{\mathcal{G}} = (V, E, \tilde{p})$ is the sum of RD*

over all vertex pair (u, v). The formal definition is as follows:

$$\Delta(\tilde{\mathcal{G}}) = \sum_{(u,v)} RD_{u,v}(\tilde{\mathcal{G}})$$

Example 1 *Given an input uncertain graph $\mathcal{G}$ in Figure 2.1(a) and its anonymization $\tilde{\mathcal{G}}$ in Figure 2.1(b) and suppose one wants to compute $RD_{a,b}(\tilde{\mathcal{G}})$. Following the definition, the reliability of node pairs (a, b) for $\mathcal{G}$ in Figure 2.1(a) equals 0.92272. While, the* reliability *of node pairs in Figure 2.1(b) equals* 0.75208*. Thus, the* reliability discrepancy *of node pair $RD_{a,b}(\tilde{\mathcal{G}})$ is* $|0.92272 - 0.75208| = 0.17064$.

We are ready to formulate the problem, reliability preserving anonymization in the context of uncertain graphs.

Problem 2 ***Reliability Preserving Anonymization over Uncertain graphs*** *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and anonymization parameters k, ϵ, the objective is to find a possible (k, ϵ)-obfuscated graph $\tilde{\mathcal{G}} = (V, E, \tilde{p})$ with minimal reliability discrepancy $\Delta(\tilde{\mathcal{G}})$, as*

$$\begin{array}{ll} \underset{\tilde{\mathcal{G}}}{\operatorname{argmin}} & \Delta(\tilde{\mathcal{G}}) \\ \textit{Subject to} & \tilde{\mathcal{G}} \textit{ is } (k, \epsilon) - obf \end{array}$$

Chapter 3

Distributed Triangle Listing Algorithm

In this chapter, I will briefly review the distributed triangle listing techniques I developed for MapReduce Framework. These techniques have been published as a conference paper [55]. In this work, we present Bermuda method which effectively reduces the size of the intermediate data via redundancy elimination and sharing of messages whenever possible. As a result, Bermuda achieves orders-of-magnitudes of speedup and enables processing larger graphs that other techniques fail to process under the same resources. Bermuda exploits the locality of processing, i.e., in which reduce instance each graph vertex will be processed, to avoid the redundancy of generating messages from mappers to reducers. Bermuda also proposes novel message sharing techniques within each reduce instance to increase the usability of the received messages.

3.1 Overview

With the MR-Baseline Algorithm, one node needs to send the same pivot message to multiple nodes residing in the same reducer. Therefore, the network traffic can be reduced by sending only one message to the destination reducer, and either have main-memory cache or distributing the message to actual graph nodes within each reducer. Although this strategy seems quite simple, and other systems such as GPS and X-Pregel [6, **?**] have implemented it, the trick lies on how to efficiently perform the caching and sharing. In this section, we propose new effective caching strategies to maximize the sharing benefit while encountering little overhead. We also present novel theoretical analysis for the proposed techniques.

In the frameworks of GPS and X-Pregel, adjacency lists of high degree nodes are used for identifying distinct destination reducer and distributing the message to target nodes in the reduce side. This method requires extensive memory and computations for message sharing. In contrast, in Bermuda, each node uses the *universal key partition function* to group its destination nodes. Thus, each node would only send the same pivot message to each reduce instance only once. At the same time, reduce instances will adopt different message-sharing strategies to guarantee the correctness of algorithm. As a result, Bermuda achieves a trade off between reducing the network communication—which is known to be a big bottleneck for map-reduce jobs—and increasing the processing cost and memory utilization.

3.2 Bermuda Edge-Centric Node++

A straightforward (and intuitive) approach for sharing the pivot messages within each reduce instance is to organize either the pivot or core messages in main-memory for efficient random access. We propose the Bermuda Edge-Centric Node++ (Bermuda-EC) algorithm, which is based on the observation that for a given input graph, it is common to have the number of core messages smaller than the number of pivot messages. Therefore, the main idea of Bermuda-EC algorithm is to first read the core messages, cache them in memory, and then stream the pivot messages, and on-the-fly intersect the pivot messages with the needed core messages (See Figure 3.1). The MapReduce code of the Bermuda-EC algorithm is presented in Algorithm 3.

In order to avoid pivot message redundancy, a universal key partitioning function is utilized by mappers. The corresponding modification in the map side is as follows. First, each node v employs a universal key partitioning function $h()$ to group its destination nodes (Line 3, Algorithm 3). This grouping captures the graph nodes that will be processed by the same reduce instance. Then, each node v sends a pivot message including the information of N_v^H to each non-empty group (Lines 4-6, Algorithm 3). Following this strategy, each reduce instance receives each pivot message exactly once even if it will be referenced multiple times.

Moreover, we use tags to distinguish core and pivot messages, which are not listed in the algorithm for simplicity. Combined with the MapReduce internal sorting function, Bermuda-EC guarantees that all core messages are received by the reduce function before any of the pivot messages as illustrated in Figure 3.1. Therefore, it becomes feasible to cache only the core messages in memory, and then perform the intersection as the pivot

Algorithm 3 Bermuda-EC

```
    Map: Input:($\langle v; N_v^H \rangle$)
    Let $h(.)$ be a key partitioning function into [0,k-1]
 1: $j \leftarrow h(v)$
 2: emit $\langle j; (v, N_v^H) \rangle$
 3: Group the set of nodes in $N_v^H$ by $h(.)$
 4: for all $i \in [0, k-1]$ do
 5:     if $gp_i \neq \emptyset$ then
 6:         emit $\langle i; (v, N_v^H) \rangle$
 7:     end if
 8: end for

    Reduce:Input:[$\langle i; (v, N_v^H) \rangle$]
 9: initiate all the core nodes' $N_u^H$ in main memory
10: for all pivot message $\langle i; (v, N_v^H) \rangle$ do
11:     for all $u \in N_v^H$ and $h(u) = i$ do
12:         for all $w \in N_v^H \cap N_u^H$ do
13:             emit $\triangle_{vuw}$
14:         end for
15:     end for
16: end for
```

messages are received.

The corresponding modification in the reduce side is as follows. For a given reduce instance R_i, it first reads all the core message into main-memory (Line 7, Algorithm 3). Then, it iterates over all pivot message. Each pivot message is intersected with the cached core messages for identifying the triangles. As presented in the MR-Baseline algorithm (Algorithm 2), each pivot message (v, N_v^H) needs to be processed in reduce instance R_i only for nodes $u : u \in N_v^H \, where \, h(u) = i$. Interestingly, this information is encoded within the pivot message. Thus, each pivot message is processed for all its requested core nodes once received (Lines 9-11, Algorithm 3).

Analysis of Bermuda Techniques

Extending the analysis in Section 2.1.4, we demonstrate that Bermuda-EC achieves improvement over MR-Baseline w.r.t both space usage and execution efficiency. Furthermore, we discuss the effect of the number of reducers k on the algorithm performance.

Theorem 1 *For a given number of reducers k, we have:*

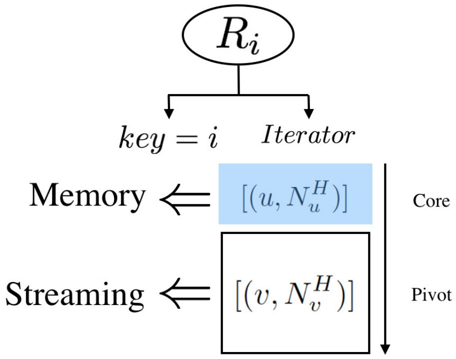


Figure 3.1: Bermuda-EC Execution.

- *The expected total size of the map output is $O(km)$.*
- *The expected size of core messages to any reduce instance is O(m/k).*

Proof 1 *As shown in Algorithm 3, the size of the map output generated by node v is at most $k * \hat{d}_v$. Thus, the total size of the map output T is as follows:*

$$T < \sum_{v \in V} k\hat{d}_v = k \sum_{v \in V} \hat{d}_v = km$$

For the second bound, observe that a random edge is present in a reduce instance R_i and represented as a core message with probability $1/k$. By following the Linearity of Expectation, *the expected number of the core messages to any reduce instance is $O(m * \frac{1}{k})$.*

Space Usage. Theorem 1 shows that when $k \ll \sqrt{m}$ (the usual case for massive graphs), then the total size of the map output generated by Bermuda-EC algorithm is significantly less than that generated by the MR-Baseline algorithm. In other words, Bermuda -EC is able to handle even larger graphs with limited compute clusters. **Execution Time.** A positive consequence of having a smaller intermediate result is that it requires less time for generating and shuffling/sorting the data. Moreover, the imbalance of the map outputs is also reduced significantly by limiting the replication factor of the pivot messages up to k. The next theorem shows the approximate variance of the number of the intermediate result from mappers. When $k < E(x)$, it implies smaller variance among the mappers than that of the MR-Baseline algorithm. Together, Bermuda -EC achieves better performance and scales to larger graphs compared to the MR-Baseline algorithm.

Theorem 2 *For a given graph $G(V, E)$, let a random variable x denotes the effective degree of any node in G and the variance of x is denotes as Var(x). Then the expectation of x ($E(x)$) equals the average degree and computed as $E(x) = \frac{m}{n}$. For typical graphs, $Var(x) \neq 0$ and $E(x) \neq 0$ always hold. Since each mapper starts with approximately the same input size (say receives c graph nodes), the variance of the map output's size under the Bermuda-EC Algorithm is $O(2ck^2Var(x))$, where k represents the number of reducers.*

Proof 2 *Assume the number of reducers is k. Given a graph node v, where its effective degree $\hat{d_v} = x$. Let random variable $y(x)$ be the number of distinct reducers processing the effective neighbors of v, and thus $y(x) \leq k$. Then, the size of the map output generated by a single node u would be xy, denoted as $g(x)$(Lines 3-4, Algorithm 3). Thus, the total size of the map output generated by c nodes in a single mapper $T(X) = \sum_{i=1}^{c} g(x_i)$. Since $x_1, x_2, ..x_c$ are independent and identically distributed random variables, then $Var(T(x)) = c * Var(g(x))$. The approximate variance of $g(x)$ is as follows*

$$\begin{aligned}
Var(xy) &= E(x^2y^2) - E(xy)^2 \\
&< E(x^2y^2) \\
&< k^2E(x^2) \\
&< k^2(E(x)^2 + Var(x)) \\
&< 2k^2Var(x)
\end{aligned}$$

As presented in [46], $E(x^2) \approx \frac{m^{\frac{3}{2}}}{n}$ and $E(x) = \frac{m}{n}$. Thus $\frac{E(x^2)}{E(x)^2} \approx \frac{n}{\sqrt{m}}$. In many real graphs where $n^2 > m$ it implies $\frac{n}{\sqrt{m}} > \sqrt{m} > 2$. It implies $E(x^2) > 2E(x)^2$, thus $Var(x) = E(x^2) - E(x)^2 > E(x)^2$.

We now study in more details the effect of parameter k (the number of reducers) on the space and time complexity for the Bermuda-EC algorithm.

Effect on Space Usage. The reducers number k trades off the memory used by a single reduce instance and the size of the intermediate data generated during the MapReduce job. The memory used by a single reducer should not exceed the available memory of a single machine, *i.e.*, $O(m/k)$ should be sub-linear to the size of main memory in a single machine. In addition, the total space used by the intermediate data must also remain bounded, *i.e.*, $O(km)$ should be no larger than the total storage. Given a cluster of machines, these two constraints define the bounds of k for a given input graph $G(V, E)$.

Effect on Execution Time. The reducers number k trades off the reduce computation time and the time for shuffling and sorting. As the parallelization degree k increases, it reduces the computational time in the reduce phase. At the same time, the size of the intermediate data, *i.e.*, $O(km)$ increases significantly as k increases (notice that m is very large), and thus the communication cost becomes a bottleneck in the job's execution. Moreover, the increasing variance among mappers $O(2ck^2Var(x))$ implies a more significant straggler problem which slows down the execution progress.

In general, Bermuda-EC algorithm favors the smaller setting of k for higher efficiency while subjects to memory bound that the expected size of core message $O(m/k)$ should not exceed the available memory of a single reduce instance.

Unfortunately, for processing web-scale graphs such as *ClueWeb* with more than 80 billion edges (and total size of approximately 700GBs)—which as we will show the state-of-art techniques cannot actually process—the number of reducers needed for Bermuda-EC for acceptable performance is in the order of 100s. Although, this number is very reasonable for most mid-size clusters, the intermediate results $O(km)$ will be huge, which leads to significant network congestion.

Disk-Based Bermuda-EC: A generalization to the proposed Bermuda-EC algorithm that guarantees no failure even under the case where the core messages cannot fit in a reducer's memory is the *Disk-Based Bermuda-EC* variation. The idea is straightforward and relies on the usage of the local disk of each reducer. The main idea is as follows: (1) Partition the core messages such that each partition fits into main memory, and (2) Buffer a group of pivot messages, and then iterate over the core messages one partition at a time, and for each partition, identify the triangles as in the standard Bermuda-EC algorithm. Obviously, such method trades off between disk I/O (pivot message scanning) and main-memory requirement. For a setting of reduce number k, the expected size of core messages in a single reduce instance is $O(m/k)$, thus the expected number of rounds is $O(\frac{m}{kM})$ where M represents the size of available main-memory for single reducer. The expected size of pivot message reaches $O(m)$. Therefore, the total disk I/O reaches $O(\frac{m^2}{kM})$. In the case of massive graph, it implies longer time.

3.3 Bermuda Vertex-Centric Node++

As discussed in Section 3.2, the Bermuda-EC algorithm assumes that the core messages can fit in the memory of a single reducer. However, it is not always guaranteed to be the

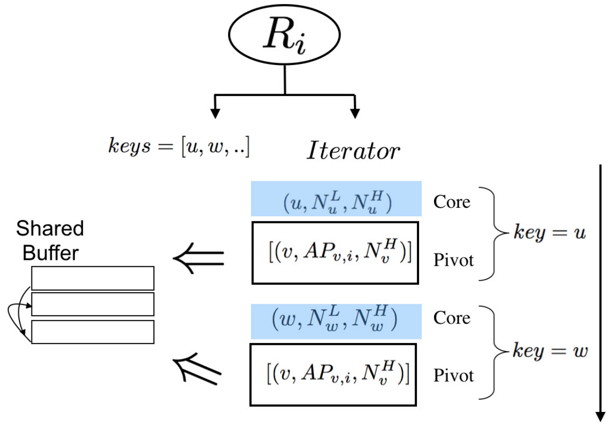


Figure 3.2: Bermuda-VC Execution.

case, especially in web-scale graphs.

One crucial observation is that the access pattern of the pivot messages can be learned and leveraged for better re-usability. In MapReduce, a single reduce instance processes many keys (graph nodes) in a specific sequential order. This order is defined based on the key comparator function. For example, let $h()$ be the key partitioning function and $l()$ be key comparator function within the MapReduce framework, then $h(u) = h(w) = i$ and $l(u, w) < 0$ implies that the reduce instance R_i is responsible for the computations over nodes $u,\ w$, and also the computations of node u precede that of node w. Given these known functions, the relative order among the keys in the same reduce instance becomes known, and the access pattern of the pivot message can be predicted. The knowledge of the access pattern of the pivot messages holds a great promise for proving better caching and better memory utilization.

Inspired by these facts, we propose the Bermuda-VC algorithm which supports random access over the pivot messages by caching them in main-memory while streaming in the core messages. More specifically, Bermuda-VC will reverse the assumption of Bermuda-EC, where we now try to make the pivot messages arrive first to reducers, get them cached and organized in memory, and then the core messages are received and processed against the pivot messages. Although the size of the pivot messages is usually larger than that of the core messages, their access pattern is more predictable which will enable better caching strategies as we will present in this section. The Bermuda-VC algorithm is presented in Algorithm 4.

The Bermuda-VC algorithm uses a shared buffer for caching the pivot messages. And

Algorithm 4 Bermuda-VC

```
    **Map**: Input:($\langle v; (N_v^L, N_v^H)\rangle$)
    Let $h(.)$ be a key partitioning function into [0,k-1]
    Let $l(.)$ be a key comparator function
 1: emit $\langle v; (v, N_v^L, N_v^H)\rangle$
 2: *Group* the set of nodes in $N_v^H$ by $h(.)$
 3: **for all** $i \in [0, k-1]$ **do**
 4:     **if** $gp_i \neq \emptyset$ **then**
 5:         $gp_i \Leftarrow sort(gp_i) basedonl(.)$
 6:         $u \Leftarrow gp_i.first$
 7:         $AP_{v,i} \Leftarrow accessPattern(gp_i)$
 8:         emit $\langle u; (v, AP_{v,i}, N_v^H)\rangle$
 9:     **end if**
10: **end for**

    **Reduce**:Input:[$\langle u; (v, AP_{v,i}, N_v^H)\rangle$]
11: initiate the core node $u$' $N_u^L, N_u^H$ in main memory
12: **for all** pivot message $\langle u; (v, AP_{v,i}, N_v^H)\rangle$ **do**
13:     **for all** $w \in N_v^H \bigcap N_u^H$ **do**
14:         emit $\triangle_{vuw}$
15:     **end for**
16:     Put $(v, AP_{v,j}, N_v^H)$ into shared buffer
17:     $N_u^L \leftarrow N_u^L - \{v\}$
18: **end for**
19: **for all** $r \in N_u^L$ **do**
20:     Fetch $(r, AP_{r,i}, N_r^H)$ from shared buffer
21:     **for all** $w \in N_r^H \bigcap N_u^H$ **do**
22:         emit $\triangle_{ruw}$
23:     **end for**
24: **end for**
```

then, for the reduce-side computations over a core node u, the reducer compares u's core message with all related pivot messages—some are associated with u's core message, while the rest should be residing in the shared buffer. Bermuda-VC algorithm applies the same scheme to avoid generating redundant pivot messages. It utilizes a universal key partitioning function to group effective neighbors N_v^H of each node v. In order to guarantee the availability of the pivot messages, a universal key comparator function is utilized to sort the destination nodes in each group (Line 5, Algorithm 4). As a result, destination nodes are sorted based on their processing order. The first node in group gp_i indicates the earliest request of a pivot message. Hence, each node v sends a pivot message to the first node of each non-empty group by emitting key value pairs where key equals the first node ID (Lines 6-8, Algorithm 4).

Combined with the sorting phase of the MapReduce framework, Bermuda-VC guarantees the availability of all needed pivot messages of any node u when u's core message is received by a reducer, i.e., the needed pivot messages are either associated with u itself or associated with another nodes processed before u.

The reducers' processing mechanism is similar to that of the MR-Baseline algorithm. Each node u reads its core message for initiating N_u^H and N_v^L (Line 9), and then it iterates over every pivot message associated with key u against its effective adjacency list N_v^H to enumerate the triangles (Lines 10-12). As discussed before, not all expected pivot messages are carried with key u. The rest of the related pivot messages reside in the shared buffer. Here, N_v^L is used for fetching the rest of these pivot messages (Line 14, Algorithm 4), and enumerating the triangles (Lines 15-18, Algorithm 4). Moreover, the new coming pivot messages associated with node u are pushed into the shared buffer for further access by other nodes (Line 13). Figure 3.2 illustrates the reduce-side processing flow of Bermuda-VC. In the following sections, we will discuss in more details the management of the pivot messages in the shared buffer.

3.3.1 Message Sharing Management

Note that, each reduce instance uses its shared buffer to organize sharing pivot messages in memory. Message sharing via shared buffer is effective because the reduce computation program access the same pivot message over and over. By keeping as much of pivot message as possible in the shared buffer, it avoids access the slower medium–disk. In general, there are two types of operations over the shared buffer inside a reduce instance, which are: *"Put"* for adding new incoming pivot messages into the shared buffer (Line 13), and *"Get"* for retrieving the needed pivot messages (Lines 15-18). For massive graphs, the main memory may not hold all the pivot messages. This problem is similar to the classical caching problem studied in [29, 44], where a *reuse-distance* factor is used to estimate the distances between consecutive references of a given cached element, and based on that effective replacement policies can be deployed. We adopt the same idea in Bermuda-VC.

Interestingly, in addition to the reuse distance, all access patterns of each pivot message can be easily estimated in our context. The access pattern AP of a pivot message is defined as the sequence of graph nodes (keys) that will reference this message. In

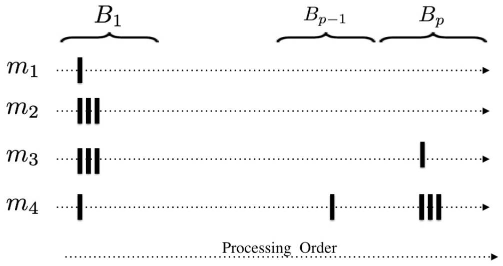


Figure 3.3: Various access Patterns for Pivot Messages.

particular, the access pattern of a pivot message from node v to reduce instance R_i can be computed based on the sorted effective nodes gp_i received by R_i. Several interesting metrics can be derived from this access pattern. For example, the first node in gp_i indicates the occurrence of the first reference, the size of gp_i equals the cumulative reference frequency. Such access pattern information is encoded within each pivot message (Lines 7-8, Algorithm 4). With the availability of this access pattern, effective message sharing strategies can be deployed under limited memory.

As an illustrative example, Figure 3.3 depicts different access patterns for four pivot messages $\{m_1,\ m_2,\ m_3,\ m_4\}$. The black bars indicate requests to the corresponding pivot message, while the gaps represent the re-use distances (which are idle periods for this message). Pivot messages may exhibit entirely different access patterns, e.g., pivot message m_1 is referenced only once, while others are utilized more than once, and some pivot messages are used in dense consecutive pattern in a short interval, *e.g.*, m_2 and m_3. Inspired by these observations, we propose two heuristic-based replacement policies, namely *usage-based tracking*, and *bucket-based tracking*. They trade off the tracking overhead with memory hits as will be described next.

•**Usage-Based Tracking** Given a pivot message originated from node v, the total use frequency is limited to$\sqrt{m}$, referring to the number of its effective neighbors, which is much smaller than the expected number of nodes processed in a single reducer, which is estimated to n/k. This implies that each pivot message may become useless (and can be discard) as a reducer progresses, and it is always desirable to detect the earliest time at which a pivot message can be discarded to maximize the memory's utilization.

The main idea of the usage-based tracking is to use a usage counter per pivot message in the shared buffer. And then, the tracking is performed as follows. Each *Put* operation sets the counter as the total use frequency. And, only the pivot messages whose usage counter is larger than zero are added to the shared buffer. Each *Get* operation decre-

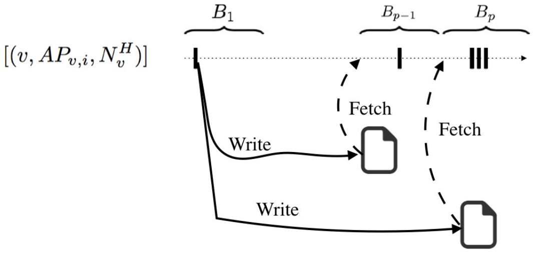


Figure 3.4: Read and Write Back-up files

ments the counter of the target pivot message by one. Once the counter reached zero, the corresponding pivot message is evicted from the shared buffer.

The usage-based scheme may fall short in optimizing sparse and scattered access patterns. For example, as shown in Figure 3.3, the reuse distance of message m_4 is large. Therefore, the usage-based tracking strategy has to keep m_4 in the shared buffer although it will not be referenced for a long time. What's worse, such scattered access is common in massive graphs. Therefore, pivot messages may unnecessarily overwhelm the available memory of each single reduce instance.

Bucket-Based Tracking We introduce the *Bucket-based* tracking strategy to optimize message sharing over scattered access patterns. The main idea is to manage the access patterns of each pivot message at a smaller granularity, called a bucket. The processing sequence of keys/nodesis sliced into buckets as illustrated in Figure 3.3. In this work, we use the range partitioning method for balancing workload among buckets. Correspondingly, the usage counter of one pivot message is defined per bucket, i.e., each message will have an array of usage counters of a length equals to the number of its buckets. For example, the usage count of m_4 in the first bucket is 1 while in the second bucket is 0. Therefore, for a pivot message that will remain idle (with no reference) for a long time, its counter array will have a long sequence of adjacent zeros. Such access pattern information can be computed in the map function, encoded in access pattern (Line 7 in Algorithm 4), and passed to the reduce side.

The corresponding modification of the *Put* operation is as follows. Each new pivot message will be pushed into the shared buffer (in memory) and backed up by local files (in disk) based on its access pattern. Figure 3.4 illustrates this procedure. For the arrival of a pivot message with the access pattern $[1, 0, .., 1, 3]$, the reduce instance actively adds this message into back-up files for buckets B_{p-1} (next-to-last bucket) and B_p (last bucket).

And then, at the end of each bucket processing and before the start of processing the next bucket, all pivot messages in the shared buffer are discarded, and a new set of pivot messages is fetched from the corresponding back-up file into memory(See Figure 3.4).

The Bucket-based tracking strategy provides better memory utilization since it prevents the long retention of unnecessary pivot messages. In addition, usage-based tracking can be applied to each bucket to combine both benefits, which is referred to as the *bucket-usage* tracking strategy.

3.3.2 Disscussion

In this section, we show the benefits of the Bermuda-VC algorithm over the Bermuda-EC algorithm. Furthermore, we discuss the effect of parameter p, which is the the number of buckets, on the performance.

Under the same settings of the number of reducers k, the Bermuda-VC algorithm generates more intermediate message and takes longer execution time. Firstly, the Bermuda-VC algorithm generates the same number of pivot messages while generating more core messages (***i.e.***, additional N_v^L for reference in the reduce side). Thus, the total size of the extra N_v^L core message is $\sum_{v \in V} N_v^L = m$. Such noticeable size of extra core messages requires additional time for generating and shuffling. Moreover, an additional computational overhead (Lines 13-14) is required for the message sharing management.

However, because of the proposed sharing strategies, the Bermuda-VC algorithm can work under smaller settings for k—which are settings under which the Bermuda-EC algorithm will probably fail. In this case, the benefits brought by having a smaller k will exceed the corresponding cost. In such cases, Bermuda-VC algorithm will outperform Bermuda-EC algorithm.

Moreover, compared to the disk-based Bermuda-EC algorithm, the Bermuda-VC algorithm has a relatively smaller disk I/O cost because the predictability of the access pattern of the pivot messages, which enable purging them early, while that is not applicable to the core messages. Notice that, for any given reduce instance, the expected usage count of pivot message from u is d_u^H/k. Thus, the expected usage count for any pivot message is $E(d_u^H)/k$, equals m/nk. Therefore, the total disk I/O with pivot messages is at most m^2/nk, smaller than disk I/O cost of Bermuda-EC algorithm m^2/Mk, where M stands for the size of the available memory in a single machine.

Effect of the number of buckets p: At a high level, p trades off the space used by the shared buffer with the I/O cost for writing and reading the back-up files. Bermuda-VC algorithm favors smaller settings of p in the capacity of main-memory. As p decreases, the expected number of reading and writing decreases, however the total size of the pivot messages in the shared buffer may exceed the capacity of the main-memory. For a setting of p, the expected size of the pivot messages for any bucket is $O(m/kp)$. Therefore, a visible solution for $O(m/kp) \leq M$ is $\geq O(m/kM)$. Here, p is set as $O(m/kM)$ where m is the size of the input graph, and M is the size of the available memory in a single machine.

3.4 Contribution List

Different from existing approaches, Bermuda effectively reduces the size of the intermediate data via redundancy elimination and sharing of messages whenever possible. Our constributions can be summarized as follows:

- ***Independence of Graph Partitioning:*** Bermuda does not require any special partitioning of the graph, which suites current applications in which graph structures are very complex and dynamically changing.
- ***Awareness of Processing Order and Locality in the reduce phase:*** Bermuda's efficiency and optimizations are driven by minimizing the communication overhead and the number of message passing over the network. Bermuda achieves these goals by dynamically keeping track of where and when vertices will be processed in the reduce phase, and then maximizing the re-usability of information among the vertices that will be processed together. We propose several reduce-side caching strategies for enabling such re-usability and sharing of information.
- ***Scalability to Large Graphs even with Limited Compute Clusters:*** As our experiments show, Bermuda's optimizations—especially the reduction in communication overheads—enable the scalability to very large graphs, while the state-of-art technique fail to finish the job given the same resources.

Chapter 4

Degree Anonymization over Uncertain Graphs

In this chapter, I will briefly review the degree anonymization tehcniques I developed for uncertain graphs. These techniques have been submitted as a conference paper [54]. In this work, we study the novel problem of anonymization in the context of uncertain graphs. We extend the existing framework to work over uncertain graph by integrating uncertainty into anonymization process. In particular, we introduce a new *reliability-based* utility metric suitable for uncertain graphs, in contrast to the existing metrics which are all geared towards deterministic graphs. We present an efficient approach which achieves the desired level of anonymity at a slight cost of reliability by perturbing edge uncertainties judiciously. To this purpose, we develop two uncertainty-aware heuristics based on the uncertain graph theory, which is *reliability-oriented* edge selection and *anonymity-oriented* edge perturbing. We show that the incorporation of uncertainty is necessary and beneficial. We experimentally evaluate the proposed approach using different real-world datasets and study the behavior of the algorithms under the different heuristics. The results demonstrate the effectiveness of our approach.

4.1 Naive Approach: Anonymization via Representative Instance

A naive approach of anonymizing an uncertain graph is to first somehow transform it to a deterministic graph, then perform anonymization processing over the deterministic

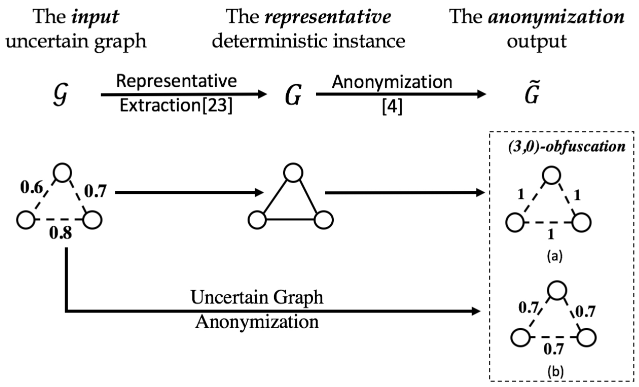


Figure 4.1: Illustration of anonymizing an uncertain graph though its representative deterministic instance and its drawback.

one. Fortunately, an increasing research effort was dedicated to the topic of exacting representative deterministic graphs from an uncertain graph [42]. Parchas *et al.* [42] ever introduced algorithms for extracting deterministic representative graph which captures key properties of the input uncertain graph. Now, it becomes realizable to anonymize an uncertain graph in two steps as shown in Figure 4.1. We first extract one deterministic representative instance G from the input uncertain graph $\mathcal{G}$. Then, we anonymize the extracted deterministic graph G, and output this result as the anonymized result of the original uncertain graph $\mathcal{G}$ (referred as Rep-An).

The Rep-An approach is attracting since it does not require any new anonymization techniques specific designed for uncertain graphs. When the extracted representative deterministic graph G is close enough to the input uncertain graph $\mathcal{G}$ in terms of graph properties, its anonymized result is expected to be a good anonymization of the input uncertain one. However, there is a non-negotiable difference between the input uncertain graph $\mathcal{G}$ and its deterministic representative instance G, as exemplified in Figure 4.1. The anonymized result of G which is structurally similar to itself instead of the input uncertain graph, consequently, may be far different from the optimal solution, as exemplified in Figure 4.1. Therefore, we believe that for many applications, the Rep-An approach, introducing a high level of noise in such fashion, do reduce the overall graph utility. In experiment section, we will further illustrate this phenomenon over real-world datasets.

Algorithm 5 Anonymization over uncertain graphs

```
Input: Uncertain graph $\mathcal{G}$, adversary knowledge $ak$, obfuscation level $k$, tolerance level $\epsilon$, size multiplier $c$ and white noise level $q$
Output: The anonymized result $\tilde{\mathcal{G}}_f$
Initiation of an lower bound $\sigma_l$ and an upper bound $\sigma_u$
while (not terminated) do
    Search of $(k, \epsilon)$-obf using standard deviation $\sigma$
    $\langle \tilde{\epsilon}, \tilde{\mathcal{G}} \rangle \leftarrow$ GenerateObfuscation($\mathcal{G}$, $\sigma_u$, $ak$, $k$, $\epsilon$, $c$, $q$)
    Update the anonymized result if necessary
    Update the lower bound or the upper bound
    Update the size multiplier $c$ if necessary
end while
return the anonymized result $\tilde{\mathcal{G}}_f$
```

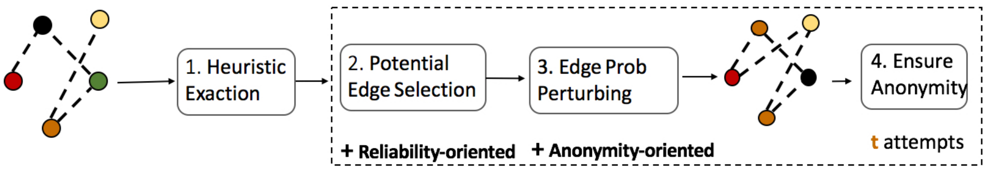


Figure 4.2: The pipeline of function GenerateObfuscation

4.2 Reliability-Preserving anonymization on uncertain graphs

In this section, we first give a brief review of the state-of-art framework which anonymizes deterministic graphs by injecting uncertainty to certain edges. Then, we discuss how to extend this framework to work over uncertain graphs by explicitly incorporating edge uncertainty.

4.2.1 Anonymization Procedure

Boldi *et al.* proposed a seminal approach of anonymizing *deterministic graphs* which injects uncertainty in the existence of the edges of the graph and publishes the resulting *uncertain graph*. Their method injects uncertainty to edges in a deterministic graph as follows: for each existing sampled edge e, it is assigned a probability $1 - r_e$, and for each non-existing sampled edge e, it is assigned a probability $r_e \leftarrow R_{\sigma_e}$. By this way, it converts the input graph into an uncertain one. In particular, the perturbation variable r_e

is generated by a truncated $[0, 1]$ normal distribution, $r_e \leftarrow R_{\sigma_e}$;

$$R_\sigma(r) := \begin{cases} \frac{\Phi_{0,\sigma}(r)}{\int_0^1 \Phi_{0,\sigma}(x)dx} & r \in [0,1] \\ 0 & \text{otherwise} \end{cases}$$

where $\Phi_{0,\sigma}$ is the density function of a Gaussian distribution. As the standard deviation σ of the normal distribution decreases, a greater mass of R_σ will concentrate near $r = 0$ and the amount of injected uncertainty will be smaller. Namely, smaller values of σ contribute towards better maintaining the characteristics of the original graph. Targeting at high utility, they formulated the graph anonymization problem as the minimization of σ and computed the minimal amount of uncertainty via a binary search on the value of the uncertainty parameter σ. The overall anonymization procedure is illustrated in Algorithm 5.

The search flow of Algorithm 5 is determined by the function `GenerateObfuscation`, whose pipeline is shown as Figure 4.2. The function `GenerateObfuscation` aims to find a (k, ϵ)-obfuscation of the original *graph* using a given standard deviation parameter σ. A key feature of this function is the utilization of heuristics for judicious edge selection and edge prob perturbing. The function first computes heuristics values based on node properties in the input graph. After that, the search of (k, ϵ)-*obf* instance starts in a randomized way: t attempts are performed. Each attempt performs the following steps:

- Selecting a subset of edges E_c
- Perturbing existence probabilities of selected edges
- Checking the resulting uncertain graph whether satisfies the anonymity requirement

If the algorithm finds a (k, ϵ)-obfuscated graph in one of its t trials, it returns the obfuscated graph with minimal ϵ. Otherwise, the algorithm indicates the failure by returning $\tilde{\epsilon} = 1$.

In this work, we extend this broad framework by incorporating its key function `GenerateObfuscation` with uncertainty. The specific method proposed in [11], has two drawbacks for anonymizing uncertain graphs. First, their method does not consider the structural relevance of edges in the critical edge selection step, which leads to unnecessary structural distortion. Second, its interior edge perturbing mechanism assumes the existence of edges is known with certainty, thus fails to handle uncertain graphs where the existence of edges is probabilistic. We will present corresponding solutions (reliability-oriented edge selection and anonymity-oriented edge perturbing).

4.2.2 Reliability-oriented Edge Selection

The most important step is the selection of potential edges, which impacts further anonymization effort significantly. Finding the optimal set of edges E_c that balances privacy and utility is a typical combinational optimization problem. The problem is computationally expensive since there are exponential many combinations to be considered.

To alleviate the combinational intractability, kind of heuristics have been utilized in the context of deterministic graphs. These heuristics can be classified into two main categories (1) Anonymity-oriented ones that suggest implementing larger perturbation to less anonymized nodes [56] (2) Utility-oriented ones that suggest implementing smaller perturbation to more influential edges/nodes [14, 57].

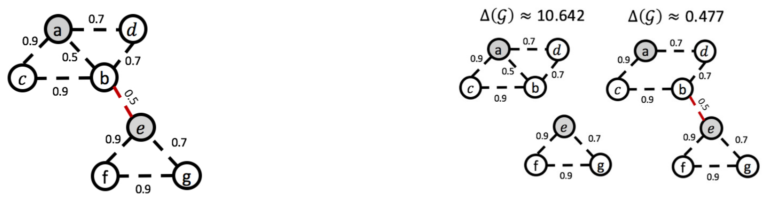
In this work, we present a sampling-based approach which combines both heuristics together. First, we adopt the idea of *uniqueness score* for quantifying the anonymity level of nodes. Second, we introduce a novel edge relevance metric, *reliability relevance* (RR), for quantifying the impact of edge modification to the overall uncertain graph. In contrast to the existing metrics such as edge betweenness [14], which are all defined in deterministic graphs, reliability relevance geared towards uncertain graphs. It allows us to select a subset of edges subjected to perturbation with less impact on reliability (reliability-oriented edge selection).

Uniqueness Score

Uniqueness score is proposed to measure how typical the node is among all the nodes in terms of its property value [11]. More formally, the uniqueness score is defined as follows.

Definition 5 ***Uniqueness Score [11]*** *Let $P : V \to \Omega_P$ be a property on the set of nodes V of the graph $\mathcal{G}$, let d be a distance function on Ω_P, and let $\theta > 0$ be a parameter. Then the $\theta-commonness$ of the property values $\omega \in \Omega_P$ is $C_\theta(\omega) := \sum_{u \in V} \Phi_{0,\theta}(d(\omega, P(v)))$, while the θ-uniqueness of $\omega \in \Omega_P$ is $U_\theta := \frac{1}{C_\theta(\omega)}$.*

Note that, the commonness of the property value ω captures how typical the value ω is among all the nodes in the graph via a weighted average function, where the weight decays exponentially as the distance d. As above defined, uniqueness scores depend on


a
0.7
d
0.9
0.5
0.7
c
b
0.9
0.5
e
0.9
0.7
f
g
0.9
$\Delta(\mathcal{G}) \approx 10.642$
$\Delta(\mathcal{G}) \approx 0.477$

(a) An uncertain graph

(b) Resulting uncertain graphs

Figure 4.3: Illustration of varying structural distortions by removing different uncertain edges (b, e) and (a, b) over an uncertain graph.

the parameter θ, which determines the decay rate of average weights. In this work, we set $\theta = \sigma_{\mathcal{G}}$, where $\sigma_{\mathcal{G}}$ represents how frequently the property value spread in the uncertain graph $\mathcal{G}$. A comprehensive discussion can be found in the literature [11].

Reliability Relevance

As exemplified in Figure 4.3, each edge modification will have *varying* impact to the graph structure in the context of uncertain graphs. From the perspective of *uniqueness*, nodes a and e are identical since adjacent edges have identical uncertainties. Accordingly, anonymity-oriented heuristics would select and perturb edges (a, b) and (b, e) without bias. As shown in Figure 4.3, the deletion of the uncertain edge (b, e), the only link connecting two *reliable* clusters, clearly incurs much larger structure distortion than the deletion of the edge (a, b). This observation calls for an effective measure of edge influence in the context of uncertain graphs.

Inspired by the significance of *reliability*, we provide a compact analytic form of reliability deviation caused by individual uncertain edge modification in a fine-grained way, namely changing its edge probability in any granularity. Following the path, we introduce edge reliability relevance for edges' influence estimation, which can be quantified by the sum of reliability discrepancy of all the node pairs. Besides, we provide an algorithm for accessing edge *reliability relevance* (RR) efficiently.

Reliability Relevance Definition Lets start with analyzing the impact of single uncertain edge alteration to the connectivity of a specific node pair (Two-terminal Reliability).

Definition 6 ***Two-terminal Reliability Relevance*** *Given an uncertain graph $\mathcal{G}$, and two nodes u, v, we consider the probability that u is reachable to v,$R_{u,v}(\mathcal{G})$, as defined in Definition 2, as a multivariate function involving all the edge probabilities. Thus, the partial* *derivative of $R_{u,v}(\mathcal{G})$ with respect to an individual variable $p(e)$, existence probability of an uncertain edge, denotes as $RR_{u,v}(e)$. It represents the sensitivity to the change of reliability $R_{u,v}$ which is determined by $p(e)$ with the others held constant. Its definitions is as follows:*

$$RR_{u,v}(e) = \frac{\partial R_{u,v}(\mathcal{G})}{\partial p(e)}$$

Lemma 1 ***Factorization Lemma*** *Given an uncertain graph $\mathcal{G}$, the reliability of the node pair (u, v) $R_{u,v}(\mathcal{G})$ can factorized via one uncertain edge e as follows:*

$$R_{u,v}(\mathcal{G}) = p(e)R_{u,v}(\mathcal{G}_e) + (1 - p(e))R_{u,v}(\mathcal{G}_{\bar{e}})$$

where uncertain graph $\mathcal{G}_e$ is identical with $\mathcal{G}$ except its edge e is an ***certain*** *edge. Similarly, the uncertain graph $\mathcal{G}_{\bar{e}}$ is identical with $\mathcal{G}$ except its edge e is an* ***certain non*** *edge.*

According to the factorization lemma, the partial derivative $RR_{u,v}(e)$ can be rewritten as

$$RR_{u,v}(e) = R_{u,v}(\mathcal{G}_e) - R_{u,v}(\mathcal{G}_{\bar{e}})$$

The equation indicates the sensitivity of reliability is constant and it does not depend on the probability value of the uncertain edge e. Another important point to remember is that because all the connected pairs remain connected after the addition of an edge, $R_{u,v}(\mathcal{G}_e) - R_{u,v}(\mathcal{G}_{\bar{e}}) \geq 0$.

For a given uncertain edge e, the derivatives $RR_{u,v}(e)$ can be arranged in a $|V| \times |V|$ matrix, where each element corresponds to one node pair (u, v) and it gives the partial derivative for the corresponding reliability $R_{u,v}$. As ever discussed, all the element are equal to or greater than zero. In this work, we define $RR(e)$ to be reliability relevance of an edge e which can be quantified by the sum of all the $RR_{u,v}(e)$, as

$$\begin{aligned} RR(e) &= \sum_{u,v} |RR_{u,v}(e)| \\ &= \sum_{u,v} |R_{u,v}(\mathcal{G}_e) - R_{u,v}(\mathcal{G}_{\bar{e}})| \\ &= \sum_{u,v} R_{u,v}(\mathcal{G}_e) - \sum_{u,v} R_{u,v}(\mathcal{G}_{\bar{e}}) \end{aligned}$$

Note that, $RR(e)$ equals the difference of the expected number of connected pairs between uncertain graphs $\mathcal{G}_e$ and $\mathcal{G}_{\bar{e}}$ by explicit incorporation of edge uncertainty. In the context of edge relevance, reliability relevance can be seen as generalization of cut-edges, which quantifies the impact of partial edge deletion or addition on the connectivity in the

Algorithm 6 Reliability Relevance Evaluation

```
    Input: $\mathcal{G} = (V, E, p)$, $K$ is the number of sampled graphs; usually $K = 1000$
    Output: $RR$ Reliability relevance of edges in $\mathcal{G}$
$CC_e \leftarrow \mathbf{0}$, $CC_{\bar{e}} \leftarrow \mathbf{0}$
for i=1 to K do
    $G \leftarrow$ A deterministic sampled instance
    $Ind(G)$ is edge existence of sampled graph $\mathcal{G}$
    $cc(G) \leftarrow$ the number of connected pairs of $G$
    $CC_e += Ind(G) \cdot cc(G)$,$CC_{\bar{e}} += (\mathbf{1} - Ind(G)) \cdot cc(G)$
end for
$RR = CC_e/p - CC_{\bar{e}}/\mathbf{1} - p$
```

uncertain graph. The higher reliability relevance score of an edge, the bigger impact of edge perturbation over the overall graph. On this basis, we define the reliability centrality of node v,$RC(v)$ to be the overall influence of graph reliability which can be quantified by the weighted sum of reliability relevance of all adjacent edges.

$$RC(u) = \sum_{e} p(e) RR(e)$$

Reliability Relevance Evaluation Note that, the evaluation of $RR(e)$ need computing all the $R_{u,v}$ over two uncertain graphs $\mathcal{G}_e$ and $\mathcal{G}_{\bar{e}}$. The two-terminal reliability detection problem is a #P-complete problem [5]. Thus, the exact computation is not practical for large uncertain graphs. A basic approach is to get reliability approximation by sampling: for each uncertain edge e 1) we first sample K possible graphs $G_1, \ldots G_K$ of $\mathcal{G}_{\bar{e}}$ according to edge probability p, and 2) we then compute the number difference of connected pairs in each sample graph after edge e is added. The basic sampling method can be rather computationally expensive. The total running time is $\Theta(|E| \cdot K \cdot \alpha(|V|)|E|)$ assuming incremental algorithm is used for computing connected components. It is impractical for large uncertain graphs with huge size of edges.

Here, we introduce a much faster approach for computing $RR(e)$ as shown in Algorithm 6. The basic idea is to partition all sampled graphs into groups according to existing edges so that the evaluation of the number of connected pairs can be reused. The complexity of reliability relevance evaluation of all edges is $\Theta(K \cdot \alpha(|V|)|E|)$.

Reliability-oriented Edge Selection Procedure

Algorithm 7 GenerateObfuscation

```
Input: Uncertain graph $\mathcal{G} = (V, E, p)$, $ak, k, \epsilon, c, q$,
and standard deviation $\sigma$
Output: A pair $\langle \tilde{\epsilon}, \tilde{\mathcal{G}} \rangle$
compute the uniqueness $U(v)$ for all the nodes
compute the centrality $RC(v)$ for all the nodes
$Q(v) \leftarrow U(v) \cdot RC(v)$
$H \leftarrow$ the set of $\lceil \frac{\epsilon}{2} |V| \rceil$ with largest $Q(v)$ {Excluding}
Normalized $RC(v)$; $Q(v) \leftarrow U(v) \cdot 1 - RC(v)$;
$\tilde{\epsilon} \leftarrow 1$
for $t$ times do
    $E_C \leftarrow E$ {Reliability-oriented Edge Selection}
    repeat
        randomly pick a vertex $u \in V \setminus H$ according to $Q$
        randomly pick a vertex $v \in V \setminus H$ according to $Q$
        draw $w$ uniformly at random from $[0, 1]$
        if $(u, v) \in E$ then
            if $w > p(e)$ then
                $E_C \leftarrow E_c \setminus \{(u, v)\}$
            end if
        else
            $E_c \leftarrow E_c \cup \{(u, v)\}$
        end if
    until $E_C = c|E|$
    for all $e \in E_C$ do
        compute $\sigma(e)$ {Edge Probability Perturbation}
        draw $w$ uniformly at random from $[0, 1]$
        if $w < q$ then
            $r_e \leftarrow U(0, 1)$
        else
            $r_e \leftarrow R_{\sigma(e)}$
        end if
        $\tilde{p}(e) \leftarrow p(e) + 2(0.5 - p(e)) \cdot r_e$
    end for
    $\hat{\epsilon} \leftarrow checkAnonymity(\tilde{\mathcal{G}})$ {Ensure Anonymity}
    Update result $\langle \tilde{\epsilon}, \tilde{\mathcal{G}} \rangle$ if $\hat{\epsilon} < \tilde{\epsilon}$
end for
return $\langle \tilde{\epsilon}, \tilde{\mathcal{G}} \rangle$
```

Now, we are ready to introduce our function `GenerateObfuscation` aims at finding a (k, ϵ)-obf instance for an input *uncertain graph* $\mathcal{G}$ using a given standard deviation parameter σ as shown in Algorithm 7. It resembles the method proposed in [11] with explicit incorporation of edge uncertainty in edge selecting and perturbing step. Another important contribution of our work is the utilization of reliability relevance (RR) for capturing structural relevance of edges in the uncertain graph.

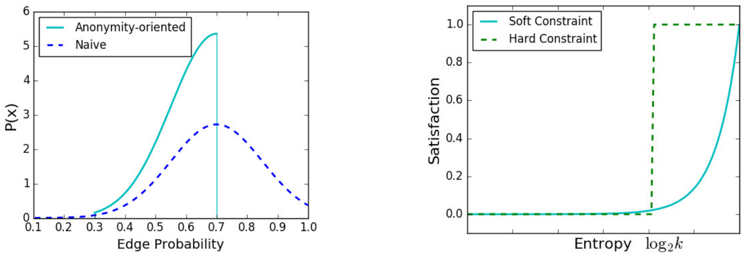
First, it computes the uniqueness level and reliability centrality for each node $v \in \mathcal{G}$. Intuitively, the more unique a node is, the harder it is to be obfuscated; the more *important* an edge is, the bigger utility loss it incurs. Such heuristic information is important for our privacy-preserving and utility-preserving purpose. In order to use the "uncertainty budget" in the most efficient way, the algorithm performs the following steps as shown in Algorithm 7.

(Line 4: Excluding) Since it is allowed not to obfuscate $\epsilon|V|$ of the nodes, the algorithm selects the set H of $\frac{\epsilon}{2}|V|$ nodes with largest uniqueness scores or reliability centrality scores, and exclude them from the subsequent obfuscation efforts. In later steps, the algorithm will inject uncertainty only to edges that are not adjacent to any of vertices in H. The key feature of our method is that it strictly preserve the 1-neighborhoods of influential nodes.

(Line 8-18: Potential Edge Selection) The set of nodes not in H will need to be anonymized. To anonymize more unique vertices, higher uncertainty is necessary. Thus, edges needed to be sampled with higher probability if they are adjacent to unique nodes. In order to better preserve graph structure, edges needed to be sampled with smaller probability if they are adjacent to more influential nodes. In order to handle this sample process, our algorithm assigns a probability $Q(v)$ to every $v \in V$, which depends on the uniqueness level $U(v)$ and reliability centrality $RC(v)$ of v.

Each attempt begins by selecting a subset of edges E_c, which will be subjected to edge probability perturbation. Then set E_c, whose target size $\|E_c\| = c\,\|E\|$, is initialized to be E. Then, the algorithm randomly selects two distinct vertices u and v, according to probability distribution Q. The pair of vertices (u, v) is removed from E_c with the probability $p(e)$ if it is an edge in the original graph, or added to E_c otherwise. The process is repeated until E_c reaches the require size. In typical uncertain graphs, the number of non-edges is significantly larger than the number of uncertain edges, the loop ends very quickly, for small values of c. And, the resulting set E_c includes most of edges in E.

(Line 20) Next, we redistribute the perturbation budget among all selected edges $e \in E_c$ in proportion to their intermediate representations $Q(v)$. Specially, we define for each



(a) Anonymity-oriented Edge Perturbing (b) Relaxing k-obfuscation constraint

$e = (u, v) \in E_c$, its uncertainty level,

$$Q(e) := \frac{Q(u) + Q(v)}{2}$$

and then set its edge perturbation parameter

$$\sigma(e) = \sigma|E_c| \cdot \frac{Q(e)}{\sum_{e \in E_c} Q(e)}$$

so that the average of $\sigma(e)$ over all $e \in E_C$ equals σ.

4.2.3 Anonymity-oriented Edge Perturbing

As ever discussed, the existing uncertainty injecting scheme is designed for deterministic graphs and can not be used to handle uncertain graphs directly. Given the uncertainty level $\sigma(e)$, a natural strategy is to consider the partial edge addition and deletion in a random way as shown in Figure 4.4(a). Through the analysis of the impact of a single edge probability alteration, we give a anonymity-oriented perturbing heuristics which is able to constraint the potential range of edge prob perturbing (referred to C). For each selected edge $e \in E_c$ and the assigned uncertainty level $\sigma(e)$, we alter its existence probability as

$$\tilde{p}(e) := p(e) + (1 - 2p(e)) \cdot r_e$$

Where the random perturbation r_e is generated according to $\sigma(e)$ as Eq. 4.2.1. Namely, for an edge with the probability p, we only consider potential edge probability $\tilde{p}$ in the limited range that more likely contributes to higher graph anonymity. Clearly, previous scheme defined in deterministic graph becomes a special case of our approach. We proceed to elaborate the rationality and the benefit of this anonymity-oriented edge perturbing scheme by treating it as a constraint satisfaction problem.

Basics

Let us consider k-obfuscate a given node v as a single constraint c_v for the target anonymization graph. Then, according to the definition 7, the satisfaction of the constraint c_v is defined as

$$\mathsf{c}_v := \begin{cases} 1 & H(Y_{P(v)}) \geq \log_2 k \\ 0 & otherwise \end{cases}$$

Namely, given the considered degree-based re-identification scenario, we lower bound the entropy of degree distribution over the anonymized graph by $log_2 k$. Accordingly, whether the anonymized graph k-obfuscates all the nodes can be expressed as degree of joint satisfaction.

$$\begin{aligned} \mathcal{C}(\mathcal{G}) &= \prod_{v \in V} \mathsf{c}_v \\ &= \prod_{\omega} \underbrace{\mathsf{c} \dots \mathsf{c}}_{s(\omega)} \end{aligned}$$

The uncertain graph is said to k-obfuscate all the nodes if and only if the $C(\mathcal{G})$ equals 1.

Re-visiting graph anonymity

As shown in Figure 4.4(b), the original satisfaction function is not everywhere differentiable.To simply the anonymity analysis, we approximate the individual constraint c_v to a fuzzy relation in which the satisfaction degree of a constraint is defined as a continuous and differentiable function, going from fully violated to fully satisfied. A natural candidate for soft satisfaction function is,

$$C_v = e^{H(Y_{P(v)}) - \log_2 |V|}$$

According to this approximation, the satisfaction function of k-obfuscate all the nodes can be rewritten as

$$\mathcal{C}(\mathcal{G}) \propto \prod_{\omega} \underbrace{e^{H(Y(\omega)} \dots e^{H(Y(\omega))}}_{s(\omega)}$$

While the original satisfaction is a binary value, the approximated satisfaction value lies in the continuous range $[0, 1]$. Note that, the higher satisfaction score indicates the higher level of anonymity achieved. Taking logarithm for both side, we get the concise formula as

$$\log \mathcal{C}(\mathcal{G}) = \sum_{\omega} s(\omega) \cdot H(Y(\omega)$$

Regarding the property P, it equals the weighted sum of entropy over all possible values ω, where $s(\omega)$ is the expected number of vertices with property value ω over all possible worlds. Targeting at high anonymity, we wish to increase the weighted sum of entropy.

Greedy search of graph anonymity

The remaining issue is to connect the single edge probability alteration with the objective. With respects to node degree, a graph can be represented as one matrix as shown in figure 4.4(b). The weighted sum of entropy is related with graph coding. Here, we utilize entropy encoding, especially Huffman coding. From the coding perspective, we have two different angles to perform graph coding: row or column of degree matrix. The encoded length should be equal to each other, as shown in the following equation.

$$\sum_{v} H(v) + n \log n = \sum_{\omega} s(\omega) \cdot H(Y(\omega)) + H(s(\omega))$$

It indicates that higher global anonymity of a uncertain graph can be achieved by increasing the entropy of individual nodes when the global distribution $H(\omega)$ keeps constant.

Note that, the probability change over a specific edge only affects degree distributions of its connected nodes. To further simply the problem, we assume that the impact of edge perturbing is independent to each other. Recall that, we suggest implementing perturbation to less anonymized nodes. In the case of degree obfuscation, they are nodes with high degree. For a node v with high degree, the probability distribution of its degree d_v may be approximated as the normal distribution as implied by the Central Limit Theorem. Hence, its induced entropy $H(v)$ can be approximated as $\frac{1}{2}\ln(2\pi e \sigma^2)$.

Targeting at maximizing the graph anonymity, we take steps proportion to the positive gradient with respect to the choice of individual edge probabilities.

$$\frac{\partial \sum_{\omega} s(\omega) H(Y(\omega))}{\partial p(e)} \propto (1 - 2p(e))$$

Therefore,

$$\tilde{p}(e) := p(e) + (1 - 2p(e)) \cdot r_e$$

Namely, our approach simulates one iteration of batch gradient ascent method for finding the local maximum of weighted entropy sum or the anonymity level.

4.3 Contribution List

To the best of our knowledge, we are the first to deal with uncertain graph anonymization aiming at less reliablity change. Our contrinutions inlcudes:

- We identify and formalize the new and important problem of privacy preserving publication of uncertain graphs. We design a new framework, namely *AUG*, for efficient anonymization of uncertain graphs under the (k, ϵ)-*obf* anonymization principle. *AUG* integrates the edge uncertainties throughout the anonymization process to ensure the desired privacy level.
- We show case that a naive approach that combines existing techniques from literature for uncertain graph anonymization do not work in practice due to significant utility loss.
- We introduce a new reliability-based utility metric for assessing the utility loss during the anonymization process. We propose uncertainty-aware heuristics for injecting noise and perturbing the graph structure to anonymize the uncertain graph while minimizing utility loss.
- We perform an extensive experimental evaluation using four real-world uncertain graph data sets from different domains. We evaluate *AUG* using different groups of graph metrics. The results demonstrate a significant advantage of the proposed framework over the naive conventional methods, which do not directly consider edge uncertainty.

Chapter 5

Probabiltic Degree Anonymization over Uncertain Graphs

In this section, we formalize the threat of re-identification combined with the degree probability distributions in the context of uncertain graphs. We study the node degree information incoporates edge uncertainty and show its power to re-identify individual in a uncertain graph. Protecting against the threat of re-identification presents novel challenges for uncertain graph data. Generalization, random perturbation and uncertain graphs methods have been developed for determinitic graph anonymization. It is not trivial to shift these strategies for obfuscating degree probability distributions of nodes in the uncertain graph.

5.1 Probabilitic Degree-based De-anonymization

In this section, we describe the threat of node re-identification in uncertain graph data, and we explain the content of external information and the use of external information in identifying anonymized individual. We develop the intuition behind achieving anonymity in an uncertain graph through structural similarity to others.

Here, we remind the reader its difference compared to the deterministic case. The incorporation of uncertainty in the graph data significantly affects the distribution of vertex property. Existing graph perturbation techniques are not equipped to operate over uncertain graphs – they will tend to ignore and destroy important structural properties. Likewise, uncertain graph structure and adversary knowledge with an incorporation of uncertainty to threaten privacy in new ways.

To gain more intuition on this attack model and the difficulties in defining a mean-

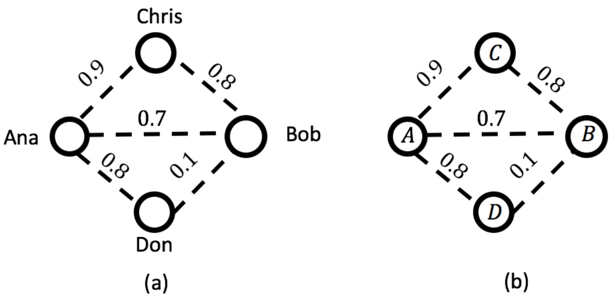


Figure 5.1: (a) The original uncertain graph (b) The naive anonymized version of uncertain graph with 5 edges and $2^5 = 32$ possible world

ingful probabilistic matching, we present a simple example. Consider the original uncertain graph and the anonymized version of the given graph shown in Figure 5.1. We assume the adversary is equipped with the external structural information about some victim nodes. In particular, the property is node degree. Similarly, the degree of any nodes in the anonymized uncertain graph can be expressed as uncertain data. In the case of uncertain graph de-anonymization, the uncertainty belongs to the representation of source objects which are being identified, and the actual assertion is also probabilistic. In other words, fixed the node v in the input uncertain graph, each node u in the anonymized graph has a degree of being the image of v, which is probabilistic in nature. Meanwhile, the matching assertion is done by comparing two probability objects.

5.1.1 Adversary Knowledge

We proceed to present the formal definition. Let us consider the knowledge that an adversary may extract from an uncertain graph about a given target vertex in $\mathcal{G}$. Following the literature, we assume that the adversary knows some vertex property P. In this work, we assume such property P is the degree. Given an uncertain graph $\mathcal{G}$, each $v \in \mathcal{G}$ and the degree value ω, we define the probability that $X_v(\omega)$ that v originated from a vertex in G with property value ω. Specifically,

$$X_v(\omega) = \sum_{G \in W(\mathcal{G})} Pr(G) \cdot \mathcal{X}_{v,\omega}(G)$$

where $Pr(G)$ is the probability that the possible world G is observed, and $\mathcal{X}_{v,w}(G)$ is 0-1 variable that indicates the vertex v has the degree value ω in the possible world G. In other world, $X_v(\omega)$ is the sum of probabilites of all possibel worlds in which the vertex v has

$\mathbf{X}_v(\omega)$	deg $= 0$	deg $= 1$	deg $= 2$	deg $= 3$
a	0.006	0.092	0.398	0.504
b	0.054	0.348	0.542	0.056
c	0.020	0.260	0.720	0.000
d	0.180	0.740	0.080	0.000

Table 5.1: The matrix $X_v(\omega)$ for the uncertain graph in Figure 5.1 and the degree property.

the given property value ω. We define the degree value of a node v in an uncertain graph d_v as an random variable with a probability distribution X_v. The probabilties $X_v(\omega)$ may be arrange in a $n \times |\Omega|$ matrix, where each row corresponding to one vertex $vin\mathcal{G}$ and it gives the probability distribution X_v.

Example 2 *Consider the uncertain graph in Figure 5.1 and assume the vertex property is degree. Table 5.1 gives the corrsponding matrix $X_v(\omega)$, in which each row gives the probability distribution regrading the dgree of the corresponding vertex in $\mathcal{G}$. For instance, the probability that a has degree 3 is $0.9 \cdot 0.7 \cdot 0.8 = 0.504$.*

5.1.2 Re-identification Attack

The degree value of a node u in the anonymized uncertain graph is defined as the corresponding random variable d_u. The matching assertion is evaluated as the probability of the event two random variable d_u and d_v is equal. Specifically,

$$\begin{aligned} F_u(v) = P(d_u = d_v) &= \sum_{\omega} P(d_u = \omega) \cdot P(d_u = \omega) \\ &= \sum_{\omega} X_v(\omega) \cdot X_u(\omega) \end{aligned}$$

The probabilities F_u^v may be arrange in a $n \times n$ matrix, where each row corresponds to one vertex $u \in \tilde{\mathcal{G}}$ and it gives the corresponding probability F_u^v over all possible target nodes $v \in V_{\mathcal{G}}$. The columns of that matrix are proportional to the probability distribution that correponding to victim nodes. More precisely, the normalized column corresponding to a target node v, *i.e.*,

$$Y_v(u) := \frac{F_u(v)}{\sum_{u \in \tilde{G}} F_u^v}$$

is the probability that u is the image in $\tilde{\mathcal{G}}$ of a vertex that had property d_v in $\mathcal{G}$.

$\mathbf{F_u^v}$	v_a	v_b	v_c	v_d
u_a	0.42092	0.27628	0.3106	0.101
u_b	0.27628	0.42092	0.4818	0.3106
u_c	0.3106	0.4818	0.5864	0.2536
u_d	0.101	0.3106	0.2536	0.5864

Table 5.2: The matrix F_u^v for the uncertain graph and itself in Figure 5.1 and the degree property.

$\mathbf{Y_u^v}$	v_a	v_b	v_c	v_d
u_a	0.37962	0.18548	0.19027	0.08069
u_b	0.24917	0.28257	0.29515	0.24816
u_c	0.28012	0.32344	0.35922	0.20262
u_d	0.09109	0.20851	0.15535	0.46852

Table 5.3: The matrix Y_u^v for the uncertain graph and itself in Figure 5.1 and the degree property.

Example 3 *For example, if in the anonymization process we do nothing, then the perturbed output* $\tilde{\mathcal{G}} = 1$*. In the case, the probability matrix induced by* $\tilde{\mathcal{G}}$ *equals to the one induced by* $\mathcal{G}$*. Table 5.2 gives the corresponding matrix* F*. For instance, the probability that* d_a *is equal to* d_c *is* $0.006 \cdot 0.02 + 0.092 \cdot 0.26 + 0.398 \cdot 0.72 + 0.504 \cdot 0 = 0.3106$*. After normalization them, give the corresponding* $Y_u(v)$ *distributions for each vertex* v *in Table 5.3. For instance, if we look for a vertex that has the same degree distribution of* Ana *in* $\mathcal{G}$*, it is either* a*, with probability around* 0.38*,* b *with probability around* 0.25*,* c *with probability around* 0.28*, or* d *with probability around* 0.09*.*

5.1.3 Privacy Notation

Likewise, we can define our notion of privacy .

Definition 7 ***(k, ϵ)-obf [12]*** *Let* P *be the degree distribution,* $k \geq 1$ *be a desired level of anonymity, and* $\epsilon > 0$ *be a tolerance parameter. The uncertain graph* $\tilde{\mathcal{G}}$ *is said to* k*-obfuscate a given vertex* $v \in V_{\mathcal{G}}$ *with respect to* P *if the entropy of the distribution* Y_v *over the vertices of* $\mathcal{G}$ *is greater than or equals to* $\log_2 k$*:*

$$H(Y_v) \geq \log_2 k.$$

The uncertain graph $\mathcal{G}$ *is* (k, ϵ)*-obf with respect to property* P *if it* k*-obfuscates at least* $(1 - \epsilon)|V|$ *vertices in* $V_{\mathcal{G}}$*.* P *can be any node properties.*

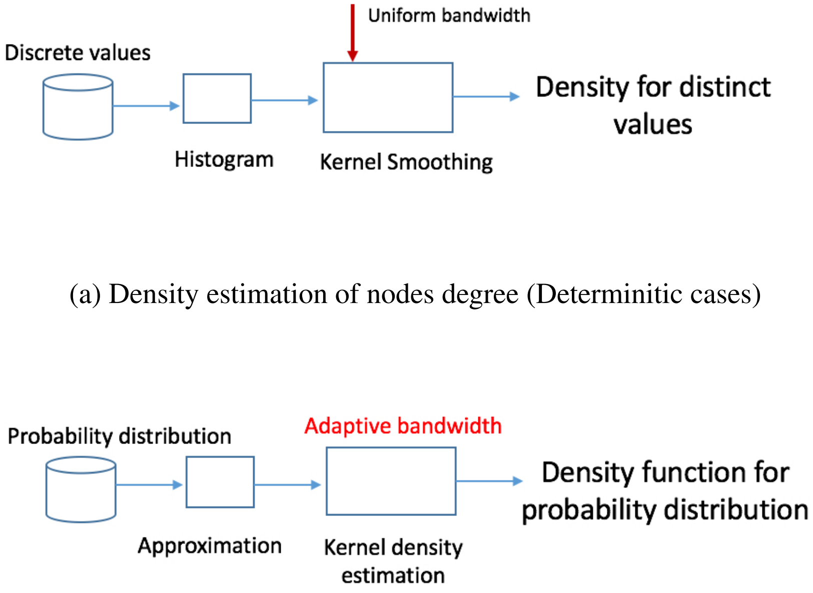
Namely, given the considered attack scenario, in which the adversary uses the degree distribution information of his target vertex v, we wish to lower bound the entropy of the distribution it induced over the perturbed graph vertices by $\log_2 k$. The general idea is exact the same with (k, ϵ)-*obf*, proposed in the literture [12]. We extend the concept of equivalent class to probabilitic scenerio.

5.2 Probabilitic Degree Anonymization

The uncertainty injecting scheme resembles the degree anonymization method designed for the deterministic graph. One common heuristics is to select the perturbation budget for each selected edge $e = (u, v)$, depending on properties of the vertices u and v. The perturbation will be larger for edges that connect unique vertices, which, require higher levels of uncertainty to “blend in the crowd", and smaller for edges that connect more “typical" vertices. Note that, it requires one effective method to capture how typical a given vertex is among all the vertices in the uncertain graph, in terms of vertex property P. In this work, let us consider vertex property P be node degree. In this case, namely, it is related to cluster uncertain data (distribution). The key feature of our method is to incorporate the statistical distance function for calculating the uniqueness level of vertices with regards to their node degree.

5.2.1 Uniqueness Score of Vertices

For vertex properties of interest, such as degree, the majority of vertices in the real uncertain graphs are already anonymous even without random perturbation. The phenomena inherit from the one in the deterministic graph. The majority of vertices in the graph have a low degree (≤ 10) with quite similar distribution. Hence, we aim at controlling the amount of applied perturbation, so that larger perturbation is added at vertices that are less anonymized in the original graph, namely, outlier point with respects to its degree distribution. In particular, we suggest calibrating the perturbation applied to an edge $e = (u, v) \in E_c$ according to the “uniqueness" of the two vertices u and v with respect to the property P. Namely, if both $P(u)$ and $P(v)$ are in the dense region, then r_e should be very small; on the other hand, if $P(u)$ and $P(v)$ are outlier values, then r_e should be higher. The definition is quite similar to the “uniqueness score", proposed in [] while extended for dealing with uncertain objects.



(a) Density estimation of nodes degree (Determinitic cases)

(b) Density estimation of nodes degree (Uncertain cases)

Figure 5.2: Comparison of determinitic and uncertain vertex density estimation in terms of node degree.

Let P: $V \to \Omega$ be node degree defined on the set of vertices V in an uncertain graph. Clearly, ω denotes uncertain object (random variable). Further, consider a distance function d between random variables in the range Ω. So, for each pair of random variables,p and q, distance $d(p, q) \geq 0$ is defined. For the degree property P_1, a natural candidate of statistical distance function is Bhattacharyya distance. For probability distribution p and q over the same domain X, the Bhattacharyya distance is defined as:

$$D_B(p, q) = -ln(\sum_x \sqrt{p(x) \cdot q(x)})$$

Before the computing of statistical distance between probability distributions, we need to get the probability distribution for all the vertices in the uncertain graph. The probability distribution of d_v may be computed exactly in time $O(n^2)$ or be approximated in time $O(n)$. The statistical distance between two probability distribution may be computed exactly in time $O(n)$. The overall complexity is $O(n^3)$. Clearly, it is not suitable for large uncertain graphs. Note that, the uniqueness function is used for locating outlier points. For the degree property, they are nodes with high degree. In such case, we may adopt an alternative approach. Since the d_v is the sum of independent random variable, it may be approximated by the normal distribution $N(\mu, \sigma^2)$, where $\mu = \sum p_{e_i}$ and $\sigma^2 = \sum p_{e_i} \cdot (1 - p_{e_i})$ as implied by the Central Limit Theorem. The Central Limit Theorem becomes effective already for $n \approx 30$. For typicals size of n in large uncertain graphs, the normal approximation becomes very accurate. According to the normal approximation, the Bhattacharyya distance between two normal distribution can be cacluated [] by exacting the mean and variances of two separate distribution objects:

$$D_B(p,q) = \frac{1}{4} \cdot \ln \frac{1}{4}(\frac{\sigma_p^2}{\sigma_q^2} + \frac{\sigma_q^2}{\sigma_p^2} + 2) + \frac{1}{4} \cdot (\frac{(\mu_p - \mu_q)^2}{\sigma_p^2 + \sigma_q^2})$$

By this way, we map the object (probability distribution) from high dimension to 2D space. It speed up the computation of “uniqueness score”. The overall time complexity is $O(n^2)$ where n is the number of vertices in the uncertain graph. The quartic complexity is not suitable for dealing with large networks.

Note that, the definition of the uniqness score is related to the kernel density estimation. For the specific method proposed in the literture [11], it adopts the gaussian function as a weighting function (kernel), the absolute difference as a distance function and the parameter θ for setting bandwidth. H For computing the distance between probability distributions, the statistical distance functions were choosed. The remaining issue is the choice of bandwidth. Note that, the choice of bandwidth has the siginificant effect on the shape of the corresponding kernel density estimator. If the bandwidth is small, we will obtain an under smoothed estimator, with high variability. On the contrary, if the value of bandwidth h is big, the resulting estimator will be over smooth and farther from the real data distribution that we are trying to estimate. Notice of the connection between uniqueness and density estimation, we adopt the adaptive bandwidth kernel density estimation method, *i.e.*, we vary the w of the kernel in different regions of the sample space. By this way, we can get the statistical robust estimation of the distribution of all the vertices in the input uncertain graph in an efficient way.

5.2.2 Proposed Research Tasks

- We identify and formulate the node re-identification attack using their degree probability distribution, and extend the (k, ϵ)-*obf* notation for quantifying privacy level. We present the reliability and privacy preserving problem in the context of uncertain graphs.

- We show case that a naive approach that simply combines existing techniques do not work in practice due to significant utility loss.

- We propose a heuristic based on the density estimation of probability distributions among all the vertices in the uncertain graph for injecting perturbation judiciously. We give an efficient and effective method for heuristic evaluation.

- We propose to explore different search strategy for solving the induced optimized problem in the process of injecting uncertainty.

- We perform an extensive experimental evaluation using four real-world uncertain graph data sets from different domains. We evaluate and compare our methods using different groups of graph metrics.

Chapter 6

Research Schedule

It is planned to finish the dissertation work by May 2017. The tentative research schedule is listed as follows. Note that the schedule is made based on current progress.

Time	Tasks
Now - March 2017	Degree Distribution Anonymization on Uncertain graphs: Explore the uncertain graph mofiication techniques for anonymizing degree distribution; Explore the randomized strategy for justificious uncertain graph modifiaction; Implement our proposed techniques; Conduct performance evaluation; Write paper
March 2017 - May 2017	Dissertation writing
May 2017	Dissertation defense

I plan to write the following paper.

- “Degree Distribution Anonymization on Uncertain Graphs” to VLDB 2017

Bibliography

[1] E. Adar and C. Re. Managing uncertainty in social networks. *IEEE Data Eng. Bull.*, 2007.

[2] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 1997.

[3] S. Arifuzzaman, M. Khan, and M. Marathe. Patric: A parallel algorithm for counting triangles in massive networks. *CIKM*, pages 529–538, 2013.

[4] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: Anonymized social networks, hidden patterns, and structural steganography. *WWW*, (12):133–141, Dec. 2011.

[5] M. O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, 1986.

[6] N. Bao and T. Suzumura. Towards highly scalable pregel-based graph processing platform with x10. *WWW*, pages 501–508, 2013.

[7] V. Batagelj and A. Mrvar. A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social networks*, 2001.

[8] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. *KDD*, pages 16–24, 2008.

[9] J. Berry, B. Hendrickson, R. LaViolette, and C. Phillips. Tolerating the community detection resolution limit with edge weighting. *Physical Review E*, 2011.

[10] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. *Proc Vldb Endow*, 2009.

[11] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting uncertainty in graphs for identity obfuscation. *SIGMOD*, 2012.

[12] F. Bonchi, A. Gionis, and T. Tassa. Identity obfuscation in graphs through the information theoretic lens. *ICDE*, 2014.

[13] L. Buriol, G. Frahling, and S. Leonardi. Counting triangles in data streams. *PODS*, pages 253–262, 2006.

[14] J. Casas-Roma. Privacy-preserving on graphs using randomization and edge-relevance. *Modeling Decisions for Artificial Intelligence*, 2015.

[15] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 1985.

[16] Colbourn and Colbourn. The combinatorics of network reliability. 1987.

[17] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *VLDB*, 2008.

[18] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 2008.

[19] J. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Academy of Sciences*, 2002.

[20] J. Gonzalez, R. Xin, A. Dave, and D. Crankshaw. Graphx: Graph processing in a distributed dataflow framework. *GRADES,SIGMOD workshop*, pages 599–613, 2014.

[21] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. *OSDI*, pages 17–30, 2012.

[22] W.-S. Han, S. Lee, K. Park, J.-H. Lee, M.-S. Kim, J. Kim, and H. Yu. Turbograph: a fast parallel graph engine handling billion-scale graphs in a single pc. *KDD*, pages 77–85, 2013.

[23] S. Hartung and N. Talmon. The complexity of degree anonymization by graph contractions. *TAMC*, 2015.

[24] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava. Anonymizing social networks. 2007.

[25] X. Hu, Y. Tao, and C.-W. Chung. I/o-efficient algorithms on triangle listing and counting. *ACM Trans. Database Syst.*, 39(4), 2014.

[26] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM*, 1978.

[27] Z. Jorgensen, T. Yu, and G. Cormode. Publishing attributed social graphs with formal privacy guarantees. pages 107–122, 2016.

[28] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. 2003.

[29] G. Keramidas and P. Petoumenos. Cache replacement based on reuse-distance prediction. *ICCD*, 2007.

[30] S. Khuller and B. Saha. On finding dense subgraphs. *Automata*, 2009.

[31] J. Kim, W. Han, S. Lee, K. Park, and H. Yu. Opt: a new framework for overlapped and parallel triangulation in large-scale graphs. *SIGMOD*, pages 637–648, 2014.

[32] N. Krogan, G. Cagney, H. Yu, G. Zhong, and X. Guo. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 2006.

[33] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. *OSDI*, pages 31–46, 2012.

[34] K. Liu and E. Terzi. Towards identity anonymization on graphs. *SIGMOD*, 2008.

[35] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[36] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, and D. Chklovskii. Network motifs: simple building blocks of complex networks. *Academy of Sciences*, 2002.

[37] P. Mittal, C. Papamanthou, and D. Song. Preserving link privacy in social network based systems. *NDSS*, 2013.

[38] F. Nagle, L. Singh, and G. Aris. *EWNI: Efficient Anonymization of Vulnerable Individuals in Social Networks*. 2012.

[39] A. Narayanan and V. Shmatikov. De-anonymizing social networks. pages 173–187, 2009.

[40] H. Nguyen, A. Imine, and M. Rusinowitch. Anonymizing social graphs via uncertainty semantics. *CCS*, 2015.

[41] M. Ninggal and J. H. Abawajy. Utility-aware social network graph anonymization. *J Netw Comput Appl*, 2015.

[42] Parchas, Gullo, Papadias, and Bonchi. The pursuit of a good possible world: extracting representative instances of uncertain graphs. *SIGMOD*, 2014.

[43] H. Park, F. Silvestri, U. Kang, and R. Pagh. MapReduce triangle enumeration with guarantees. *CIKM*, pages 1739–1748, 2014.

[44] P. Petoumenos and G. Keramidas. Instruction-based reuse-distance prediction for effective cache management. *MSP*, pages 60–68, 2009.

[45] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *VLDB*, 2010.

[46] T. Schank. Algorithmic aspects of triangle-based network analysis. *Phd in computer science*, 2007.

[47] Y. Shao, B. Cui, L. Chen, L. Ma, J. Yao, and N. Xu. Parallel subgraph listing in a large-scale graph. *SIGMOD*, pages 625–636, 2014.

[48] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. *KDD*, pages 607–614, 2011.

[49] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga. Arabesque: a system for distributed graph mining. *SOSP*, pages 425–440, 2015.

[50] B. Thompson and D. Yao. The union-split algorithm and cluster-based anonymization of social networks. *ASIACCS*, 2009.

[51] Y. Wang, L. Xie, B. Zheng, and K. C. K. Lee. Utility-oriented k-anonymization on social networks. *DASFAA*, 2011.

[52] T. White. Hadoop: The definitive guide. 2010.

[53] W. Wu, Y. Xiao, W. Wang, Z. He, and Z. Wang. k-symmetry model for identity anonymization in social networks. *EDBT*, 2010.

[54] D. Xiao, M. Y. Eltabakh, and X. Kong. Reliability-preserving anonymization on uncertain graphs.

[55] D. Xiao, M. Y. Eltabakh, and X. Kong. Bermuda: An efficient mapreduce triangle listing algorithm for web-scale graphs. *SSDBM*, pages 10:1–10:12, 2016.

[56] X. Ying, K. Pan, X. Wu, and L. Guo. Comparisons of randomization and k-degree anonymization schemes for privacy preserving social network publishing. *SNA-KDD*, 2009.

[57] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. pages 739–750, 2008.

[58] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. *ICDE*, 2008.

[59] L. Zou, L. Chen, and M. Özsu. K-automorphism: A general framework for privacy preserving network publication. *Proc. VLDB Endow.*, 2(1):946–957, 2009.