Discovering Top-k Reliable Subgraphs in Uncertain Graphs

Bachelor report (15 ECTS) in Computer Science
Department of Computer Science, Aarhus University

Students
Sebastian Bugge Loeschcke (20180446)
Mads Toftrup(201806005)
Christian Leth-Espensen(201805807)

Advisors
Cigdem Aslay
Panagiotis Karras

June 18, 2022

Abstract

Uncertain graphs are widely used for representing the inherent uncertainty in complex systems. This project reviews and extends the work of Jin et al. [1], by studying the problem of discovering the k most reliable subgraphs in uncertain graphs. This problem is relevant in the analysis of many network applications, such as social networks, network routing, and protein-protein interaction networks. As the problem is #P-complete, and thus computationally intractable, a sampling scheme is introduced to provide ϵ-approximations with high probability. In doing this, the problem is transformed to the problem of discovering the k most frequent cohesive sets in the setting of deterministic graphs. This transformation gives way to an *eager* and *attentive* approach. The eager approach combines peeling techniques and an iterative version of the Apriori algorithm to give approximate solutions with certain ϵ-guarantees. To ensure that the correct top-k results are found with high probability, the attentive approach builds on top of the eager approach by adding a progressive sampling step. The eager and attentive approaches allow for a trade-off between efficiency and precision. Extensive experiments verify that the novel approaches provide significant improvements compared to naively extending the work of Jin et al. to the top-k setting.

Contents

1 Introduction

Graphs are universal representations used within several scientific disciplines and domains. Graphs represent entities (vertices) and their relationships (edges). The relationship between two entities is sometimes uncertain due to noisy measurements [2], inconsistent and incorrect information sources [3], or because of the dynamic nature of topological structures [4]. Such graphs are called uncertain or probabilistic graphs. The edges of an uncertain graph have labels that indicate the probability of the edge existing in the graph. Uncertain graphs give us a way of representing the confidence for which it is believed that a relationship between two vertices exists in reality [5].

Graph data needs intelligent tools to analyze and gain insightful knowledge. Research on graphs provides many different algorithms for exact graph mining. However, not as much research has been put into graph mining algorithms for uncertain graphs. Since uncertain graphs are useful in many applications, it is important to develop state-of-the-art algorithms and systems that explicitly model uncertainty [3].
Recently, graph mining in uncertain graphs has attracted significant research attention. This includes studies examining problems such as frequent pattern mining [6], queries for finding shortest paths [7] and finding top k-cliques [8].

In our work, we will focus on finding the k most reliable subgraphs in uncertain graphs. This problem focuses on identifying the k induced subgraphs that have the k highest probabilities of being connected in the uncertain graph. Jin et al. present novel solutions to finding subgraphs with a reliability above a given threshold [1]. In our work, we will present the work of Jin et al., and then, using the same semantics, we will present the problem of finding the k most reliable subgraphs. Our work consists of different approximation approaches for solving this problem, where each approach will provide different statistical guarantees and efficiencies in regard to the associated algorithms. The different approaches offer a choice between precision and efficiency. To the best of our knowledge, no previous work exists in the literature that tries to solve the same problem.

The problem of finding reliable subgraphs in uncertain graphs is useful in a wide range of network applications. For instance, in large social networks, uncertain graphs with edge probabilities may be used to discover cohesive groups of users [9]. Additionally, in a biological setting, highly reliable subgraphs can be used to identify the core of protein complexes in protein-protein interaction networks [10].

2 Related Work

We will now discuss the previous work related to our project. This section will focus on algorithms for finding the most reliable subgraphs in uncertain graphs and different definitions of reliability. Furthermore, we will discuss how our work relates to existing top-k mining approaches.

2.1 The reliability problem

A fundamental problem in the context of uncertain graphs is the *reliability problem*. This problem aims to compute the probability that sets of vertices in a network are connected and to

estimate the quality of these connections. In the literature, there exist diverse meanings of the reliability problem on uncertain graphs. A variant of the problem is *reliability detection*, where the goal is to measure the probability that certain events occur [11], e.g., the *network reliability problem* [12], also called the all-terminal reliability problem [13]. This problem looks at reachability and tries to determine the probability that all pairs of vertices in an uncertain graph are reachable from one another. Other problems are concerned with *reliability search*. Here, the goal is to discover all vertices with highly reliable connections from a source vertex [14] or compute all vertices that are reachable from a set of queried vertices with probability no less than a given threshold [11]. Many different approaches have been proposed to approximate solutions to these problems [1, 11, 14]. The solutions devised for these problems use ϵ-approximations and Monte-Carlo sampling in a way that provides statistical guarantees on the precision of the result. Our project will use a similar approach.

We will look at the *reliability detection* problem of discovering all highly reliable subgraphs in uncertain graphs. This problem has previously been addressed as finding all maximal highly reliable sets of vertices, where the induced subgraphs have a network reliability above a user-specified threshold [1]. However, there is no principled way of deciding the right threshold value, which makes it a task that requires domain expertise. Choosing the right threshold is critical since it greatly affects the number of candidates to evaluate. A too high threshold will result in finding too few reliable subgraphs, and thus some interesting subgraphs will be missed. A too low threshold will generate a large number of reliable subgraphs, which will increase the algorithm's time and space consumption and make it hard for the user to interpret and analyze the result. A consequence of this is that the algorithm may be run several times to find an appropriate threshold value that generates an analyzable output. Instead, we aim to develop a top-k approach that does not require a user-specified reliability threshold but finds the k most reliable induced subgraphs.

2.2 Mining top-k frequent patterns

In pattern and subgraph mining, one tries to find useful information in graph networks. This is often done by setting a minimum threshold, which is used to extract meaningful patterns and subgraphs that have a desired value above this threshold. As described in the previous section, the user-specified threshold accordingly affects the size of the mining result and runtime of the algorithm. To address this problem, *top-k queries* are useful. Rather than using a minimum threshold, the user specifies the desired number k of interesting patterns or subgraphs that must be included in the mining result.

A typical approach to top-k mining, used in the literature [15–17], is to save the values of the top-k items discovered so far and then use the k'th item as the minimum threshold. Hereby, the first mining operations are conducted with a very low threshold value, and the threshold is then updated dynamically. Many algorithms for deterministic graphs have been adapted to the context of uncertain graphs, e.g. top-k algorithms have been proposed for k-nearest neighbors [18], and also for reliability search [14]. However, the problem described in the paper by Zhu et al. [14] is significantly different from our problem formulation. They are trying to solve the following problem: given an uncertain graph $\mathcal{G}$, a source vertex s, and an integer $k > 0$, the goal is to find the most reliable subgraph of k vertices $V = v_1, ..., v_k$ including the vertices in V and s. Instead, we try to find the k most reliable subgraphs among all subgraphs in $\mathcal{G}$ without specifying any source vertex. In the paper by Zhu et al. [14], they also generalize their top-k reliability search problem definition to the multi-source case, where multiple source vertices are

specified as input instead of a single vertex. Even with the generalization to the multi-source approach, their algorithm cannot be easily adapted to the problem we are trying to solve. Our definition of top-k relates to the k most reliable subgraphs, with only one constraint stating that the size of the subgraph is greater than two vertices. Instead, their definition of top-k constrains the output to have k vertices.

Top-k mining approaches in the literature [15, 16] typically use the *anti-monotonicity* property (also called *Apriori* property or *downward closure* property) to reduce the search space of the mining task and improve the runtime [19]. The anti-monotonicity property states that all supersets of an infrequent set are also infrequent [3]. As shown in the work by Jin et al. [1], the frequent cohesive sets that we are trying to discover do not have this property. An approach to compensate for not having this property is to use effective overestimation models, to reduce the number of candidates for performance improvements [19]. We have also used overestimation models to create an effective threshold raising strategy. Here we have used the same peeling approach as described in [1] and utilized that we can use frequent linked sets to upper bound the frequency of cohesive sets.

One of the main challenges of using sampling to approximate computational heavy mining tasks is to ensure with high probability that we find all itemsets that we are looking for. Jin et al. proposed a sampling scheme that gives guarantees on expected fractions of missed highly reliable subgraphs with some probability δ based on a user-specified threshold [1]. Using sampling to approximate top-k queries is a more challenging task since we do not know the threshold in advance [20]. Pietracaprina et al. presented a novel sampling approach for effective top-k frequent itemsets mining [20]. They define an ϵ-approximation to the top-k itemsets as a family of itemsets and their estimated frequencies. They guarantee that the k returned itemsets have a frequency of at least $f_k - \epsilon$ and that no itemset with a frequency above $f_k + \epsilon$ is missed, where f_k is the frequency of the k'th most frequent itemset. We will use a similar definition in our novel approach.

3 Preliminaries and review of literature

In this section we will review the work of Jin et al. [1] and introduce the terminology used in the rapport.

3.1 Table of notation

$\mathcal{G} = (V, E, P)$	Uncertain graph
$G_i \sqsubseteq \mathcal{G}$	Possible graph
$G[V_s]$	Induced subgraph of the possible graph G that consists of the vertex set V_s
$\mathcal{G}[V_s]$	Induced subgraph of the uncertain graph $\mathcal{G}$ that consists of the vertex set V_s
$Pr[G]$	Probability of the possible graph
$R[\mathcal{G}]$	Reliability of an uncertain graph
$R[\mathcal{G}[V_s]]$, $R[V_s]$	Subgraph reliability with respect to V_s
$\hat{R}[\mathcal{G}[V_s]]$, $\hat{R}[V_s]$	Subgraph reliability estimate with respect to V_s
$\hat{R}_L[\mathcal{G}[V_s]]$	Linked subgraph reliability estimate with respect to V_s
S_a	Family of all highly reliable vertex sets
$\mathcal{K}$	Set of the true k most reliable subgraphs
$\hat{\mathcal{K}}$	Set of subgraphs with reliability estimates above $\hat{R}[m_{\hat{k}}] - 2\epsilon$
m_k	Subgraph with the true k'th reliability
$m_{\hat{k}}$	Subgraph estimated to be have the true k'th reliability

Table 1: Table of notation

3.2 Uncertain graphs and reliability

In the uncertain graph model, each edge is associated with an edge existence probability. We denote an uncertain undirected graph as $\mathcal{G} = (V, E, P)$, where V is a set of vertices, E is a set of edges, and $P : E \to (0, 1]$ is a function that determines the probability of existence of an edge in the graph [1]. In this project, we will work under the assumption that edge probabilities are independent of each other.

A possible graph $G = (V_G, E_G)$ of an uncertain graph $\mathcal{G}$ is a deterministic graph, which is an outcome of the random variables representing the edges. We denote a possible graph $G \sqsubseteq \mathcal{G}$. In a given uncertain graph, $2^{|E|}$ different possible graphs exist. i.e. a possible graph for each combination of existing edges. The probability of a possible graph G is given by the following sampling probability [1]:

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

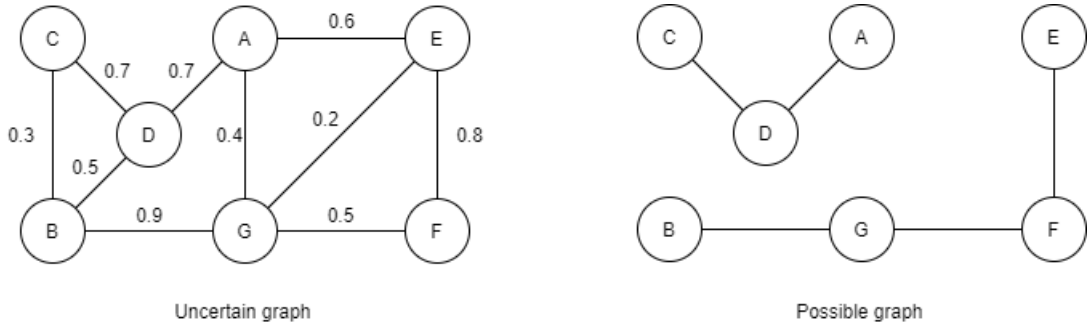


Figure 1: Example of an uncertain graph and a corresponding possible graph with a probability of 0.0118541

Figure 1 shows an uncertain graph and a corresponding possible graph. The uncertain graph has 2^{10} possible graphs, where the sampling probability of the possible graph in the figure is defined as follows:
$Pr[G] = p(C,D) \cdot p(A,D) \cdot p(B,G) \cdot p(F,G) \cdot p(E,F) \cdot (1-p(B,C)) \cdot (1-p(B,D)) \cdot (1-p(A,G)) \cdot (1-p(A,E)) \cdot (1-p(E,G)) = 0.0118541$

In the highly reliable subgraph problem presented in [1], the goal is to determine subgraphs of an uncertain graph that have a network reliability (abbreviated as reliability) above a certain threshold. The network reliability is a measurement for the likelihood that the entire graph is connected in a sampled possible graph G. The network reliability of an uncertain graph can be defined as follows:

DEFINITION 1 (**Network Reliability** [1]) *Given an uncertain graph $\mathcal{G} = (V, E, P)$, its network reliability $\boldsymbol{R}[\mathcal{G}]$ is defined as the probability that its sampled realizations remain connected. This probability can be mathematically quantified as follows:*

$$R[\mathcal{G}] = \sum_{G \sqsubseteq \mathcal{G}} I(G) \cdot \Pr[G]$$

where $\boldsymbol{I}(G)$ is an indicator function of value 1 if G is connected, and 0 otherwise.

Definition 1 can be generalized to the induced subgraph $\mathcal{G}[V_s]$ for a subset of vertices in $\mathcal{G}$ [1]:

$$R[\mathcal{G}[V_s]] = \sum_{G \sqsubseteq \mathcal{G}} I(G[V_s]) \cdot Pr[G]$$

where $G[V_s]$ is the induced subgraph of V_s and $R[\mathcal{G}[V_s]]$ is the subgraph reliability with respect to the subset $V_s \subseteq V$ [1]. Note that a subgraph $\mathcal{G}[V_s]$ of $\mathcal{G}$ is induced, if for any two vertices u and v in $\mathcal{G}[V_s]$, u and v are adjacent (connected by an edge), if and only if they are adjacent in $\mathcal{G}$ [21].
Jin et al. uses Definition 1 to define the following problem:

DEFINITION 2 (**Highly Reliable Subgraph(HRS) Problem** [1]) *Given an uncertain graph $\mathcal{G} = (V, E, P)$ and a reliability threshold $\alpha \in [0, 1]$, determine all induced subgraph whose network reliability is at least α.*

This can also be formulated as finding all subgraphs, denoted as S_α, for $V_s \subseteq V$ where $R[\mathcal{G}[V_s]] \geq \alpha$. An important observation, described by Jin et al., is that the anti-monotonicity property does not hold for reliable subgraphs [1]. Hereby, it is not implied that all subgraphs of a reliable subgraph are also reliable:

LEMMA 1: [1] *Given an uncertain graph $\mathcal{G}$ and two subsets V_s and V_s', where $V_s' \subseteq V_s$, we can neither claim that $R[\mathcal{G}[V_s]] \geq R[\mathcal{G}[V_s']]$ or $R[\mathcal{G}[V_s]] \leq R[\mathcal{G}[V_s']]$.*

As an example of Lemma 1, consider the graph in Figure 1. Here, we have that $R[\mathcal{G}[\{B, C, D\}]] = 0.5 > R[\mathcal{G}[\{B, C\}]] = 0.3$ and $R[\mathcal{G}[\{A, G, F\}]] = 0.2 < R[\mathcal{G}[\{G, F\}]] = 0.5$. Note that the formula from Definition 1 is needed to find the exact reliability. To use this, one has to iterate over all $2^{|E|}$ possible graphs. It is infeasible to perform these calculations on larger graphs, and hence an approximation approach is necessary.
As discussed in the Related Work section, the lack of anti-monotonicity raises a challenge when devising algorithms for finding reliable subgraphs. This is because pruning potential candidates from the search space becomes non-trivial.

Jin et al. use another property of uncertain graphs as part of a preprocessing step, that makes it possible to prune certain edges in order to reduce the computational complexity of the problem. This property can be stated as: *Given an uncertain graph $\mathcal{G} = (V, E, P)$ and a reliability threshold α, for an induced subgraph $\mathcal{G}[V_s]$ of $\mathcal{G}$, if there exists an edge cut $C \subseteq E(\mathcal{G}[V_s])$ in $\mathcal{G}[V_s]$(where C makes the subgraph $\mathcal{G}[V_s]$ disconnected), such that $P(C) = \sum_{e \in C} p(e) < \alpha$, then $R[\mathcal{G}[V_s]] < \alpha$.* From this property, the following lemma can be stated:

LEMMA 2 - **Edge-Cut Lemma** [1]: *Given an uncertain graph $\mathcal{G} = (V, E, N)$ and a reliability threshold α, for any edge cut C in $\mathcal{G}$, where $C \subseteq E$ and C divides the uncertain graph into two parts $\mathcal{G}[V_1]$ and $\mathcal{G}[V_2]$($V_1 \cup V_2 = V$ and $C \subseteq V_1 \times V_2$), then if $p(C) = \sum_{e \in C} p(e)$, then, for any highly reliable subgraph $\mathcal{G}[V_s]$, either it holds that $V_s \subseteq V_1$ or $V_s \subseteq V_2$. This can also be formulated as: there is no highly reliable subgraph $\mathcal{G}[V_s]$ with $V_s \cap V_1 = \emptyset$ and $V_s \cap V_2 = \emptyset$*

The implications of this Lemma 2 is that if there exists a cut C in the uncertain graph $\mathcal{G}$, where $p(C) < \alpha$, then we can safely drop all the edges in C and be sure that we are not missing any highly reliable subgraphs [1]. Hereby, Jin et al. can use Lemma 2 to reduce the number of candidate sets that we consider. Jin et al. find all such cuts by applying a simple minimum-cut algorithm that starts from each vertex in the uncertain graph and checks if any edges can be dropped. Specifically, their algorithm starts from an arbitrary vertex or vertex set in the graph and iteratively selects the vertex most tightly connected to the current set of vertices and adds it to the set. This process continues until a cut C with $p(C) < \alpha$ is discovered, which makes it possible to drop all edges in the cut [1]. This procedure is then repeated for all vertices in the uncertain graph as part of a preprocessing step.

The Edge-Cut Lemma is a very useful preprocessing step in the setting of a threshold based algorithm, where the goal is to find all highly reliable subgraphs above a user-specified threshold. However, in the setting of finding the k most reliably subgraphs using effective threshold raising strategies, it is non-trivial to adapt the Edge-Cut Lemma. Our top-k approach will start with an initial threshold of 0, and thus no edges can be dropped. After raising the threshold, we would have to apply the minimum-cut algorithm and find edges that can be dropped in the uncertain graph $\mathcal{G}$. Then, we would have to create new samples with the updated edges. However, this resampling would require the algorithm to start from scratch again since the re-

liabilities of all candidate sets generated could have changed due to the dropped edges.

3.3 Sampling

The network reliability problem is #P-complete [1]. There exist methods that can exactly determine the reliability of a subgraph. However, these methods are designed for small graphs with at most ten vertices [22]. Jin et al. propose a sampling scheme that makes it possible to approximate the discovery of highly reliable subgraphs in uncertain graphs with a guaranteed probabilistic precision [1].

The sampling approach can be described as follows: N possible graphs, $G_1, G_2, ..., G_N$, are sampled from the uncertain graph, $\mathcal{G}$, using Monte-Carlo sampling. For any subset of vertices V_s an indicator function $I(G_i[V_s])$ checks if the set of vertices V_s is connected in the graph G_i. From this sampling scheme Jin et al. define an unbiased sampling estimator, $\hat{R}[[G[V_s]]$ for the subgraph reliability problem as follows [1]:

$$R[\mathcal{G}[V_s]] \approx \hat{R}[\mathcal{G}[V_s]] = \frac{\sum_{i=1}^{N} I(G_i[V_s])}{N}$$

By using the Chernoff-Hoeffding Bound [23, 24], Jin et al. provide a lower bound on the sample size, which guarantees that it is possible to determine if induced subgraphs are highly reliable with a high probability [1]:

LEMMA 3 [1]. *With samples size $N \geq \frac{2}{\epsilon^2} ln \frac{2}{\delta}$, for any subset of vertices V_s, $Pr(|\hat{R}[\mathcal{G}[V_s]] - R[\mathcal{G}[V_s]]| \geq \epsilon) \leq \delta$*

The sampling approach used to determine the reliability of induced subgraphs is a multiple hypothesis test, which makes it hard to probabilistically control the number of false-negatives and false-positives [1]. This also makes it hard to provide any guarantees on recall and precision [1].
Jin et al. tackle this problem by utilizing two sets $\bar{S}$ and $\underline{S}$, to approximate the family of all highly reliable vertex sets S_a [1]. The two sets $\bar{S}$ and $\underline{S}$ provide flexibility and complementary choices dependent on the application [1]. If the focus is to not miss any highly reliable subgraphs, one can conveniently choose to use $\bar{S}$. This will include subgraphs that have a reliability estimate within the range $[\alpha - \epsilon, 1]$ with the goal of reducing the number of false-negatives. Afterward, one can filter out any subgraphs with reliability estimate in the range $[0, \alpha + \epsilon]$ on new samples, which will minimize the chance of getting false-positives. Hereby, the first set $\bar{S}$ focuses on maximizing recall, whereas the second set $\underline{S}$ focuses on maximizing precision. When both sets are similar, it is possible to achieve both high recall and precision [1]. Jin et al. sample $\bar{S}$ and $\underline{S}$ using a two-step approach:

Step 1 [1]: Sample $N_1 = \frac{2}{\epsilon^2} \ln \frac{2}{\delta}$ possible graphs from $\mathcal{G}$. From these sampled graphs, all induced subgraphs $\mathcal{G}[V_s]$ are discovered with reliability $\hat{R}[\mathcal{G}[V_s]] \geq \alpha - \epsilon$:

$$\bar{S} = \{\mathcal{G}[V_s] \hat{R}[\mathcal{G}[V_s]] \geq \alpha - \epsilon\}$$

Step2 [1]: Sample $N_2 = \frac{2}{\epsilon^2} \ln \frac{2|\bar{S}|}{\delta}$ possible graphs from $\mathcal{G}$. From these sampled graphs all in-

duced subgraphs $\mathcal{G}[V_s] \in \bar{S}$ are discovered with reliability $\hat{R}[\mathcal{G}[V_s]] \geq \alpha + \epsilon$:

$$\underline{S} = \{\mathcal{G}[V_s]\hat{R}[\mathcal{G}[V_s]] \geq \alpha + \epsilon\} \cap \bar{S}$$

The two-step sampling approach proposed by Jin et al. provides some properties in regards to precision and recall, which are formulated in the following theorem:

THEOREM 1 *properties of the sampled sets $\bar{S}$ and $\underline{S}$ [1]:*

1. The expected fraction of missed highly reliable subgraphs(false-negatives) is at most δ. Alternatively, this can be formulated as $E(\frac{S_\alpha / \bar{S}}{|S_\alpha|}) \leq \delta$;
2. With probability at least $1 - \delta$, all induced subgraphs in $\underline{S}$ are highly reliable: $Pr(\forall \mathcal{G}[V_s] \in \underline{S}, R[\mathcal{G}[V_s]] \geq \alpha) \geq 1 - \delta$ and $Pr(\frac{|\underline{S} \cap S_\alpha|}{\underline{S}} = 1) \geq 1 - \delta$;
3. With probability at least $1 - \delta$, the precision of $\bar{S}$ is no less than $\beta = \frac{|\underline{S}|}{|\bar{S}|}(\underline{S} \subseteq \bar{S}) : Pr(\frac{|S_\alpha \cap \bar{S}|}{\bar{S}} \geq \beta) \geq 1 - \delta$. If β is close to 1 it shows that $\underline{S}$ and $\bar{S}$ are very similar, which in turn increases precision;
4. The expected fraction of missed highly reliable subgraphs in $\bar{S}$ is at most $\delta + 1 - \beta + \frac{\delta\beta}{|\bar{S}|} \approx 1 + \delta - \beta$. If β is close to 1, we get that the expected false-negative rate is small.

Since the parameters δ and ϵ are used to control the precision and recall in the algorithm for discovering reliable subgraphs, they influence the quality of the approximation. Both parameters are inversely related to the sample sizes N_1 and N_2 and thereby influence the computational cost. Deciding values for the δ and ϵ is thus a consideration of the balance between computational cost and approximation quality [1].

The two-step sampling approach from Jin et al. is very useful for the threshold based algorithm. Since we are devising an algorithm for finding the top k most reliable subgraphs and not just all subgraphs above a user-specified threshold, this two-step sampling approach is not directly applicable to our case. Instead, we introduce a new metric of $Precision@k$, which will be elaborated in a later section.

3.4 Proof of Theorem 1

In this section, we prove Theorem 1, which was not included in the original paper by Jin et al.

3.4.1 Proof of Theorem 1.1

The expected fraction of missed highly reliable subgraphs is at most δ. In other words, we have that $E(\frac{|\mathcal{S}_\alpha \setminus \bar{S}|}{|\mathcal{S}_\alpha|}) \leq \delta$.

Let the number of highly reliable subgraphs be x, $|\mathcal{S}_\alpha| = x$. We can describe the set $\mathcal{S}_\alpha \setminus \bar{S}$ as the set of missed highly reliable subgraphs: $\mathcal{S}_\alpha \setminus \bar{S} = \{m : R[m] \geq \alpha, \hat{R}[m] < \alpha - \epsilon\}$. From Theorem 1 we have that: for any $m : R[m] \geq \alpha$ then $Pr[\hat{R}[m] < \alpha - \epsilon] \leq \delta$. In other words, we miss a highly reliable subgraph only when we cannot estimate its reliability within ϵ-error range, which happens with probability at most δ. The proof is then as follows:

$$E(\frac{|\mathcal{S}_\alpha \setminus \bar{S}|}{|\mathcal{S}_\alpha|}) = E(\frac{|\mathcal{S}_\alpha \setminus \bar{S}|}{x}) \tag{1}$$

$$\leq \frac{1}{x}\sum_{r=0}^{x} r \cdot \delta^r \cdot (1-\delta)^{x-r} \tag{2}$$

$$\leq \frac{1}{x}\sum_{r=0}^{x} r\binom{x}{r}\delta^r \cdot (1-\delta)^{x-r} \tag{3}$$

$$= \frac{1}{x}\sum_{r=1}^{x} r\binom{x}{r}\delta^r \cdot (1-\delta)^{x-r} \tag{4}$$

$$= \frac{1}{x}\sum_{r=1}^{x} x\binom{x-1}{r-1}\delta^r \cdot (1-\delta)^{x-r} \tag{5}$$

$$= \frac{1}{x} \cdot x \cdot \delta \sum_{r=1}^{x} \binom{x-1}{r-1}\delta^{r-1} \cdot (1-\delta)^{(x-1)-(r-1)} \tag{6}$$

$$= \frac{1}{x} \cdot x \cdot \delta \tag{7}$$

$$= \delta \tag{8}$$

Step (6) to step (7) we get from the Binomial Theorem.

3.4.2 Proof of Theorem 1.2

With probability at least $1-\delta$ all induced subgraphs in $\underline{S}$ are highly reliable subgraphs: $Pr(\forall \mathcal{G}[V_s] \in \underline{S}, R[\mathcal{G}[V_s]] \geq \alpha) \geq 1-\delta$ or equivalently: $Pr(\frac{|S_\alpha \cap \underline{S}|}{|\underline{S}|} = 1) \geq 1-\delta$.
To prove this, we describe the probability that a single subgraph m_i in $\underline{S}$ is not highly reliable and denote this event as A_i. This happens if the subgraph m was overestimated by more than ϵ, which by the Hoeffding's inequality [24], with q as the sample size, can be described as

$$Pr(A_i) := Pr(|R[m_i] - \hat{R}[m_i]| \geq \epsilon) \leq 2e^{-2q\epsilon^2}$$

We can rewrite this by inserting the sample size N_2:

$$Pr(|R[m] - \hat{R}[m]| \geq \epsilon) \leq 2e^{-2(\frac{2}{\epsilon^2}\ln\frac{2|\bar{S}|}{\delta})\epsilon^2} \tag{1}$$

$$= 2e^{-4\ln\frac{2|\bar{S}|}{\delta}} \tag{2}$$

$$= 2(e^{\ln\frac{2|\bar{S}|}{\delta}})^{-4} \tag{3}$$

$$= 2(\frac{2|\bar{S}|}{\delta})^{-4} \tag{4}$$

$$= 2(\frac{\delta}{2|\bar{S}|})^{4} \tag{5}$$

$$= \frac{\delta^4}{8|\bar{S}|^4} \tag{6}$$

The probability that one or more graphs in $\bar{S}$ does not satisfy A can be expressed as:

$$Pr(\bigcup_{i}^{|\bar{S}|} A_i)$$

Using union bound and the fact that $\delta \in [0;1]$, $|\bar{S}| \geq 1$ we have that

$$Pr(\bigcup_{i}^{|\bar{S}|} A_i) \leq \sum_{i}^{|\bar{S}|} Pr(A_i) \leq \sum_{i}^{|\bar{S}|} \frac{\delta^4}{8|\bar{S}|^4} \leq \frac{1}{8} \sum_{i}^{|\bar{S}|} \frac{\delta^4}{|\bar{S}|} = \frac{1}{8} \frac{|\bar{S}|\delta^4}{|\bar{S}|} \leq \delta$$

Hence we get that the probability that all subgraphs are highly reliable:

$$1 - Pr(\bigcup_{i}^{|\bar{S}|} A_i) \geq 1 - \delta$$

3.4.3 Proof of Theorem 1.3

The third part of Theorem 1 states: *with probability at least* $1 - \delta$ *the precision of* $\bar{S}$ is no less than $\beta = \frac{|\underline{S}|}{|\bar{S}|}$ $\underline{S} \subseteq \bar{S}$ which can be expressed as: $Pr(\frac{|S_\alpha \cap \bar{S}|}{\bar{S}} \geq \beta) \geq 1 - \delta$.

$$\begin{aligned}
Pr(\frac{|S_\alpha \cap \bar{S}|}{|\bar{S}|} \geq \beta) &= Pr(\frac{|S_\alpha \cap \bar{S}|}{|\bar{S}|} \geq \frac{|\underline{S}|}{|\bar{S}|}) && (1)\\
&= Pr(|S_\alpha \cap \bar{S}| \geq |\underline{S}|) && (2)\\
&\geq Pr(|S_\alpha \cap \underline{S}| \geq |\underline{S}|) && (3)\\
&= Pr(|S_\alpha \cap \underline{S}| = |\underline{S}|) && (4)\\
&= Pr(\frac{|S_\alpha \cap \underline{S}|}{|\underline{S}|} = 1) \geq 1 - \delta && (5)
\end{aligned}$$

The last inequality we get from Theorem 1.2. This proof shows that if β is close to one, the precision of $\bar{S}$ will also be close to one with high probability.

3.5 Discovering frequent cohesive sets

The sampling approach to discovering highly reliable subgraphs gives rise to a new graph mining problem that Jin et al. call the *Frequent Cohesive Set* (FCS) problem. Jin et al. define the problem as follows:

DEFINITION 3 (**Frequent Cohesive Set Problem**) [1]: *For a graph G and a subset of vertices $V_s \subseteq V[G]$, if its induced subgraph $G[V_s]$ is a connected component, then V_s is called a cohesive set. If we have a set of graphs $D = G_1, ..., G_n$ containing the same vertices: $V(G_1) = ... = V(G_N) = V$, then we define a frequent cohesive set (FCS) as any subset of vertices $V_s \subseteq V$ that is cohesive in at least $N \cdot \theta$ graphs, where θ defines the minimum support threshold.*

Note that the definition of a frequent cohesive set corresponds to the definition of a highly reliable subgraph in Definition 2, when using the reliability estimate. Thus, Definition 3 bridges the problem of finding highly reliable subgraphs with the sampling approach.
From Definition 3, we define ***maximal frequent cohesive sets(MFCS)*** to be the set of all FCS in D, where none of their supersets are frequent [1].

In *step 1* of the aforementioned sampling algorithm (section 3.3), N_1 samples are used to generate the approximate set $\bar{S}$. This set corresponds to discovering all FCS with minimum support $\theta = \alpha - \epsilon$ [1]. After having discovered $\bar{S}$, in *step 2* the set N_2 is sampled and used to compute the reliabilities of all FCS in $\bar{S}$.

3.6 Mining frequent cohesive sets

The problem of discovering frequent cohesive sets is similar to finding reliable subgraphs, as they both lack the anti-monotonicity property [1]. To handle this problem, Jin et al. use a mining algorithm. The algorithm is an iterative top-down peeling process, where patterns are refined to make them converge into a set of MFCS. Jin et al. refer to this algorithm as the "peeling algorithm" as it produces subsets from supersets during the refinement (peeling) in the pattern discovery [1].

3.6.1 Relaxation approach

To gain the initial supersets from which the MFCS are to be peeled, Jin et al. introduce a relaxation approach, where the cohesive condition on a vertex set is relaxed by allowing connectivity through vertices outside the set. Jin et al. use the relaxed approach to introduce the maximal frequent linked set problem [1]. The relaxed problem satisfies the *anti-monotonicity* property, which makes it easier to devise an efficient pruning algorithm:

DEFINITION 4 (**Maximal Frequent Linked set Problem** [1]): *A subset of vertices $V_s \subseteq V[G]$ in graph G is said to be a **Linked set** if it belongs to a connected component in G. Given a graph database $D = G_1, G_2, ..., G_N$ with $V(G_1) = V(G_2) = ... = V(G_N) = V$ and a minimum support threshold θ, a subset of vertices $V_s \subseteq V$ is referred to as a frequent linked set, if it is linked in at least $N \cdot \theta$.*

The frequency of V_s being linked connected is denoted as $\hat{R}_L[V_s]$, and the maximal frequent linked set problem tries to identify all the linked sets in D, such that no superset of these sets are frequent linked. It is clear that any frequent cohesive set must also be a frequent linked set. However, is not necessarily true that a frequent linked set is also frequent cohesive. This can be formalized as follows: *If any V_s is a maximal frequent cohesive set, then there must be a maximal frequent linked set V_s' such that $V_s \subseteq V_s'$ [1].* Thus the set of maximal frequent linked sets (MFLS) can serve as an initial pattern set and be used as an upper bound for all maximal frequent cohesive sets (MFCS). Discovering all MFLS can be reduced to the well-studied problem of finding all maximal frequent itemsets (MFI) using a transactional database T of connected components. [1]. Jin et al. formulate this as a Lemma:

LEMMA 4 [1]: *The set of all maximal frequent itemsets (MFI) in T with minimum support θ is equivalent to maximal frequent linked sets (MFLS) in D with the same minimum support level* [1].

There are several algorithms devised for mining MFI such as Apriori [25] or FP-Growth [26]. Furthermore, these algorithms can be adapted to both a top-down and bottom-up approach [27]. Jin et al. do not specify which algorithm they use for mining MFI or whether they use a top-down or bottom-up approach. We choose to use the bottom-up approach since this makes sense for our top-k approach, where we no longer look for maximal frequent cohesive sets above a given threshold but instead look for the k most frequent cohesive sets. Furthermore, we have chosen to use Apriori [25] for mining MFI since it can easily be adapted to the iterative approach used in our novel top-k algorithm. The Apriori algorithm will be further elaborated in the implementation section.

The algorithm Jin et al. use for mining MFI takes a set of connected components as input. A connected component [28] in an undirected graph $\mathcal{G}$, is an induced subgraph $\mathcal{G}[V_s]$, in which any two vertices are connected to each other by edges, and not connected to any additional vertices in the graph. Connected components of a graph can be found in linear time by using a

depth-first search. The search, beginning in some an arbitrary vertex v, will find all vertices of the component that v is contained in and no other vertices [28]. To find all components, one has to loop over all vertices not already contained in a component.
To use an MFI algorithm, a transformation procedure, that converts our graph database D into a transactional database T, is required. This procedure can be described as follows:

DEFINITION 5 (**Transformation Procedure**) [1] *For each graph $G_i \in D$, find all its connected components denoted $G_i = C_{i1} \cup C_{i2} \cup ... \cup C_{ik}$. Output the vertex set of each connected component as an independent transaction, i.e., $T = T \cup \{V[C_{i1}] \cup V[C_{i2}] \cup ... \cup V[C_{ik}]\}$, where $T = \emptyset$ to begin with.* [1]

The Transformation process can be executed very fast since it has a linear computational complexity in regards to the size of the database D. This is because each vertex is only looked at once in the procedure of finding connected components in each graph G [1].
By Lemma 4, we can use an MFI algorithm on the transformed database to get the set of all MFLS.

3.6.2 Description and pseudocode for Peeling

The idea of the peeling algorithm is that for each initially discovered MFLS m in D, the linked sets are peeled such that they converge to FCS. If m is not a frequent cohesive set, the graph database is pruned, such that it only contains vertices in m. In other words, when recursively refining m, the algorithm has to work on the partial database $D[m] = \{G_1[m], G_2[m], ..., G_N[m]\}$ that only includes the induced subgraphs of m for each possible graph G_i in D [1]. After pruning D, such that it only contains vertices in m, m might no longer be connected, as it could be linked connected through vertices not in m. Additional MFLS are then discovered in the partial database $D[m]$. Hereby, this recursive process can be performed until all MFLS converge into FCS [1]. When a new FCS is discovered, all FCS that are contained by other FCS in the result set are pruned. This way, all FCS in result set will satisfy the maximality constraint.

Algorithm 1: Peeling [1]

```
   Result: MFCS
1  Parameter: D {G_1, G_2, ... G_N};
2  Parameter: MFLS {The intermediate maximal frequent linked sets};
3  Parameter: MFCS {The final maximal frequent cohesive sets};
4  foreach m ∈ MFLS do
5      if R̂[m|D[m]] ≥ θ {If m is an FCS} then
6          if ∄m' ∈ MFCS ∧ m' ⊇ m {If there is no superset of m in MFCS} then
7              MFCS = prune(MFCS ∪ {m}) {Add m to MFCS, and remove subsets of m};
8          end
9      else
10         T = Transform(D[m]) {Find connected components in in m};
11         MFLS' = MFI(T, θ) {Find MFLS in T};
12         Peeling(D, MFLS', MFCS)
13     end
14 end
```

Algorithm 1 shows the Peeling algorithm. The input to the algorithm is D, $MFLS$, and $MFCS$, where D is graphs sampled using Monte Carlo sampling, MFLS are the Maximal frequent linked sets obtained using an MFI algorithm, and MFCS is the running set of discovered maximal frequent cohesive sets. The loop in line 4 iterates over each MFLS m. In this loop, m

is added to the running set of MFCS if it is indeed frequent and maximal since there is no superset of m in MFCS (line 5-7). Furthermore, any FCS in the set MFCS is pruned if it is a subset of m, and thus not maximal (line 7). If m is not an FCS, a transactional database of connected components in m (line 10) is created and is used in the MFI algorithm to find MFLS (line 11). The algorithm can then, in a recursive manner, find potential MFCS in these new MFLS.

3.6.3 Complexity of the Peeling algorithm:

The MFI-algorithm is recursively invoked on smaller sets through the peeling process. Since all calls to MFI are on subsets of much smaller size. This is typically very fast for lower levels of the recursion [1]. Computing the reliability estimate $\hat{R}[m|D[m]]$ and transforming the graph database D to a transactional database T is done with a simple DFS scan of each induced subgraph $D[m]$. The recursive steps can be seen as a very small overhead compared to the complexity of the MFI algorithm [1]. The Peeling algorithm may redundantly examine the same immediate set of MFLS several times during execution. This is the case when we have two overlapping MFLS, which may cause the individual peeling processes to produce the same intermediate MFLS. The recursion in the peeling algorithm could then generate a combinatorial explosion, where most of the combinations could be redundant [1]. Jin et al. remove this redundancy in the FastPeeling algorithm.

3.7 FastPeeling

We will now describe the FastPeeling algorithm that removes redundant processing in the Peeling algorithm.

3.7.1 Layered Peeling

To speed up the Peeling algorithm Jin et al. use a Layered Peeling technique that reduces the number of redundant intermediate patterns. The layered peeling approach starts by peeling an initial set of MFLS in the first layer. This produces a set of intermediate frequent linked sets in the second layer. The peeling process continues to produce new intermediate layers. If a frequent linked set is also frequent cohesive, it is removed from the next layer [1]. Each layer only consists of maximal patterns since it is sufficient to work with maximal linked sets without losing information [1]. This follows from the property:

LEMMA 5 [1]: *Given two frequent linked sets m and m', if $m \subseteq m'$ then the MFCS in m must also be contained in the MFCS of m'. Furthermore, if a pattern T is an FCS, subsets of T can safely be pruned in new layers.*

This property allows all unique intermediate patterns m to be peeled only once [1].

We provide an example of how the layered approach works. The example is based on the graph shown in Figure 1. In this example, we need three layers to discover all MFCS in $\mathcal{G}$. The first layer includes the initially discovered MFLS in $\mathcal{G}$: $\{a, b, e, f, g\}$ and $\{a, b, d, f, g\}$. Then in the second layer we have two MFCS: $\{b, e, f, g\}$, $\{a, e, f, g\}$ and two intermediate MFLS patterns: $\{a, b, e, f\}$, $\{a, b, d, g\}$. Lastly, in the third layer, we find an MFLS $\{b, e, f\}$. Since we already know that $\{b, e, f, g\}$ is a MFCS, we can prune $\{b, e, f\}$. Consequently, the third layer becomes empty, and no further search is needed.

3.7.2 Transaction reduction

The Layered Peeling approach ensures that patterns in a given layer do not contain each other [1]. However, it does not ensure that no non-maximal patterns are generated in lower levels. That is, patterns created later in a lower level may be subsets of those produced in earlier layers. For further computation speedups, Jin et al. prevent non-maximal patterns from being generated by using the following Lemma:

LEMMA 6 [1]: *Assume that patterns in a layer are visited sequentially. Let $P \subseteq L$ include the patterns that have already been peeled. Let $V_{i,j}$ denote the vertex set of a connected component $C_{i,j}$ in $G_i[m]$, where $m \in L$ has not been peeled. If there is another FLS $m' \in P$ that has been peeled and $V_{i,j} \subseteq m'$ then it is safe to drop $V_{i,j}$ in the transactional database that has been transformed from $D[m]$ without losing any potential $FCSs$ in D.*

Lemma 6 can also be used for any discovered $(M)FCS$. If any transaction is a subset of an MFCS in the database, it can safely be pruned [1]. The argument behind this is that such a transaction can not contribute to any new MFCS. They can only generate sets that are subsets of those that have already been visited. By dropping these patterns, it is possible to prevent the generation of unnecessary patterns. This will result in a smaller database of transactions to process, which will help reduce computation costs.

3.7.3 Description and pseudocode for FastPeeling

That algorithm takes only the graph database D as input. The algorithm proceeds to find all MFLS using the MFI algorithm and setting the initial L to the set of MFLS (line 3). Furthermore, the intermediate layer L' and the set $MFCS$ are both initialized to the empty set (line 4). The algorithm then works iteratively by visiting each m in L in decreasing order of pattern size. This order maximizes the efficiency of the pruning search space (Lemma 6). For each m in L, the algorithm checks if m is an FCS by computing the reliability estimate. If m is an FCS, it is only added to the set $MFCS$, if there is no superset of m in the set. Lastly, subsets of m in $MFCS$ are pruned in order to maintain the maximal constraint (line 7-10). Alternatively, if m is not an FCS, a new transactional database is created (line 12), using m and the transaction reduction method. A new set $MFLS'$ is then created, using the MFI algorithm (line 13), and lastly, L' is pruned, meaning that subsets are removed (line 14 and Lemma 5). Before the next iteration, a new layer of FLS is created by removing all FLS contained in already discovered MFCS (line 18). The algorithm terminates when a newly produced layer is empty.

Algorithm 2: FastPeeling [1]

Result: MFCS
Parameter: D $\{G_1, ..., G_N\}$
$L = MFI(transform[D], \theta)$;
$L' = \emptyset$; $MFCS = \emptyset$;
while $L \neq \emptyset$ **do**
  $P = \emptyset$ {*Already peeled sets in L* } ;
  foreach $m \in L$ {*Decreasing order of pattern size*} **do**
    if $\hat{R}[m|D[m]] \geq \theta$ { *if m is a FCS* } **then**
      if $\nexists m' \in MFCS \wedge m \subseteq m'$ **then**
        $MFCS = prune(MFCS \cup m)$;
      end
    else
      $T = TransReduce(D[m], P \cup MFCS)$ { *Lemma 6*};
      $MFLS' = MFI(T, \theta)$ {*Get maximal frequent itemsets*} ;
      $L' = prune(L' \cup MFLS')$ {*Lemma 5*};
    end
    $P = P \cup m$
  end
  $L = prune(L' \cup MFCS) \setminus MFCS$ {*Lemma 5*} ;
  $L' = \emptyset$;
end

3.7.4 Computational cost of FastPeeling

The computational cost of the FastPeeling algorithm is dominated by the Maximal Frequent Itemset (MFI) algorithm. More specifically, the MFI mining for the entire graph database, as the computational cost of the MFI mining in subsequent layers is insignificant compared to the computational cost of initial MFI mining [1]. Jin et al. show by experiments that the overall computational time of FastPeeling is no higher than two times that of initial MFI mining. This is in accordance with our own intermediary experiments of the FastPeeling algorithm. However, these experiments are not included in the experiment section, as we will focus on experiments for the novel top-k algorithm.

3.8 Mining Non-Maximal

Peeling and FastPeeling provide a way to discover maximal frequent cohesive sets. It might, however also be beneficial to discover all non-maximal frequent cohesive sets. A naive approach would be to check the reliability estimates of all subsets of the MFCS found by either peeling algorithm. If FastPeeling for example found {a,b,c,d,e}, one would have to check {a,b,c,d}, {a,b,c,e},{a,b,d,e} and so on. This will however be very slow, as a DFS on each sample for each subset is required. Instead, Jin et al. have proposed a faster alternative, which utilizes techniques for fast subset checking and fast connectivity testing [1].

3.8.1 Fast subset checking

Let each MFCS have a unique ID, and associate each vertex in the uncertain graph with a list of IDs for each MFCS containing the given vertex. Using these IDs, it is possible to keep a running

list of MFCS containing a given vertex set. More concretely, when a vertex is added to the vertex set, the list of IDs will be the intersection of the current list of IDs and the list associated with the added vertex [1]. If the list is empty, the vertex set is not a subset of any MFCS.

3.8.2 Fast connectivity test

The naive way of checking if a given vertex set V_s is a frequent cohesive set is, as described earlier in this section, to perform a DFS on each sample to check if V_s is induced in the given sample. Jin et al. improve this strategy by making the following observation: *For a connected induced subgraph, if a new vertex is added, and it is connected to at least one vertex in the subgraph, then the new subgraph is also connected [1].* To utilize this observation Jin et al. use a binary vector to record whether a vertex set is connected in each sample. With the binary vector, it is easy to determine if its expansion by one is connected in a sample by checking the corresponding entry in the binary vector and if the newly added vertex is connected to the vertex set. This way, it is only necessary to traverse the sample, when the vertex set is not connected, but might instead be connected through the newly added vertex [1]. The binary vector can also be used to quickly get the reliability of the V_s, as it will be a normalized aggregate of the vector.

3.8.3 Description and pseudocode for Mining non-maximal

Algorithm 3 describes a DFS approach to mining non-maximal frequent cohesive sets [1]. The idea of the algorithm is to perform a DFS on the uncertain graph $\mathcal{G}$ to discover any connected vertex set that is a subset of at least one MFCS previously discovered (using fast subset checking), and has a reliability estimate above the threshold (using fast connectivity test).

The algorithm takes as input v, V_s, ID, Con, N, Ex where v is the newly added vertex, V_s is the current vertex set in the uncertain graph $\mathcal{G}$, ID is the list of IDs of MFCS that contain the current vertex set, Con is the binary vector of connectivity, N is the set of all neighbors of V_s, and Ex is the exclusion list of vertices which we have already visited. Whenever a new vertex is added to the vertex set, the algorithm checks if it is an FCS by iterating over all samples while updating the connectivity vector and using this updated connectivity vector to estimate the reliability of the vertex set (line 7 to 16). In line 17 to 22 each neighbor w to the vertex set, not in the exclusion list, is visited. More precisely, we find the IDs of MFCS containing $V_s \cup \{w\}$, and add w to the exclusion list. The vertex w is then only visited in the DFS if more than one MFCS contains $V_s \cup \{w\}$ or one MFCS contains $V_s \cup \{w\}$, and it is not equal to the MFCS. This way, we know that we have not yet exhausted all possible FCS yet [1].

Algorithm 3: MiningNonMaximal [1]

Result: MFCS
1 **Parameter:** v {*Newly added edge-vertex in* V_s}
2 **Parameter:** V_s {*The current vertex set*}
3 **Parameter:** ID {*The IDs of MFCS containing* V_s}
4 **Parameter:** Con {*The binary vector for connectivity*}
5 **Parameter:** N {*The neighbors of* V_s}
6 **Parameter:** Ex {*The exclusion list of vertices that already have been explored*}
7 **foreach** $G_i \in D$ **do**
8 | **if** $Con[i]$ {*If* V_s *is connected in* G_i} **then**
9 | | $Con'[i] = (Neighbor(v|G_i) \cap V_s) \neq \emptyset$ {*Check if* v *has neighbors from* V_s *in* G_i} ;
10 | **else**
11 | | $Con'[i] = Connected(G_i[V_s])$ {*Is the set of vertices* V_s *connected in* G_i} ;
12 | **end**
13 **end**
14 **if** $\hat{R}[V_s] \geq \theta$ {*Compute reliability estimate using* Con' } **then**
15 | $FCS = FCS \cup V_s$ { V_s *is non-maximal frequent cohesive set* };
16 **end**
17 **foreach** $w \in N \setminus Ex$ **do**
18 | $ID' = ID \cap ID[w]$ {*Find MFCS containing* V_s};
19 | $Ex = Ex \cup \{w\}$;
20 | **if** $|ID'| > 1 \vee (|ID'| = 1 \wedge V_s \cup \{w\} \neq MFCS \text{ in } ID'$ **then**
21 | | MiningNonMaximal($w, V_s \cup \{w\}, ID', Con', Neighbor(w|\mathcal{G}) \cup N, Ex$)
22 | **end**
23 **end**
24 **Procedure Main:**
25 $Ex = \emptyset$;
26 **foreach** $w \in V \setminus Ex$ **do**
27 | $Ex = Ex \cup \{w\}$;
28 | MiningNonMaximal($w, \{w\}, ID[w], \mathbf{0}, Neighbor(w|\mathcal{G}), Ex$) ;
29 **end**

Note that in line 20 and 21 of the pseudocode provided by Jin et al., they add the newly added vertex v, and not w. We consider this an error, as v is already added to the vertex set in the initial call. Furthermore, adding v instead of w will result in errors in the connectivity vector, as line 9 will be ahead of line 11. It is also clear, that it is an error, as in the $then$ case (line 9), the values of the entries in the connectivity vector are based on whether v is a neighbor to the vertex set or not. This does not make sense since v is not a part of the vertex set and hence should not influence if the vertex set is an FCS or not.

3.8.4 Computational Cost of Mining Non-Maximal

The worst case computational cost of the mining non-maximal algorithm is determined by the number of connected vertex sets. This is bounded by the MFCS [1]. For each connected vertex set, a DFS on each sample is required. Each DFS has linear cost.

4 Apriori

Jin et al. do not specify which specific MFI algorithm they use to obtain the maximal frequent linked sets. We have decided to use the Apriori algorithm in conjunction with a pruning algorithm in order to obtain the maximal frequent itemsets.

The Apriori algorithm is a bottom-up approach to finding frequent itemsets. It uses the Anti-monotonicty property stating, that all supersets of infrequent itemsets must be infrequent.

4.1 Description and pseudo code for Apriori

Algorithm 4: Apriori

Result: MFI
Parameter: θ {*Support-level*}
Parameter: T {*Transactional database of connected components*}
$C_1 = vertices \in T$;
foreach $c \in C_1$ {*Count occurences of each item*} **do**
  foreach $t \in T$ **do**
    if $c \subseteq t$ **then**
      $c.count++$;
    end
  end
end
$L_1 = \{c \in C_1 | \frac{c.count}{|Samples|} \geq \theta\}$ {*Add all itemsets with support above* θ};
for $z = 2; L_{z-1} \neq \emptyset; z++$ {*Continue until previous layer is empty*} **do**
  $C_z = join(L_{z-1})$;
  foreach $c \in C_z$ **do**
    foreach $t \in T$ **do**
      if $c \subseteq t$ **then**
        $c.count++$;
      end
    end
  end
  $L_z = \{c \in C_z | \frac{c.count}{|Samples|} \geq \theta\}$ {*Add all itemsets with support above* θ};
end
$MFI = Prune(\bigcup_z L_z)$ {*Remove all contained itemsets and itemsets of size 1 and 2*} ;

The input to the Apriori algorithm is a transactional database, of connected components along with the support level, which is equivalent to the threshold used by the peeling algorithm itself. In order to obtain the transactional database, we use the transformation procedure from Definition 5. In the generation of the transactional database, we also exclude connected components of size one and two, as these will never contribute to the frequency of subsets of size three and above. Furthermore, as we have the anti-monotonicity property, we know that subsets of frequent itemsets are also frequent, meaning that we can classify the subsets of size one and two as frequent, as long as they have a frequent superset, hence no relevant information is lost in this exclusion.

Algorithm 4 describes the bottom-up approach to Apriori [25]. It works by identifying fre-

quent individual items in the transactional database (line 3-11). The idea is then, to join the frequent itemsets in the previous iterations (line 13), and calculate the frequency of those larger itemsets, by iterating over the transactional database and identifying the joined itemsets (line 14-20). In each iteration, the algorithm prunes away itemsets, that do not have a support above the given support level (line 21). Here, the support for a given itemset is the count normalized by the sample size. We normalize using the sample size, and not the size of the transactional database, as the support of an itemset is then directly translatable to the linked reliability $\hat{R}_L[V_s]$ of the subgraph V_s of the given itemset. This makes sense, as each vertex only can appear once per sample, while the number of components per sample can vary a lot. Lastly, we are only interested in the maximal frequent itemsets, and we can thus prune all frequent subsets of frequent supersets (line 23). Furthermore, we also prune itemsets of sizes one and two. This is because singleton sets will have a reliability estimate of 1, and all sets containing only two vertices will have an expected reliability estimate equal to the probability of the edge connecting them and is hence not of much interest.

Figure 2 shows an execution of the Apriori algorithm. Note that the pruning step, in line 23 of the Apriori algorithm, is not displayed in the figure.

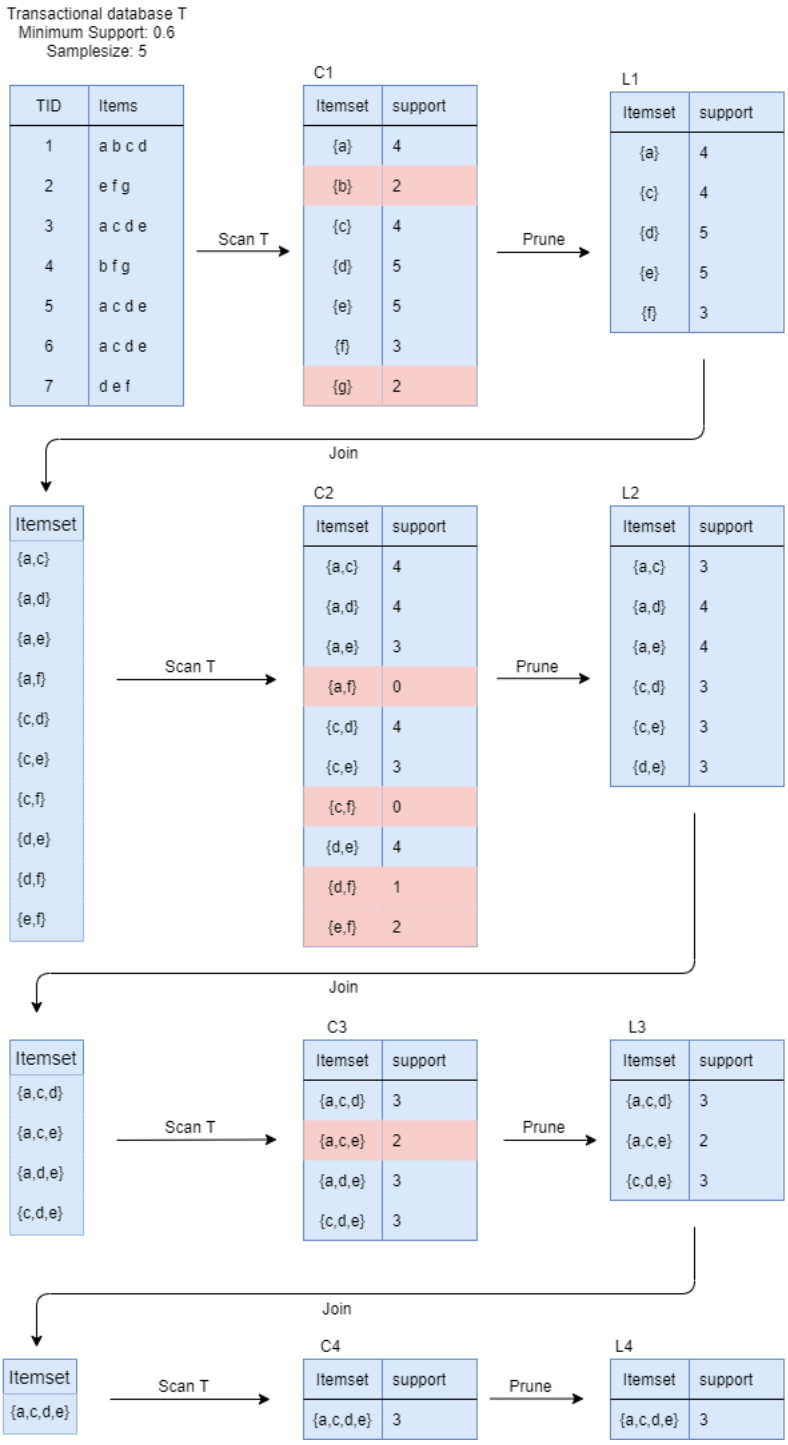


Figure 2: Example of an execution of the Apriori algorithm. Itemsets marked with red are the sets that are pruned.

4.2 Computational cost of Apriori

The computational costs of Peeling and FastPeeling are dominated by the Apriori algorithm used to find the maximal frequent linked sets. The worst-case computational cost of Apriori is $\mathcal{O}(2^{|\mathcal{G}[V]|} \cdot |T| \cdot |\mathcal{G}[V]|)$ as it potentially has to make all combinations of the vertices in the uncertain graph and for each combination traverse the transactional database, where the size of each transaction is bounded by the number of vertices in the uncertain graph.

5 Devising novel solutions to finding k most reliable subgraphs

The goal of the paper by Jin et al. [1] is to find all maximal frequent cohesive sets for a given threshold θ. Instead, we would like to solve the problem of finding the k most reliable subgraphs:

DEFINITION 6 (k **Most Reliable Subgraph Problem**): *Given an uncertain graph $\mathcal{G} = (V, E, P)$ and an integer k, determine the k most reliable subgraphs of at least three vertices.*

In other words, determine all subgraphs m_i such that $R[m_i] \geq R[m_k]$, where m_k is the k'th most reliable subgraph. We add the condition that the subgraphs must include at least three vertices since a single vertex has reliability of 1. For two vertices, the reliability is just the edge with the highest reliability. The problem described in Definition 6 is #P-complete since the Highly Reliable Subgraph Problem from Definition 2 can be reduced to the k most reliable subgraph problem in Definition 6. The proof for this reduction works as follows:

Proof of reduction from Highly Reliable Subgraph Problem to the k Most Reliable Subgraph Problem:
Assume we have an algorithm $X(\mathcal{G}, k)$ that finds exactly the k most reliable subgraphs. We can then use X to find all subgraphs $\mathcal{G}[V_s] \in \mathcal{G}$, where $R[\mathcal{G}[V_s]] \geq \theta$ as follows: We use increasing values of k until we find a subgraph $\mathcal{G}[V_s']$ where $R[\mathcal{G}[V_s'] < \theta]$ for some $k = k'$. The answer is then the top-k subgraphs with $k = k' - 1$. As we have an exponential amount of combinations, we do logarithmic steps in k, and then backtrack with binary search on the interval between two steps in order to have a polynomial reduction. If $X(\mathcal{G}, k)$ is a polynomial-time algorithm, then the whole process is polynomial time. This is a contradiction, as we know that the HRS problem is #P-complete. The k most Reliable Subgraph Problem must then also be #P-complete.

To address the issues related to solving a #P-complete problem, we use an approximation scheme similar to the one used by Jin et al **[?]**, described in section 3.3. This gives way to a new problem of finding the k most frequent cohesive sets, which we denote kFCS, where cohesive sets are defined as in Definition 3:

DEFINITION 7: (k **Most Frequent Cohesive Set Problem**) *Given a set of graphs $D = \{G_1, G_2, ...G_N\}$ with vertices $V(G_1) = V(G_2) = ... = V(G_N)$, and a number k, determine the k most frequent cohesive sets.*

When dealing with top-k problems, the notion of precision and recall are mutually inclusive. If we have a false-positive in the result, it is implied that there is also a false-negative that is not included in the result. Instead, we reason about the precision of the algorithm by using

$Precision@k$, which is defined as follows:

$$Precision@k = \frac{|\mathcal{K} \cap \hat{\mathcal{K}}|}{k}$$

where $\mathcal{K}$ is the set containing the true k most frequent cohesive sets, and $\hat{\mathcal{K}}$ is the set containing k cohesive sets returned by the algorithm.

We will now present different approaches to solving the problem in Definition 7.

5.1 Eager approach

The approach in Jin et al. [1] consists of a long preprocessing step that runs an MFI algorithm to find all candidate sets that are MFLS, followed by running the peeling algorithm on these candidates. The basic idea of the novel approach is to merge the MFI algorithm, specifically the Apriori algorithm, and the peeling part such that it is possible to use a dynamic threshold for both MFI and the peeling part. For this purpose, we have created an Iterative Apriori algorithm.

In the EagerPeeling algorithm, the Iterative Apriori algorithm is used to find one frequent linked set(FLS) m at a time. In EagerPeeling, the reliability estimate of m is calculated, and if it is above the current threshold, m is added to the set kFCS containing the subgraphs estimated to be the k most reliable. The threshold used in both the Iterative Apriori and EagerPeeling algorithms are then updated to the new top-k reliability threshold. This means that we potentially increase the reliability threshold once for each candidate set. Hereby, the Iterative Apriori algorithm avoids generating many redundant candidate sets, as itemsets classified as infrequent itemsets will never be joined. By using a bottom-up version of the Apriori algorithm, we are guaranteed not to prune away FLS that can be peeled to possible top-k candidates.

To find the k most reliable subgraphs, the EagerPeeling algorithm starts with a reliability threshold of 0 as we do not have a lower bound on the probabilities in the unreliable graph.

5.1.1 Description and pseudocode for Iterative Apriori

Algorithm 5: GetNextCandidate

```
1  Field: L {Previous frequent candidates}
2  Field: z {Used to look at z sized sets in L}
3  Field: numSamples
4  Field: T {transactional database of connected components}
5  Parameter: θ_curr
6  if No more candidates to be joined in L_z then
7      z++
8  end
9  c = Join new candidate of size z+1 from L_z;
10 foreach t ∈ T do
11     if c ⊆ t then
12         c.count++ ;
13     end
14 end
15 if c.count / numSamples ≥ θ_curr then
16     L_{z+1} = L_{z+1} ∪ c;
17     return c
18 else
19     return GetNextCandidate()
20 end
```

Algorithm 5 describes the GetNextCandidate method which is a part of our iterative version of the Apriori algorithm. Instead of returning all candidate sets, it returns a single candidate set, i.e an FLS. This allows the support threshold to be updated before generating more candidate sets. As the algorithm is invoked several times, we need an internal state in order for the Iterative Apriori to be congruent with the EagerPeeling algorithm. This includes keeping a state of which itemsets were previously in the candidate set (L in line 1), which subset sizes are we currently looking at (z in line 2), the number of samples we are working with (numSamples in line 3) and the transactional database of connected components (line 4). Additionally, each time the GetNextCandidate algorithm is called it takes the current threshold value as a parameter (θ_{curr} in line 5).
GetNextCandidate joins two sets of size z into a set of size $z + 1$ (line 9). Once the internal state has exhausted the combinations of sets of size z it increments z and continues with sets of size $z + 1$ (line 6-8). The algorithm then iterates over all transactions to find the support of the candidate (line 10-14). If the frequency of the candidate is higher than the threshold, it is added to L_{z+1}, (line 15-17) such that L_{z+2} sets can be joined from it later on. The candidate is then returned. Otherwise, if the candidate is not frequent, GetNextCandidate is called again combining, two new sets from L_z (line 18-20). In summary, GetNextCandidate returns the next itemset of size $z + 1$ that is above the current threshold and maintains its state for the next iteration. Note that the algorithm is not invoked in EagerPeeling without verifying that more candidates are available.

5.1.2 Description and pseudocode for EagerPeeling

Algorithm 6 describes the eager approach to finding the k most frequent cohesive sets. The method HasNext and GetNextCandidate are from the Iterative Apriori, where HasNext (line 10) tells whether more FLS exists.

Algorithm 6: EagerPeeling

```
   Result: kFCS
1  Parameter: k {The number of subgraphs to be returned};
2  Parameter: D {The sampled possible graphs};
3  Parameter: T {Transactional database of connected components};
4  kFCS = ∅;
5  θ = 0;
6  while |kFCS| < k {Fill kFCS with arbitrary candidates} do
7  |  kFCS = kFCS ∪ nextCandidate(T, θ);
8  end
9  θ = lowest reliability in kFCS;
10 while hasNext(θ) do
11 |  m = getNextCandidate(θ);
12 |  if R̂[m|D[m]] > θ {If m is a top-k reliable subgraph} then
13 |  |  m_lowest = {m' ∈ kFCS | R̂[m'|D[m']] = θ} {get FCS with lowest reliability estimate};
14 |  |  kFCS = (kFCS ∪ m) \ m_lowest;
15 |  |  θ = lowest reliability in kFCS ;
16 |  end
17 end
18 Procedure Main:
19 numSamples = calculateSampleSize {From Theorem 2 or 3 in sections 5.1.5 and 5.1.6};
20 D = Sample(G, numSamples);
21 T = GetConnectedComponents(D) ;
22 kFCS = EagerPeeling(k, D, T)
```

EagerPeeling takes as parameters the number of cohesive sets (k) to return, the samples, and the connected components which are used by the Iterative Apriori to get the frequent linked sets.

First, we calculate the sample size that provides approximation guarantees as a function of ϵ and δ in accordance with Theorem 2 in section 5.1.5 or Theorem 3 in section 5.1.6 (line 19). After having sampled a number of graphs(line 20), we find all connected components in each sample (line 21). Subsequently, the EagerPeeling algorithm is called with the sampled graphs, the connected components, and value k. In EagerPeeling, kFCS is initialized to the empty set, and the threshold is initialized to 0 (line 4 and 5). The kFCS set is then filled with k cohesive sets, and the threshold is updated to the k'th highest reliability estimate (line 6-9). We then loop over all candidates (line 10 - 17). In each iteration, the reliability estimate is computed for the cohesive set m (line 12). If the reliability estimate is above the current threshold (line 13) m replaces the cohesive set with the lowest reliability estimate (line 14), and the threshold is updated to the new lowest reliability estimate in kFCS (line 15).

5.1.3 Computational cost of EagerPeeling

As in the FastPeeling algorithm, devised in Jin et al., the runtime of EagerPeeling is bounded by the MFI algorithm, which in our case is Apriori. The runtime of Apriori is described in section 4. In addition to Apriori, EagerPeeling performs a DFS search on each sample for each candidate returned by the iterative Apriori, which in the worst case is $2^{|\mathcal{G}[V]|}$ candidates. This means, that the computational cost of EagerPeeling is: $\mathcal{O}(2^{|\mathcal{G}[V]|} \cdot |T| \cdot |\mathcal{G}[V]| + 2^{|\mathcal{G}[V]|} \cdot (|\mathcal{G}[V]| + |\mathcal{G}(E)|))$. However, as the depth-first searches are bounded by the computational cost of Apriori, the computational cost of EagerPeeling is $\mathcal{O}(2^{|\mathcal{G}[V]|} \cdot |T| \cdot |\mathcal{G}[V]|)$

5.1.4 Correctness of EagerPeeling

From Jin et al. we have that for a vertex set V_s, the linked reliability estimate is greater than or equal to the cohesive reliability estimate: $\hat{R}_L[V_s] \geq \hat{R}[V_s]$. This allows us to dynamically increase the threshold used in the Iterative Apriori algorithm, to the k'th highest cohesive reliability estimate $\hat{R}[V_s]$, without missing any cohesive sets with a potentially higher reliability estimate. This is true, as subsets with a linked reliability estimate below the threshold can never have a cohesive reliability estimate above the threshold and are thus irrelevant.

5.1.5 Statistical guarantees for EagerPeeling

The statistical guarantees for the novel algorithms that can be proven are not the same as in [1], because the notion of precision and recall in a top-k problem is mutually inclusive. Instead, we define approximate top-k results as in the work by Pietracaprina et al. [20]. In the following definition let S be the result set of k subgraphs and m_k be the subgraph with the true k'th highest reliability.

Eager result properties: With a sample size of $N \geq \frac{1}{2\epsilon^2}((n+1)ln(2) - ln(\delta))$ we get the following three properties with probability $1 - \delta$:

1. $\forall m : |\hat{R}[m] - R[m]| \leq \epsilon$
2. $m \in S \implies R[m] \geq R[m_k] - 2\epsilon$
3. $m \notin S \implies R[m] \leq R[m_k] + 2\epsilon$

The problem of finding top-k results is inherently harder than finding all subgraphs above a certain threshold. When we do not have a fixed threshold, any subgraph reliability $R[m_i]$ that is estimated poorly, such that $\hat{R}[m_i] > R[m_i] + \epsilon$, can potentially raise the dynamic threshold too high. This can result in actual top-k reliable subgraphs not being included in the result. We must hence require that no candidate subsets are estimated outside their ϵ-range.
Proof of property 1: To satisfy criteria 1 with some probability $1-\delta$, we must find the minimum required sample size according to the Hoeffding bound [23, 24]. Let A_i be the event that the i'th subgraph is estimated outside its ϵ-range. This can be upper bounded by the Hoeffding inequality, with q being the number of samples:

$$P(A_i) = Pr(|R[m_i] - \hat{R}[m_i]| \geq \epsilon) \leq 2e^{-2q\epsilon^2}$$

No subgraph can be estimated outside it's ϵ-range except with some probability δ. The probability that any of the reliabilities are estimated wrongly can be upperbounded by union bound

over all 2^n possible vertex sets:

$$Pr(\bigcup_{i=1}^{2^n} A_i) \leq \sum_{i=1}^{2^n} Pr(A_i) \leq 2^n 2e^{-2q\epsilon^2}$$

To make sure that no subgraph is estimated wrongly, except with probability δ, any single estimation can at most fail with probability:

$$Pr(A_i) \leq 2e^{-2qt^2} \leq \frac{\delta}{2^n}$$

The required sample size to ensure this is found by isolating q:

$$\frac{\delta}{2^n} = 2e^{-2q\epsilon^2} \tag{1}$$

$$\frac{\delta}{2^{n+1}} = e^{-2q\epsilon^2} \tag{2}$$

$$\frac{2^{n+1}}{\delta} = e^{2q\epsilon^2} \tag{3}$$

$$ln\left(\frac{2^{n+1}}{\delta}\right) = 2q\epsilon^2 \tag{4}$$

$$\frac{1}{2\epsilon^2} ln\left(\frac{2^{n+1}}{\delta}\right) \leq q \tag{5}$$

$$\frac{1}{2\epsilon^2}(ln(2^{n+1}) - ln(\delta)) \leq q \tag{6}$$

$$\frac{1}{2\epsilon^2}((n+1)ln(2) - ln(\delta)) \leq q \tag{7}$$

The required sample size is thus linearly dependent on the input size and quadratically dependent on ϵ. In section 5.2 another algorithm that avoids large sample sizes when searching for accurate results is proposed. If the sample size is greater than $\frac{1}{2\epsilon^2}((n+1)ln(2) - ln(\delta))$ condition 1 should hold with probability $1 - \delta$.

Proof of property 2: Property 2 follows from condition 1 as $R[m_k]$ can at most be underestimated by ϵ and $R[m]$ can at most be overestimated by ϵ. In EagerPeeling, subgraphs with reliability estimates higher than the k'th estimate are included which gives the following:

$$\hat{R}[m] \geq \hat{R}[m_{\hat{k}}] \implies \hat{R}[m] \geq R[m_k] - \epsilon \tag{1}$$

$$\implies R[m] + \epsilon \geq R[m_k] - \epsilon \tag{2}$$

$$\implies R[m] \geq R[m_k] - 2\epsilon \tag{3}$$

Implication 1 follows from the fact that any of the top-k subgraphs have a reliability above m_k and we know that none of these are estimated outside their ϵ-bound by condition 1. This means that at least k elements are estimated as $R[m_k] - \epsilon$ or higher. Implication 2 follows from the fact that $\hat{R}[m] \leq R[m] + \epsilon$.

Proof of property 3: Condition 3 follows from the fact that $R[m_k]$ can be overestimated by ϵ and $R[m]$ can be underestimated by ϵ:

$$\hat{R}[m] \leq \hat{R}[m_{\hat{k}}] \implies \hat{R}[m] \leq R[m_k] + \epsilon \tag{1}$$

$$\implies R[m] - \epsilon \leq R[m_k] + \epsilon \tag{2}$$

$$\implies R[m] \leq R[m_k] + 2\epsilon \tag{3}$$

Implication 1 holds because any subgraph not in top-k has a reliability less than or equal to $R[m_k]$.

As condition 2 and 3 always follow from condition 1, and we have proved that condition 1 holds with probability $1-\delta$, we have that with probability $1-\delta$ condition 2 and 3 hold.

5.1.6 EagerPeeling+

Pietracaprina et al. [20] have proved statistical guarantees in the context of finding top-k frequent itemsets. By using a similar approach, it is possible to derive a tighter statistical guarantee of ϵ instead of 2ϵ with a larger sample size that is dependent on k. We reason that it might be worth to introduce a dependency on k, if k is not too large since the sample size only has a logarithmic dependency on k. When using this larger sample size with EagerPeeling, we call it EagerPeeling+.

LEMMA 7 [20] *If $R[m_i] \geq R[m_j] + \epsilon$ we have:*

1. $Pr(\hat{R}[m_i] < \hat{R}[m_j]) \leq e^{-\frac{\epsilon^2 t}{2}}$
2. $Pr(|\hat{R}[m_i] - R[m_i]| \geq \epsilon) \leq 2e^{-\frac{\epsilon^2 t}{2}}$

Lemma 7 is used for proving the tighter bound. We include a proof for 7.1 which was omitted in the work by Pietracaprina et al.

Proof of 7.1 Given a sample of q possible graphs, we remind, that the reliability estimate of a subgraph $\mathcal{G}[V_s]$ is $\hat{R}[V_s] = \frac{\sum_{r=1}^{q} I(G_r[V_s])}{q}$, where $I(G_r(V_s))$ is the indicator function that evaluates to 1 if the vertex set V_s is connected in the possible graph r, and 0 otherwise. This means, that $I(V_s)$ is a Bernoulli random variable with a probability of success equal to $R[V_s]$. Thus, $\sum_{r=1}^{q} I(G_r[V_s])$ is then a Binomial random variable with parameters $Bin(q, R[V_s])$.

For any possible graph $G_r \sqsubseteq \mathcal{G}$ let $z_r = I(G_r[m_j]) - I(G_r[m_i])$. We then have the following equivalence:

$$\hat{R}[m_j] - \hat{R}[m_i] = \frac{\sum_{r=1}^{q} I(G_r[m_j])}{q} - \frac{\sum_{r=1}^{q} I(G_r[m_i])}{q} \quad (1)$$

$$= \frac{\sum_{r=1}^{q} (I(G_r[m_j]) - I(G_r[m_i]))}{q} \quad (2)$$

$$= \frac{\sum_{r=1}^{q} z_r}{q} \quad (3)$$

$$= \bar{z} \quad (4)$$

Each z_r is a random variable bounded in the interval [-1, 1], and has an expectation defined as:

$$E[z_r] = E[I(G_r[m_j]) - I(G_r[m_i])] \quad (1)$$

$$= E[I(G_r[m_j])] - E[I(G_r[m_i])] \quad (2)$$

$$= R[m_j] - R[m_i] \quad (3)$$

Likewise, we have that the expectation of $\bar{z}$ is defined as:

$$E[\bar{z}] = E[\frac{\sum_{r=1}^{q} z_r}{q}] \tag{1}$$
$$= \frac{\sum_{r=1}^{q} E[z_r]}{q} \tag{2}$$
$$= R[m_j] - R[m_i] \tag{3}$$

From Lemma 7.1 we have that $-E[\bar{z}] \geq \epsilon$, and $-E[\bar{z}] \geq 0$

$$R[m_i] \geq R[m_j] + \epsilon \iff R[m_i] - R[m_i] \geq R[m_j] - R[m_i] + \epsilon \tag{1}$$
$$\iff -\epsilon \geq E[\bar{z}] \tag{2}$$
$$\iff -E[\bar{z}] \geq \epsilon \tag{3}$$
$$\implies -E[\bar{z}] \geq 0 \tag{4}$$

By using the Hoeffding bound, we can do the following:

$$Pr(\hat{R}[m_i] < \hat{R}[m_j]) = Pr(\bar{z} > 0) \tag{1}$$
$$= Pr(\bar{z} - E[\bar{z}] > -E[\bar{z}]) \tag{2}$$
$$\leq e^{-\frac{2q\cdot(-E[z])^2}{4}} \tag{3}$$
$$\leq e^{-\frac{q\cdot\epsilon^2}{2}} \tag{4}$$

which concludes the proof.

At the cost of more samples it is possible to provide better statistical guarantees for the top-k results. These are described in Theorem 3:

THEOREM 3 (**Improved top-k result properties**): *With a sample size of* $\frac{2}{\epsilon^2} ln(\frac{2^{n+1}+K(2^n-K)}{\delta})$ *we get the following three properties with probability* $1-\delta$*:*

1. $\forall m : |\hat{R}[m] - R[m]| \leq \epsilon$
2. $m \in S \implies R[m] \geq R[m_k] - \epsilon$
3. $m \notin S \implies R[m] \leq R[m_k] + \epsilon$

To accomplish these properties we use the sample size derived by Pietracaprina et al.,:

$$q = \frac{2}{\epsilon^2} ln(\frac{2\omega + k(\omega - k)}{\delta})$$

The difference between the sample size used in EagerPeeling is that we introduce a dependency on k. By substituting ω with the number of possible subgraphs we get:

$$q = \frac{2}{\epsilon^2} ln(\frac{2\omega + k(\omega - k)}{\delta}) \tag{1}$$
$$= \frac{2}{\epsilon^2} ln(\frac{2(2^n) + k(2^n - k)}{\delta}) \tag{2}$$
$$= \frac{2}{\epsilon^2} ln(\frac{2^{n+1} + k(2^n - k)}{\delta}) \tag{3}$$

The following proofs of Theorem 3 use Lemma 7 and assume a canonical ordering on the possible candidate subgraphs depending on their actual reliability: $m_1, m_2, ...m_\omega$ st: $R[m_1] \geq R[m_2] ... \geq R[m_\omega]$. We use a similar approach in the proofs as Pietracaprina et al. [20].

Proof of property 2
We have rewritten the proof for property 2, derived by Pietracaprina et al., as a proof by contradiction:
Define the following property Q as follows: *For every* i, j *where* $1 \leq i \leq k < j \leq \omega$ *and* $R[m_i] \geq R[m_j] + \epsilon$, *we have* $\hat{R}[m_i] \geq \hat{R}[m_j]$. Assume that Q holds, then property 2 holds by the following contradiction:
Consider a subgraph $m \in S$ and assume by contradiction that $R[m] < R[m_k] - \epsilon$, which would imply by Q that $\hat{R}[m] < \hat{R}[m_k]$ which in turn would mean that $m \notin S$. This is a contradiction.

Proof of property 3 [20]
Assume again that fact Q holds, and consider a subgraph $m \notin S$. Suppose by contradiction that $R[m] \geq R[m_k] + \epsilon$, then $m = m_i$ for some i with $1 \leq i \leq k$. We know that S contains k pairs, so it must now contain a subgraph m_j with $j > k$ and $\hat{R}[m_j] \geq \hat{R}[m_i]$, which is impossible because $R[m_i] \geq R[m_k] + \epsilon \geq R[m_j] + \epsilon$.

Proof of Q and property 1 [20]
We show that if the sample size is chosen as stated in Theorem 3, Q and property 1 will hold with at least probability $1 - \delta$. From Lemma 7 we have that $Pr(\hat{R}[m_i] < \hat{R}[m_j]) \leq e^{-\frac{\epsilon^2 t}{2}}$ for any i, j where $1 \leq i \leq k < j \leq \omega$, and that $Pr(|\hat{R}[m_i] - R[m_i]| \geq \epsilon) \leq 2e^{\frac{\epsilon^2 q}{2}}$ for any $i \leq \omega$. $k(\omega - k)$ pairs are involved in Q, and at most ω itemsets are involved in property P3. By using union bound we get that the probability that either Q or P3 does not hold is at most $(2\omega + k(\omega - k))e^{-\frac{\epsilon^2 q}{2}} \leq \delta$, which the selection of q satisfies. As property 2 and 3 follows from Q, and Q holds with probability $1 - \delta$. Thus, all of the properties are satisfied with probability $1 - \delta$.

5.2 Attentive approach

While the above algorithm provides the guarantee that the reliabilities of the returned kFCS have a reliability of at least $R[m_k] - 2\epsilon$ (or $R[m_k] - \epsilon$ for EagerPeeling+) with probability $1 - \delta$, the algorithm is inefficient for very small ϵ-values, as the sample size has a quadratic dependency on ϵ. To retrieve the actual kFCS, a small ϵ-value might be necessary in order to make a distinction between two frequent cohesive sets with very similar reliability estimates. If estimates have overlapping ϵ-ranges it is impossible to determine which subgraph is more reliable. A small ϵ requires a large sample size, and thus the algorithm becomes almost infeasible to run, as Apriori is run on all samples. To accommodate a potential need for more precision, we introduce a two-step top-k algorithm that we call AttentivePeeling.

The first step of the algorithm is almost equivalent to EagerPeeling. In addition to the threshold, it keeps a relaxed threshold which is 2ϵ less than the actual threshold. Furthermore, it keeps a buffer in which subgraphs that satisfy the relaxed threshold are stored. The idea behind this is to filter out redundant candidates while maximizing recall. In the second step, we can then progressively increase the sample size and in turn, decrease ϵ until the difference between the k'th and k+1'th reliability estimates is greater than 2ϵ. By doing this, we can make a stronger guarantee stating: with probability $1 - \delta$, we obtain the true kFCS. The details behind this guarantee

are described in section 5.2.3.

5.2.1 Description and pseudocode for AttentivePeeling

Algorithm 7 describes the attentive approach to finding the k most reliable subgraphs.

Algorithm 7: ProgressiveSampling (AttentivePeeling)

```
   Result: kFCS
1  Parameter: kFCS {k most frequent cohesive sets}
2  Parameter: bufferFCS { Buffered cohesive sets found in step 1}
3  Parameter: ϵ
4  Parameter: ϵ_Limit
5  Parameter: samplesPerIteration
6  Parameter: numberOfSamples
7  candidates = kFCS ∪ bufferFCS ;
8  t = calculateT(δ, ϵ_limit, |candidates|, samplesPerIteration);
9  while |candidates| > k {While there are more than k candidates} do
10     newSamples = sample(G, samplesPerIteration) ;
11     updateReliabilities(candidates, newSamples, numberOfSamples) ;
12     numberOfSamples = newSamples + numberOfSamples;
13     ϵ = updateEpsilon(t, δ, numberOfSamples, |candidates|) ;
14     candidates = {c ∈ candidates | R̂[c] ≥ R̂[candidates[k]] − 2 · ϵ} ;
15     if ϵ < ϵ_Limit then
16        return Top-k candidates {Early stopping, if ϵ gets too low}
17     end
18 end
19 return candidates;
20 Procedure Main:
21 numberOfSamples = CalculateNumberOfSamples(ϵ, δ, G[V]) {By Lemma 8};
22 D = sample(G, numberOfSamples) ;
23 T = GetConnectedComponents(D);
24 eagerResult = EagerPeeling(D, k, T, ϵ);
25 attentiveResult = ProgressiveSampling(eagerResult.kFCS, eagerResult.bufferFCS, ϵ,
     ϵ_limit samplesPerIteration, numberOfSamples);
```

In the main procedure, the sample size is calculated, possible graphs are sampled, and the modified EagerPeeling is run. The output of the EagerPeeling algorithm is then used as input to the ProgressiveSampling step. As input, this algorithm takes $kFCS$ and $bufferFCS$ returned from the modified EagerPeeling, the current ϵ-value, a limit for the ϵ-value, samples pr iteration, and the number of samples used in EagerPeeling.
First, an upper bound on the number of iterations t is calculated (line 8), as this is necessary in order to provide the correct statistical guarantees. This is described in section 5.2.4. The algorithm works iteratively by looping as long as the number of candidates is strictly greater than k (line 9-18). In each iteration, new possible graphs are sampled (line 10), which are then used to update the reliability estimates of the candidates (line 11). This is done by finding the reliability estimate of each candidate $candidate_i$ in the new samples and then updating the reliability estimates of each candidate as

$$newReliability_i = \frac{newReliability_i|newSamples| + oldReliability_i|numberOfSamples|}{|newSamples| + |numberOfSamples|}.$$

The ϵ-value is updated (line 13), using the formula derived in section 5.2.4. After having updated ϵ, we filter out all candidates with a reliability estimates below the estimate of the k'th candidate subtracted by 2ϵ (line 14). The k'th and k+1'th candidates might have very similar reliabilities, to the point where they are identical. This will require indefinite sampling, and for this reason, an ϵ-limit is introduced (line 15-17), meaning that we will consider two candidates with reliabilities within ϵ-limit of each other to be identical, and then returns either of them.

The motivation for this algorithm is that, checking the connectivity for the already discovered candidates is cheaper compared to running Apriori on the larger sample size. This is because checking connectivity can be done in $\mathcal{O}(|E| + |V|)$ for each new sample. However, AttentivePeeling requires a buffer of cohesive sets satisfying the relaxed threshold, meaning that the running time of the modified EagerPeeling is slower than the EagerPeeling described in algorithm 6. Therefore, this algorithm should be used instead of the EagerPeeling algorithm only when a better guarantee is necessary.

It is important to note that that we sample in batches of size $samplesPerIteration$. This is to avoid memory problems, e.g. it might be infeasible to sample all samples at once. A possible value of $samplesPerIteration$ parameter could be something in the vicinity of the sample size for step one since we have already used this sample size in step one. Consequently, we know that it fits into memory.

5.2.2 Computational cost of AttentivePeeling

The computational cost of AttentivePeeling is the sum of the computational cost for EagerPeeling and the progressive sampling part. The the computational cost of progressive sampling part is as follows:
Sampling a single graph is $\mathcal{O}(\mathcal{G}[E]|)$, where $\mathcal{G}[E]|$ is the set of edges in the uncertain graph $\mathbf{G}$. Filtering the candidates can be done in $\mathcal{O}(|candidates|))$, since we iterate over all candidates twice. Furthermore, a DFS is run on each new sample in each iteration, which is $\mathcal{O}((\mathcal{G}[V]| + \mathcal{G}[E]|) \cdot |samples|)$. Each iteration then takes $\mathcal{O}(\mathcal{G}[E]| \cdot |samples| + |candidates| + (\mathcal{G}[V]| + \mathcal{G}[E]|) \cdot |samples|)$. Filtering candidates is negligible compared to DFS and sampling, hence the running time of each iteration is $\mathcal{O}((\mathcal{G}[V]| + \mathcal{G}[E]|) \cdot |samples|)$.
From section 5.1.3 we have that the runtime of EagerPeeling is $\mathcal{O}(2^{|\mathcal{G}[V]|} \cdot |T| \cdot |\mathcal{G}[V]|)$.
Consequently, the computational cost of AttentivePeeling is $\mathcal{O}(2^{|\mathcal{G}[V]|} \cdot |T| \cdot |\mathcal{G}[V]| + (\mathcal{G}[V]| + \mathcal{G}[E]|) \cdot |samples|)$.

5.2.3 Statistical guarantee for AttentivePeeling step 1

LEMMA 8: *With a sample size of* $q \geq \frac{1}{2\epsilon^2}((n+2)ln(2) - ln(\delta))$ *with probability* $1 - \frac{\delta}{2}$ *step 1 of AttentivePeeling returns a set of cohesive sets* $\hat{\mathcal{K}}$ *containing all top-k reliable subgraphs.*

The proof of Lemma 8 is as follows: Let $\mathcal{K}$ be the set of subgraphs that have the true k highest reliabilities. Step 1 of the algorithm seeks to find a set $\hat{\mathcal{K}} \supseteq \mathcal{K}$, i.e. minimizing the number of false-negatives while filtering many subgraphs that are not in $\mathcal{K}$.
Step 1 maintains a threshold of the k'th reliability estimate and includes all subgraphs with a reliability estimate above $\hat{R}[m_{\hat{k}}] - 2\epsilon$: $\hat{\mathcal{K}} = \{m : \hat{R}[m] \geq \hat{R}[m_{\hat{k}}] - 2\epsilon\}$. By choosing a large enough sample size it can be guaranteed that $\mathcal{K} \subseteq \hat{\mathcal{K}}$ except with probability $\frac{\delta}{2}$. The subset condition holds if no graph reliability is estimated outside it's ϵ-range. The sample size q must be chosen

similarly to what is described in section 5.1.5, where we define A_i to be the event of estimating the i'th subgraph outside its ϵ-range: $P(A_i) = Pr(|R[m_i] - \hat{R}[m_i]| \geq \epsilon) \leq 2e^{-2q\epsilon^2}$.
The probability of any subgraph being wrongly estimated should be less than $\frac{\delta}{2}$:

$$Pr(\bigcup_{i=1}^{2^n} A_i) \leq \sum_{i=1}^{2^n} Pr(A_i) \leq 2^n 2e^{-2q\epsilon^2} \leq \frac{\delta}{2}$$

This means that the probability of estimating each subgraph wrongly can at most be:

$$Pr(A_i) \leq e^{-2q\epsilon^2} \leq \frac{\delta}{2^{n+2}}$$

Solving for q we get

$$\frac{1}{2\epsilon^2}((n+2)ln(2) - ln(\delta)) \leq q$$

If step 1 of AttentivePeeling is run using the above sample size, it can be guaranteed that except with probability $\frac{\delta}{2}$ we include the actual top-k results in the candidate set $\hat{\mathcal{K}}$. The important thing here is that in practice $|\hat{\mathcal{K}}| \ll 2^n$, which keeps the running time low when increasing the sample size, as step 2 will only check the connectivity of $|\hat{\mathcal{K}}|$ subgraphs.

5.2.4 Statistical guarantee for AttentivePeeling step 2

LEMMA 9: *Given a set $\hat{\mathcal{K}}$ where $\mathcal{K} \subseteq \hat{\mathcal{K}}$, step 2 of AttentivePeeling returns cohesive sets containing all top-k reliable subgraphs ($\mathcal{K}$) with probability $1 - \frac{\delta}{2}$.*

The proof of Lemma 9 is as follows: In step 2, more samples are drawn while keeping a fixed probability $\frac{\delta}{2}$ of estimating outside the ϵ-range. The effect of increasing the sample size is that the ϵ-value decreases, making it possible to determine which of the two subgraphs is more reliable. Once the difference between two reliability estimates is bigger than 2ϵ we get that $R[m_i] > R[m_j]$, if none of them are estimated outside their ϵ-range. Thus, we can conclude with probability $1 - \frac{\delta}{2}$ that $R[m_i] > R[m_j]$, meaning that m_i is the more reliable subgraph.

The current ϵ can be calculated as a function of how many subgraphs are left in $\hat{\mathcal{K}}$, the number of samples, and δ. We can calculate ϵ using the fact, that we only allow the subgraphs in $\hat{\mathcal{K}}$ to be estimated outside their ϵ-range with probability at most $\frac{\delta}{2}$:

$$Pr(\bigcup_{i=1}^{|\hat{\mathcal{K}}|} A_i) \leq \sum_{i=1}^{|\hat{\mathcal{K}}|} Pr(A_i) \leq |\hat{\mathcal{K}}|2e^{-2q\epsilon^2} \leq \frac{\delta}{2}$$

The candidates in $\hat{\mathcal{K}}$, are filtered according to the above formula in every iteration of step 2, thus accumulating the probability of failure. t is defined to be the maximum number of iterations before reaching ϵ_{limit}. Hence, we get that the probability of estimating the reliability outside its ϵ-range can in each iteration be at most

$$Pr(A_i) \leq |\hat{\mathcal{K}}|2e^{-2q\epsilon^2} \leq \frac{\delta}{2t}$$

Isolating for ϵ:

$$\begin{aligned}
\frac{\delta}{2t} &\geq |\hat{\mathcal{K}}|2e^{-2q\epsilon^2} \\
\frac{\delta}{4|\hat{\mathcal{K}}|t} &\geq e^{-2q\epsilon^2} \\
\frac{4|\hat{\mathcal{K}}|t}{\delta} &\leq e^{2q\epsilon^2} \\
ln\left(\frac{4|\hat{\mathcal{K}}|t}{\delta}\right) &\leq 2q\epsilon^2 \\
\frac{1}{2q}ln\left(\frac{4|\hat{\mathcal{K}}|t}{\delta}\right) &\leq \epsilon^2 \\
\sqrt{\frac{1}{2q}ln\left(\frac{4|\hat{\mathcal{K}}|t}{\delta}\right)} &\leq \epsilon
\end{aligned}$$

The value of t is calculated by inserting ϵ_{limit} and replacing q with $t \cdot s$ where s is the number of samples per iteration:

$$\begin{aligned}
\frac{1}{2t \cdot s}ln\left(\frac{4|\hat{\mathcal{K}}|t}{\delta}\right) &\leq \epsilon_{limit}^2 \\
ln(\frac{4|\hat{\mathcal{K}}|t}{\delta}) &\leq \epsilon_{limit}^2 \cdot 2t \cdot s
\end{aligned}$$

Define $y = -ln(\frac{4|\hat{\mathcal{K}}|t}{\delta}) \implies t = \frac{e^{-y}\delta}{4k}$. By substitution we get

$$\begin{aligned}
-y &= \epsilon_{limit}^2 \cdot 2s \cdot \frac{e^{-y}\delta}{4k} \\
ye^y &= \frac{-\epsilon_{limit}^2 s\delta}{2k} \\
y &= W(-\frac{\epsilon_{limit}^2 s \cdot t}{2k})
\end{aligned}$$

W is the Lambert-W function which is the inverse of $f(x) = xe^x$. Lastly we get

$$t = \frac{e^{-W(-\frac{\epsilon_{limit}^2 s \cdot t}{2k})}\delta}{4k}$$

Using the calculated ϵ-value, any candidate m_j that has $\hat{R}[m_j] \leq \hat{R}[m_{\hat{k}}] - 2\epsilon$ can then be excluded safely, except with a small probability $\frac{\delta}{2}$.

AttentivePeeling property: *AttentivePeeling returns the correct k most reliable subgraphs with probability $1 - \delta$.*
This proof follows from Lemma 8 and 9. The probability of having a false-negative in step 1 is

$Pr(step1Failure) = \frac{\delta}{2}$ (Lemma 8) and the probability of filtering a wrong candidate in step 2 is also $Pr(step2Failure) = \frac{\delta}{2}$ (Lemma 9). Using union bound the probability of either of these happening is at most $Pr(step1Failure) + Pr(step2Failure) \leq \delta$. Which concludes the strong guarantee, that the algorithm finds the actual k most reliable subgraphs with probability $1 - \delta$.

5.3 Naive approach

To compare the novel top-k algorithms devised in the previous two sections, we have developed a naive solution that extends the work of Jin et al. The naive algorithm uses the $FastPeeling$ and $MiningNonMaximal$ algorithms to find kFCS in a set of sampled graphs. Instead of a user-specified threshold, the naive top-k algorithm starts with a high threshold and checks if more than k cohesive sets exist. If not, it decreases the threshold and runs the algorithm again with the decreased threshold. As soon as the algorithm with a given threshold produces more than k cohesive sets, a linear search in the results will be the fastest way to retrieve the k most frequent cohesive sets. This is clearly a naive approach since the algorithm might run the costly Apriori algorithms from scratch several times without finding k cohesive sets.

5.3.1 Description and pseudocode

Algorithm 8: NaivePeeling

```
   Result: kFCS
1  Parameter: D, G_1, ..., G_N ;
2  Parameter: T {Transactional database of connected components};
3  Parameter: k ;
5  MFCS = kFCS = FCS = ∅ ;
7  θ = 0.95;
9  while |FCS| < k {While less than k FCS} do
11 |   MFI = getMFI(T, θ) ;
13 |   MFCS = FastPeeling(D, MFI, θ) ;
15 |   if |MFCS| > 0 {If any MFCS are found} then
17 |   |   FCS = NonMaximal(MFCS, θ) ∪ MFCS;
18 |   end
20 |   θ = θ − 0.1 ;
21 end
23 kFCS = Prune(FCS) {Keep k most reliable FCS};
```

As input, the naive top-k algorithm takes D, T, k, where D is the graph database of sampled worlds, T is a transactional database of the connected components. The algorithm starts with an initial threshold value of 0.95 and then decreases it in small steps of 0.10 if we do not find k cohesive sets. Alternatively, we could have used a binary step approach, starting at 0.5. However, since low thresholds greatly increase the number of components to enumerate, it is arguably a better approach to start with a high threshold value and decrease it in small steps. In each iteration, Apriori is used (line 11) to retrieve the MFI (equivalent to MFLS from Lemma 4). The MFLS are used as an input to the FastPeeling algorithm (line 12). The FastPeeling algorithm then finds the set of MFCS, and if the returned set of MFCS is non-empty, we use the MiningNonMaximal algorithm to return all non-maximal FCS (line 15-16). If $MFCS \cup FCS \geq k$, we return the k most frequent ones. If not, we decrease the threshold (line 20).

The correctness of the algorithm follows the correctness of FastPeeling and MiningNonMaximal algorithms devised by Jin et al. The NaivePeeling algorithm just ensures that we get the k most frequent sets by running with different thresholds.

5.3.2 Statistical guarantees for NaivePeeling

Even though the naive approach extends the work of Jin et al. it does not give the same guarantees for precision and recall since they are mutually inclusive. Instead, we use the same approximate top-k results as in the EagerPeeling algorithm described in section 5.1.5. The reason for this is that the algorithm just picks k results with the highest estimated reliabilities and uses the same sample size as the novel approach. This gives the same statistical guarantees, except with a slightly lower probability, because the sample size has not been adjusted for the probability of failing over different reruns. To achieve the exact same guarantees as EagerPeeling, the sample size must be increased to compensate for the reruns of FastPeeling. This will only make the algorithm slower and is thus irrelevant for the sake of stating the contributions of the novel approaches.

6 Experiments

We will now describe metrics of evaluation, compare the different algorithms in terms of guarantees and implementations and describe and analyze the experiments.

6.1 Evaluation metrics

In this section we will describe the metrics of evaluation. Here we are interested in the questions of how precise the algorithms are in terms of $Precision@k$, and how efficient the algorithms are in term of runtime.

6.1.1 Precision

When evaluating for precision we cannot use the same approach as Jin et al, as we do not have the same notion of precision and recall in the top-k algorithms. Instead, we evaluate for $Precision@k$ as discussed in section 5.

In order to reason about the precision, a ground truth is needed. As earlier stated, the network reliability problem is #P-complete, and we can thus only determine exact reliabilities for small graphs. In section 5.2.3 we argue, that the attentive algorithm will have a $Precision@k$ of 1. This allows us to use the results of this algorithm as the ground truth for larger graphs, where computing the exact reliabilities is infeasible. In order to validate the claim that the results of the attentive algorithm have a $Precision@k$ of 1, we have computed the true top 5 most frequent sets in the graph from Figure 1 and compared them to the results of EagerPeeling+ and AttentivePeeling. All algorithms are run with $k = 5$, $\epsilon = 0.05$, $\delta = 0.001$, and AttentivePeeling is run with an ϵ_{limit} of 10^{-4}. We run the algorithms with a δ value that is a factor 10 smaller than in the other experiments to provide accurate reliability estimates with higher probability.

Exact	Attentive	Eager+
$R[\{A,B,D,G\}] = 0.5089$	$\hat{R}[\{A,B,D,G\}] = 0.5089$	$\hat{R}[\{A,B,D,G\}] = 0.5180$
$R[\{E,F,G\}] = 0.4999$	$\hat{R}[\{B,C,D\}] = 0.5003$	$\hat{R}[\{B,C,D\}] = 0.5086$
$R[\{B,C,D\}] = 0.4999$	$\hat{R}[\{E,F,G\}] = 0.4996$	$\hat{R}[\{E,F,G\}] = 0.5017$
$R[\{A,C,D\}] = 0.4899$	$\hat{R}[\{A,C,D\}] = 0.4888$	$\hat{R}[\{A,C,D\}] = 0.4981$
$R[\{A,E,F\}] = 0.4799$	$\hat{R}[\{A,E,F\}] = 0.4799$	$\hat{R}[\{A,E,F,G\}] = 0.4779$

Table 2: Table of precision. Each cell in the "Exact" column contains the reliability for a given subgraph. In the other columns, this is instead a reliability estimate.

The results in table 2 show that AttentivePeeling returns the same subgraphs as the exact algorithm. EagerPeeling, on the other hand, fails to produce the same result, as it returns the subgraph $\{A, E, F, G\}$ (marked in red), where the other two return $\{A, E, F\}$. Lastly, it is worth noting, that if the algorithms were run with $k = 2$, the exact and attentive results might not be equal, as the reliabilities of $\{E, F, G\}$ and $\{B, C, D\}$ are equal to the 15th decimal (Only four decimals shown in Table 2). This means that as AttentivePeeling has an ϵ_{limit} of 10^{-4}, it would consider them equal and return either one of them. These results indicate that AttentivePeeling can be used as a ground truth. However, as we have only validated this for a single small graph, we use this ground truth knowing it is subject to exceptions.

6.1.2 Efficiency

In order to thoroughly test and compare the efficiency of EagerPeeling, EagerPeeling+, AttentivePeeling, and NaivePeeling we perform experiments on different synthetic graphs. This means, that our metric of evaluation in terms of efficiency, will be runtime as a function of the varying variable for a given experiment.

The reason for using synthetic graphs is, that we can compare the performance of all algorithms with regard to isolated variables, and thus control the conditions of the experiments. We conduct experiments varying the following variables: *number of vertices, edge degree, required cohesive sets* k, ϵ, δ. For instance, when experimenting on the number of vertices, we make sure to use the same k, ϵ and δ values for the algorithm, and to let the vertices in the graph have the same average edge degree.

For the efficiency experiments, ten graphs are constructed with the specifications of a given experiment. This is done as the structure of the graph is still somewhat random, and can influence the performance of the algorithms. We can then run each experiment ten times. Furthermore, in five of the graphs, the probabilities on the edges are chosen uniformly at random between 0 and 1, and in the other five, we use the multi-valency model, where the probabilities on the edges are chosen uniformly at random between $[0.1, 0.3, 0.5, 0.7, 0.9]$. This is done as the multi-valency model better reflects real-world data, where probabilities are not uniform between 0 and 1 [29].

When creating synthetic uncertain graphs, we use randomness in the generation of both their edges and the associated edge probabilities. When we define the edge degree for the graph, it is the average edge degree and not the number of edges for each vertex. This means, that even though a graph has an average edge degree of 1.5, some vertices might have four edges, and some might have zero.
All algorithms were implemented using C++ and the Standard Template Library (STL), and

were conducted on an Intel Core i9 10940X 3.3GHz, 256 GB ram and running Linux. The program was not parallelized.

6.2 Algorithms compared

In order to fairly compare and test the different approaches, we have compared the data structures of the C++ implementations and ensured that they are in line with each other. This is done because operations such as lookups, deletions, and insertions vary greatly in runtime complexity for different data structures.

It is also important to note that the different algorithms provide different guarantees for the results. EagerPeeling and NaivePeeling both provide the guarantee that the returned reliability estimates are at least the true k'th reliability - 2ϵ with probability $1 - \delta$. EagerPeeling+ provides the same guarantee as Eager- and NaivePeeling, but with a tighter bound of ϵ. AttentivePeeling provides the guarantee that with probability $1 - \delta$ the algorithm outputs the k most frequent cohesive sets.

6.3 Precision of EagerPeeling+

With precaution, we have established that AttentivePeeling produces results equivalent to the true k most frequent cohesive sets. We can thus reason about the $Precision@k$ of the eager algorithm by comparing the results of this algorithm to the results of the attentive algorithm. For this purpose, we have constructed an experiment, where we run both algorithms, using the parameters $k = 10, \epsilon = 0.05, \delta = 0.01, \epsilon_{limit} = 10^{-4}$, on graphs with a varying number of vertices and an edge degree of 1.5. Furthermore, we construct five synthetic graphs for each number of vertices, meaning that we run each experiment five times. This is done to get results congruent with the true precision of EagerPeeling.

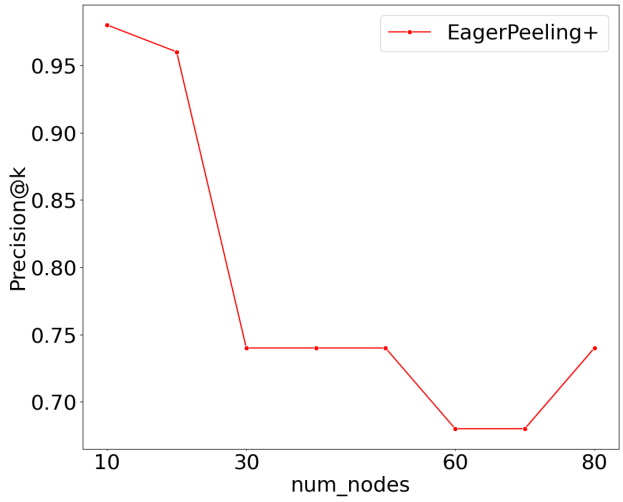


Figure 3: Precision experiments with a varying number of vertices, $k = 10$, edge degree $= 1.5$, $\delta = 0.01$ and $\epsilon = 0.05$.

The result of the experiments is seen in Figure 3. The number of vertices is plotted on the horizontal axis and average Precision@k is plotted on the vertical axis. By plotting the precision

vs the number of vertices, it is evident, that EagerPeeling+ does not give the same precision as AttentivePeeling. Instead, the precision decreases as the number of vertices and hence also edges increase. The reason for this pattern is most likely that the number of top-k candidates increases as the number of vertices and edges increases. This means that the reliabilities of the candidates will be closer, and thus more reliability estimates will have overlapping ϵ-ranges. Hereby, the algorithm may potentially return the wrong candidates.

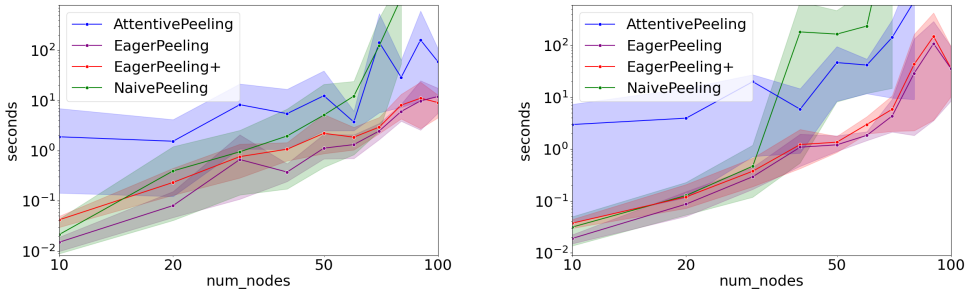
6.4 Efficiency experiments

In this section, we discuss the results of the experiments. In each figure, we plot the variable we are analyzing vs the runtime in seconds. Each line plot shows the average runtime and the colored area around a line shows a $[min; max]$ runtime interval that all experiments lie within. We choose to visualize the mean, min and max value since the runtime varies a lot depending on the graph structure. Notice that we use logarithmic scales for both axes in all plots. All experiments with AttentivePeeling use $\epsilon_{limit} = \cdot 10^{-4}$, meaning that two subgraphs $\mathcal{G}[V_i]$ and $\mathcal{G}[V_j]$ with reliabilites $|\mathcal{G}[V_i] - \mathcal{G}[V_j]| \leq 10^{-4}$ are considered to have the same reliability estimate.

A general pattern in all experiments is that the runtime of NaivePeeling varies more than the other algorithms. The naive approach is more dependent on the data set since the algorithm starts with a threshold of 0.95 and decreases it in each iteration if it does not find k subgraphs. This will make NaivePeeling faster on graphs, where there are many subgraphs with high reliability but slow when this is not the case. EagerPeeling, EagerPeeling+ and AttentivePeeling all start from low thresholds and increase it incrementally based on the current k'th most reliable subgraph that has been discovered. This gives a more stable runtime, which performs much better than the naive approach if the top-k subgraphs are lower than 0.95. The eager and attentive algorithms still have runtimes that vary, since they depend on the order in which the reliabilities for subgraphs are calculated. If the algorithms are "lucky" and find a subgraph with high reliability early, they are able to prune away most of the candidates and thus the runtime will be better.

6.4.1 Varying number of vertices

In Figure 4 we see the different experiments for the number of vertices plotted vs time in seconds for both uniform (a) and multi-valency (b) sampling.



(a) Uniform sampling.

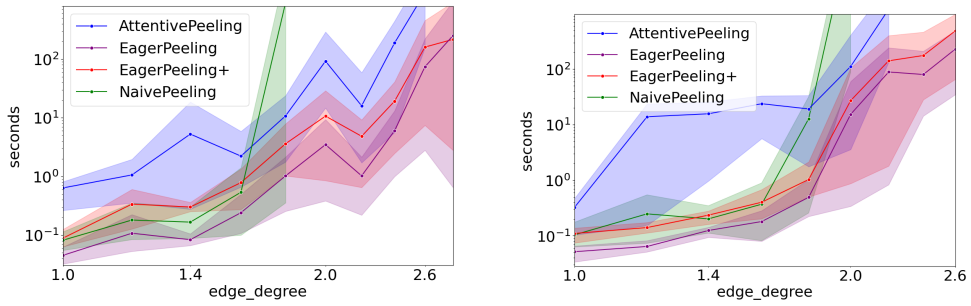
(b) Multi valency sampling.

Figure 4: Plot of number of vertices vs time in seconds with fixed $k = 3$, edge degree $= 1.5$, $\delta = 0.01$ and $\epsilon = 0.05$.

It is evident that the number of vertices greatly affects the runtime of all algorithms. Furthermore, we can see that EagerPeeling and EagerPeeling+ have very similar runtimes and that they outperform both the NaivePeeling and AttentivePeeling for min, max and mean runtimes when run on larger graphs. The mean value of the EagerPeeling experiments show an almost polynomial increase in runtime when the number of vertices increases. The result of using uniform sampling vs multi-valency sampling does not seem to affect the runtime much for varying numbers of vertices. However, AttentivePeeling is slower in the multi-valency experiments. This is probably because a smaller sample space for the edge probabilities will result in more similar reliability estimates, which the progressive sampling step needs to distinguish.

6.4.2 Varying edge degree

In Figure 5, we see how the edge degree affects the running time of the algorithms.



(a) Uniform sampling.

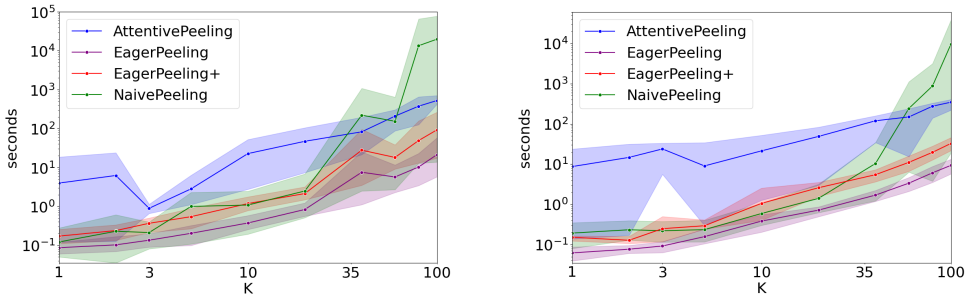
(b) Multi valency sampling.

Figure 5: Plot of edge degree vs time in seconds with fixed $k = 3$ number of vertices $= 30$, $\delta = 0.01$ and $\epsilon = 0.05$.

In this experiment, we see a steep increase in runtime, when the number of edges in the graph increases. We see that NaivePeeling only runs up to an edge degree of 1.8 in (a) and 2.0 in (b) and in AttentivePeeling runs till an edge degree of 2.4 in (a) and 2.6 in (b). This shows that Naive- and AttentivePeeling have a significant higher runtime than the EagerPeeling algorithms, especially when increasing the number of edges, as the number of candidates for cohesive sets then increases. This requires all algorithms to generate more candidates in their respective versions of the Apriori algorithms, which we know bounds the running time. Furthermore, in the valency experiments, we see that AttentivePeeling reaches a plateau in the runtime. This is most likely because the edge probabilities are drawn from a limited set of values. Therefore, more subgraphs will have similar reliabilities, which might cause the progressive sampling step of AttentivePeeling to sample until the ϵ_{limit} is met. Since the progressive sampling step is only linearly dependent on the number of edges, the progressive sampling step may account for a larger portion of the runtime on smaller graphs. This experiment indicates that the edge degree has a large impact on the runtime compared to the number of vertices.

6.4.3 Varying k

In the experiments for different values of k, it makes sense to use the same graph for different k values, such that the runtime depends on k and not on how the different graphs were generated.



(a) Uniform sampling.

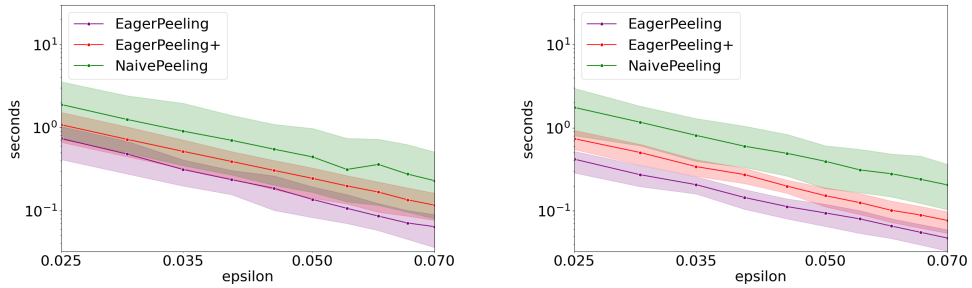
(b) Multi valency sampling.

Figure 6: Plot of varying values of k vs time in seconds with fixed number of vertices $= 30$, edge degree $= 1.5$, $\delta = 0.01$ and $\epsilon = 0.05$.

Figure 6 shows the results for the experiments for *varying* k. Here, we see a more moderately increasing curve compared to the experiments of edge degree and number of vertices. This is however not the case for the naive algorithm. An explanation could be, that with an increased k the naive algorithm is less likely to find k cohesive sets in earlier iterations. Hereby, as k is increased, NaivePeeling may run more iterations of the computationally heavy Apriori algorithm. The other algorithms have a more moderate increase in running time, this increase probably stems from the fact that the k'th reliability is lower and hence fewer candidates can be filtered away in the Apriori. Overall we see that increasing k has less influence on the runtime compared to increasing the number of vertices and edges. As k increases, the sample size of EagerPeeling+ increases. This is not the case for EagerPeeling, and we should thus see a larger difference in the runtime of the two approaches as k is increased. This is not evident in the experiment, possibly because we only have a logarithmic dependency on k and the values for k are not large enough to see a significant difference in runtime.

6.4.4 Varying ϵ

In this experiment, we test the runtime of varying values of ϵ. We do not run the ϵ-experiment for AttentivePeeling since ϵ is only used for the first step and does not affect the progressive sampling step. The runtime of AttentivePeeling instead depends on the value of ϵ_{limit} and how closely the estimated reliabilities are.



(a) Uniform sampling.

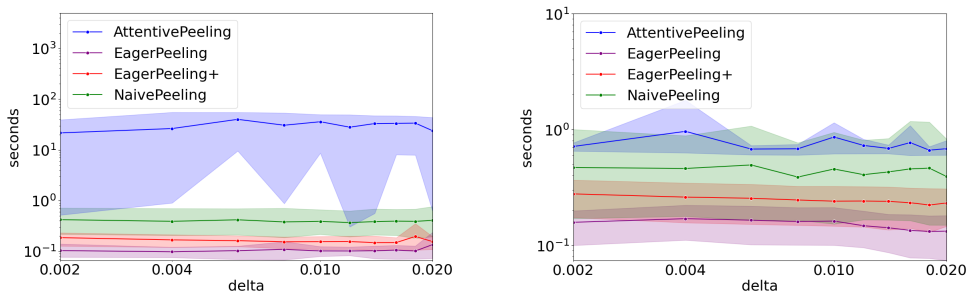
(b) Multi valency sampling.

Figure 7: Plot of varying values of ϵ vs time in seconds with number of vertices = 30, edge degree = 1.5 and $\delta = 0.01$

We see that the runtimes of the three algorithms have similar decreasing runtime for decreasing values of ϵ. This may be because the sample size for all three algorithms is quadratically dependent on ϵ.

6.4.5 Varying δ

In Figure 8 we see the experiment of varying δ.



(a) Uniform sampling.

(b) Multi valency sampling.

Figure 8: Plot of varying values of δ vs time in seconds with number of vertices = 30, edge degree = 1.5 and $\delta = 0.01$.

From the plot in Figure 8, we see that the runtimes are not affected much by different values of δ. A possible explanation is, that the sample size has a logarithmic dependency on δ, making the differences in sample size insignificant compared to for example ϵ or the number of vertices. The runtime of AttentivePeeling varies a lot compared to the other algorithms. The variations are most likely caused by the progressive sampling, as it is only required when reliability estimates of the k'th and k+1'th are within ϵ of each other.

6.4.6 Scaling example

In all previous experiments, the edge probabilities have been uniformly distributed either directly or by choosing uniformly between multi-valency values. This may cause that many subgraphs have similar reliabilities, which makes it hard to prune out candidates when updating the threshold. In this experiment, we try running EagerPeeling+ on a dataset with a skewed distribution, such that most edges have a low probability, while fewer edges have a higher probability. The effect is, that once a frequent cohesive set has been found, the algorithm demonstrates its ability to prune out all the redundant infrequent itemsets quickly, which in turn decreases the execution time of the algorithm.
We created an experiment, where graphs are generated with edge probabilities that follow an F-distribution [30]. M values are drawn from an F-distribution, using Numpy [30] and are normalized, such that the values lie within the range $(0; 1)$. A plot of the F-distribution we used for generating uncertain graphs is shown in Figure 9. For selecting F-distribution parameters, we tried different values and found one with a hard skew.

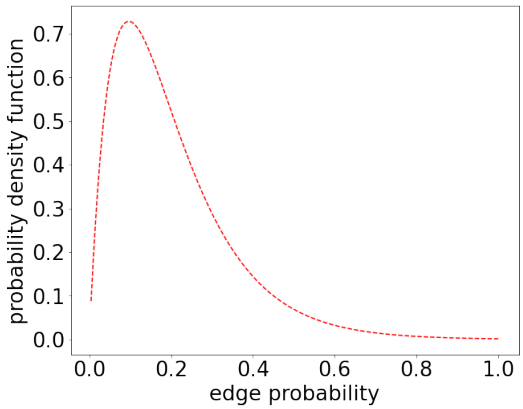


Figure 9: F-distribution with 90 degrees of freedom in denominator, 4 degrees of freedom in numerator and normalized egde probabilities in the range $(0; 1)$

In this experiment, we only use the EagerPeeling+ algorithm since it only performs slightly worse than EagerPeeling, while giving better guarantees. Consequently, EagerPeeling+ would be the algorithm to use, if you want to run on bigger graphs. The results in Figure 10 show that the EagerPeeling+ algorithm is able to scale on bigger graphs when the edge probabilities have a skewed distribution like the F-distribution.

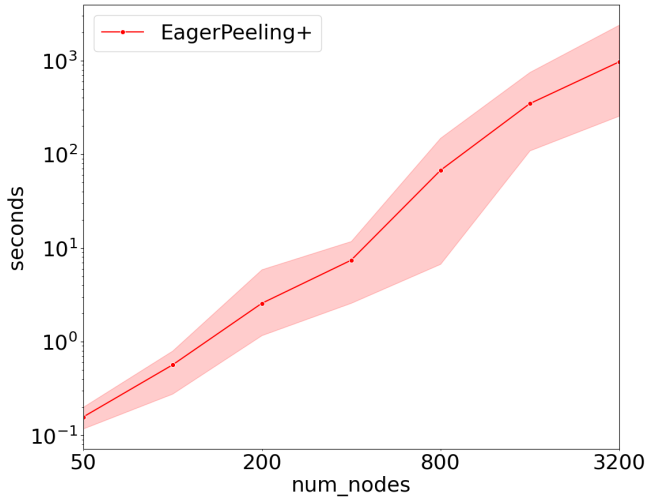


Figure 10: Varying number of vertices for edge probabilities sampled from F-distribution in Figure 9. With $k = 1$, edge degree $= 1.5$, $\epsilon = 0.05$, $\delta = 0.01$.

7 Future work

Lastly, we will discuss ideas for future work. We did not attempt to complete these ideas, either because that we did not have enough time to implement them or because they were not within the scope of the project.

7.1 Bit representation of samples

In order to reduce the need for memory, it is possible to represent the possible graphs using bit strings. Each bit would act as an indicator for the presence of each edge, meaning that m possible graphs would be represented as m bitstrings of equal length. This could be especially useful when producing many possible graphs like we do in AttentivePeeling. We did however not manage to implement this within the deadline of this project, and it is hence an idea for future work.

7.2 VC dimension

In all statistical guarantees, we give for our algorithms we use the Chernoff-Hoeffding [23, 24] Bound. This is the same approach used by Jin et al. However, the Chernoff-Hoeffding bound does not take the data and problem at hand into consideration, which might lead to a lower bound on the sample size that is too big. Instead, it might be possible to use the Vapnik–Chervonenkis (VC) [31] dimension to derive better sample sizes that give the same guarantees. The VC-dimension for a space of points is a measure of capacity (complexity or expressive power) for a family of subsets or an indicator function defined for that space. Bounding the VC-dimension for a given set will bound the number of random samples needed for approximating that set. Using VC-dimension for data mining tasks is not well studied in the literature. However, there exist examples of work that provide sample complexity results based on VC dimension for approximate frequent itemset mining problem [32], selectivity estimation [33] and approximate betweenness centrality [34]. The existing literature, that shows use cases of VC-dimension for data mining tasks, motivates finding a VC-dimension for the problem of discovering top-k cohesive sets and using this bound to derive better sample sizes in future work.

7.3 MFI algorithm

As described in section 5.1.2, the MFI algorithm is the algorithm that bounds the complexity of our top-k algorithms. It would be interesting to test other MFI algorithms, e.g. the FP-Growth algorithm [26] or Top-down approaches [27]. This would require devising a similar iterative approach to the one used in EagerPeeling and AttentivePeeling.

7.4 EagerPeeling+ guarantees for AttentivePeeling

The first step in AttentivePeeling keeps a buffer of cohesive sets that satisfy the relaxed threshold of 2ϵ less than the actual threshold. This means that the modified Apriori algorithm in step 1 must run with the relaxed threshold. In order to improve this, it might be possible to make use of the guarantees in EagerPeeling+, where we at the cost of an increased sample size achieve tighter statistical guarantees. By using these tighter guarantees, it might be possible to use a relaxed threshold of only ϵ less than the actual threshold, and thus improve the runtime of Apriori, as we can filter out more candidates. This is a matter for further investigation.

7.5 Edge cutting

Jin et al. use the Edge-Cut Lemma to reduce the edge degree and thus the search space. We initially discarded the idea of including this in the novel approaches, as it would require resampling and rerunning Apriori every time the threshold is updated. However, as the experiments show, the edge degree greatly affects the performance of the algorithms, and it might be worth investigating intelligent ways of including the Edge-Cut Lemma in the novel approaches.

8 Conclusion

In this project, we have presented and reviewed the threshold-based methods for discovering reliable subgraphs in uncertain graphs devised by Jin et al. [1]. In the same setting of uncertain graphs, we have presented three algorithms for mining the k most reliable subgraphs. We have shown that this problem is #P-complete. For this reason, we have presented efficient probabilistic sampling methods for mining the k most reliable subgraphs. These sampling methods provide different ϵ-approximations. We have devised an eager approach that combines an Iterative Apriori algorithm and a peeling process. This approach utilizes a dynamic threshold for discovering reliable subgraphs. For the eager approach, we have devised different sample sizes that provide different guarantees, hence we make the distinction of eager and eager+. Furthermore, we have developed an attentive approach that builds on top of the eager approach by adding a progressive sampling step. The progressive sampling provides tighter guarantees in exchange for the added computational cost. Lastly, we have devised a naive approach, by simply extending the work of Jin et al. This approach is used to evaluate the contributions of the eager and attentive approaches.

We have conducted several experiments that illustrate the precision and efficiency of the eager, eager+, attentive, and naive approaches. From these experiments, we concluded that the novel approaches are more efficient than the naive approach. The eager and eager+ approaches may not provide the true k most reliable subgraphs, since the reliability estimates are ϵ-approximations. They do however prove to be more efficient and to scale better than Attentive- and NaivePeeling. The experiments show that the eager and eager+ approaches are very similar in terms of efficiency, and a natural choice would thus be to use the eager+ approach as it provides better guarantees. While the attentive approach does not scale as well as the eager approaches, it provides better guarantees and is thus a suitable choice of algorithm, when high precision is required. Hereby, choosing between the eager+ and the attentive approach, one has to consider a trade-off between precision and efficiency.

9 Acknowledgements

We thank Cigdem Aslay and Panagiotis Karras for their invaluable inputs and excellent guidance on this project.

10 Appendix

The C++ code of this project can be found at `https://github.com/Sloeschcke/BSc_project`

References

[1] R. Jin, L. Liu, and C. C. Aggarwal, "Discovering highly reliable subgraphs in uncertain graphs," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, (New York, NY, USA), p. 992–1000, Association for Computing Machinery, 2011.

[2] C. Aggarwal, "Managing and mining uncertain data," in *Advances in Database Systems*, 2009.

[3] A. Khan, Y. Ye, L. Chen, and H. V. Jagadish, *On Uncertain Graphs*. Morgan Claypool Publishers, 2018.

[4] S. Jamil, A. Khan, Z. Halim, and A. R. Baig, "Weighted muse for frequent sub-graph pattern finding in uncertain dblp data," in *2011 International Conference on Internet Technology and Applications*, pp. 1–6, 2011.

[5] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi, "The pursuit of a good possible world: Extracting representative instances of uncertain graphs," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, (New York, NY, USA), p. 967–978, Association for Computing Machinery, 2014.

[6] C. Ma, R. Cheng, L. V. S. Lakshmanan, T. Grubenmann, Y. Fang, and X. Li, "Linc: A motif counting algorithm for uncertain graphs," *Proc. VLDB Endow.*, vol. 13, p. 155–168, Oct. 2019.

[7] Y. Yuan, L. Chen, and G. Wang, "Efficiently answering probability threshold-based shortest path queries over uncertain graphs," in *Database Systems for Advanced Applications* (H. Kitagawa, Y. Ishikawa, Q. Li, and C. Watanabe, eds.), (Berlin, Heidelberg), pp. 155–170, Springer Berlin Heidelberg, 2010.

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

[9] D. R. White and F. Harary, "The cohesiveness of blocks in social networks: Node connectivity and conditional density," *Sociological Methodology*, vol. 31, pp. 305–359, 2001.

[10] T. Can, O. Çamoundefinedlu, and A. K. Singh, "Analysis of protein-protein interaction networks using random walks," in *Proceedings of the 5th International Workshop on Bioinformatics*, BIOKDD '05, (New York, NY, USA), p. 61–68, Association for Computing Machinery, 2005.

[11] A. Khan, F. Bonchi, A. Gionis, and F. Gullo, "Fast reliability search in uncertain graphs," in *EDBT*, 2014.

[12] C. J. Colbourn, *The Combinatorics of Network Reliability*. USA: Oxford University Press, Inc., 1987.

[13] A. R. Sharafat and O. Ma'rouzi, "All-terminal network reliability using recursive truncation algorithm," *Reliability, IEEE Transactions on*, vol. 58, pp. 338 – 347, 07 2009.

[14] R. Zhu, Z. Zou, and J. Li, "Top-k reliability search on uncertain graphs," in *2015 IEEE International Conference on Data Mining*, pp. 659–668, 2015.

[15] Y.-L. Cheung and A. W.-C. Fu, "Mining frequent itemsets without support threshold: with and without item constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1052–1069, 2004.

[16] G. Pyun and U. Yun, "Mining top-k frequent patterns with combination reducing techniques," *Applied Intelligence*, vol. 41, pp. 76–98, 07 2014.

[17] S. Kashyap and P. Karras, "Scalable knn search on vertically stored time series," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, (New York, NY, USA), p. 1334–1342, Association for Computing Machinery, 2011.

[18] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "K-nearest neighbors in uncertain graphs," *Proc. VLDB Endow.*, vol. 3, p. 997–1008, Sept. 2010.

[19] H. Ryang and U. Yun, "Top-k high utility pattern mining with effective threshold raising strategies," *Knowledge-Based Systems*, vol. 76, pp. 109–126, 2015.

[20] A. Pietracaprina, M. Riondato, E. Upfal, and F. Vandin, "Mining top-k frequent itemsets through progressive sampling," *Data Mining and Knowledge Discovery*, vol. 21, p. 310–326, Jul 2010.

[21] Wikipedia, "Induced subgraph." "https://en.wikipedia.org/wiki/Induced_subgraph".

[22] G. Rubino, "Network reliability evaluation," *State-of-the art in performance modeling and simulation*, pp. 275–302, 1999.

[23] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics*, vol. 23, pp. 493–507, 1952.

[24] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[25] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo, *et al.*, "Fast discovery of association rules.," *Advances in knowledge discovery and data mining*, vol. 12, no. 1, pp. 307–328, 1996.

[26] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, p. 1–12, May 2000.

[27] D. Burdick, M. Calimlim, and J. Gehrke, "Mafia: a maximal frequent itemset algorithm for transactional databases," in *Proceedings 17th International Conference on Data Engineering*, pp. 443–452, 2001.

[28] Wikipedia, "Component (graph theory)." "https://en.wikipedia.org/wiki/Component_(graph_theory)".

[29] S. Ivanov and P. Karras, "Harvester: Influence optimization in symmetric interaction networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 61–70, 2016.

[30] "numpy.random.f." https://numpy.org/doc/stable/reference/random/generated/numpy.random.f.html.

[31] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264–280, 1971.

[32] M. Riondato and E. Upfal, "Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees," *CoRR*, vol. abs/1111.6937, 2011.

[33] M. Riondato, M. Akdere, U. Cetintemel, S. B. Zdonik, and E. Upfal, "The vc-dimension of queries and selectivity estimation through sampling," 2011.

[34] M. Riondato and E. M. Kornaropoulos, "Fast approximation of betweenness centrality through sampling," in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, (New York, NY, USA), p. 413–422, Association for Computing Machinery, 2014.