Truss Decomposition of Probabilistic Graphs: Semantics and Algorithms

Xin Huang, Wei Lu, Laks V.S. Lakshmanan
University of British Columbia
{xin0, welu, laks}@cs.ubc.ca

ABSTRACT

A key operation in network analysis is the discovery of cohesive subgraphs. The notion of k-truss has gained considerable popularity in this regard, based on its rich structure and efficient computability. However, many complex networks such as social, biological and communication networks feature uncertainty, best modeled using probabilities. Unfortunately the problem of discovering k-trusses in probabilistic graphs has received little attention to date.

In this paper, given a probabilistic graph $\mathcal{G}$, number k and parameter $\gamma \in (0, 1]$, we define a (k, γ)-*truss* as a maximal connected subgraph $\mathcal{H} \subseteq \mathcal{G}$, in which for each edge, the probability that it is contained in at least $(k-2)$ triangles is at least γ. We develop an efficient dynamic programming algorithm for decomposing a probabilistic graph into such maximal (k, γ)-trusses. The above definition is *local* in that the "witness" graphs that has the $(k-2)$ triangles containing an edge in $\mathcal{H}$ may be quite different for distinct edges. Hence, we also propose *global* (k, γ)-truss, which in addition to being a local (k, γ)-truss, has to satisfy the condition that the probability that $\mathcal{H}$ contains a k-truss is at least γ. We show that unlike local (k, γ)-trusses, the global (k, γ)-truss decomposition on a probabilistic graph is intractable. We propose a novel sampling technique which enables approximate discovery of global (k, γ)-trusses with high probability. Our extensive experiments on real datasets demonstrate the efficacy of our proposed approach and the usefulness of local and global (k, γ)-truss.

1. INTRODUCTION

Network data analytics play a key role in many scientific fields such as biological, social and communication networks [18]. A large body of such real-world networks are associated with uncertainty, due to the data collection process, machine-learning methods employed at preprocessing, or privacy-preserving reasons. For instance, in Protein-Protein Interaction (PPI) networks, the edges represent interactions between proteins, which are derived through noisy and error-prone lab experiments and therefore entail uncertainty [19]; Moreover, many edges (interactions) in PPI networks are actually predicted by biologists using algorithms based on features of the proteins, instead of being actually observed in experiments [29]. Naturally, the predictions are associated with certain confidence levels, which are best modeled by probabilities [11, 12, 29]. As another example, in social networks, link prediction and peer influence motivate the need to model interactions between users with uncertainty, and in mobile ad-hoc networks, mobile nodes move and connect to each other, and a link between nodes can be unreliable and may fail with a certain probability. We model such uncertain networks as *probabilistic graphs*, also referred to as *uncertain graphs* in the literature. Each edge in a probabilistic graph is associated with a probability of existence [16, 27].

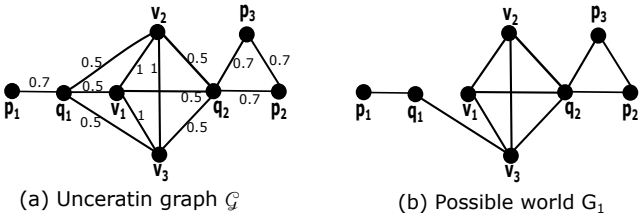
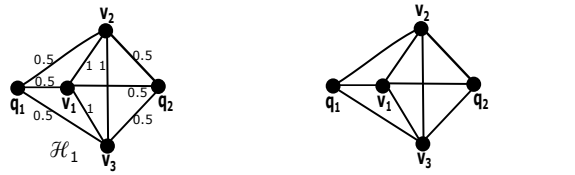


Figure 1: A running example

In many network analysis tasks, a fundamental problem is to identify various cohesive subgraphs [33]. Many notions of cohesive subgraphs have been proposed in the literature; these include cliques, quasi-cliques [24], n-clans [22], n-club [22], k-plexes [28]. The computation of all the above cohesive subgraphs is NP-hard.

A popular notion of cohesive subgraph that has found many recent applications is k-truss. A k-*truss* of a graph, is a subgraph in which each edge is contained in at least $(k-2)$ triangles within that subgraph, i.e., its support is at least $(k-2)$. A k-truss is maximal if it is not a proper subgraph of any other k-truss. For example, in Figure 1(a), ignoring probabilities, the entire graph G is a 2-truss but not a 3-truss. The subgraph induced by the nodes $\{q_1, q_2, v_1, v_2, v_3\}$ is a 4-truss, and the subgraph induced by all nodes but p_1 is a 3-truss. All of these are maximal k-trusses; the subgraph induced by $\{q_1, v_1, v_2, v_3\}$ is a non-maximal 4-truss. The set of all maximal k-trusses of a graph G, for various k, forms the truss decomposition of G. Recently, due to its efficient computability and cohesive structure, truss decomposition has attracted a lot of attention, and has been studied in various settings, including in-memory [9], external-memory [33], and dynamic graphs [15]. Unlike the aforementioned types of cohesive subgraphs, k-trusses [10] can be computed in polynomial time, similar to k-cores [2].

The closest work to us is the probabilistic extension to k-cores [4]. In addition to laying down the semantics, the authors provide efficient algorithms for finding subgraphs of probabilistic graphs that are k-cores with probability over a given threshold. A detailed comparison with this work appears in Section 2 but it's worth noting that k-trusses enjoy much more cohesiveness than k-cores.



(a) Local (4, 0.125)-truss $\mathscr{H}_1$ (b) Possible world of $\mathscr{H}_1$ as a connected 4-truss

Figure 2: A local (k, γ)-truss of $\mathcal{G}$ in Figure 1: $k = 4, \gamma = 0.125$.

The discovery of k-truss in uncertain graphs can be beneficial for a wide range of application domains. For instance, in network science domain, k-trusses as dense subgraphs can represent cohesive groups or communities. k-trusses have successfully become the basis of several community models [15,20]. Moreover, k-truss can be used to speed up the computation of finding maximum cliques as a k-clique must be in a k-truss, which can be significantly smaller than the original graph; Also, k-truss is a useful tool for visualization of complex networks [37]. All above applications can be naturally extended to uncertain graphs. In biology, a fundamental task is to detect and identify *functional modules*, which technically are cohesive subnetworks of proteins, from probabilistic PPI networks [1, 11, 12, 25, 29]. Modules capture subnetworks of proteins strongly expressed together by genes. Thus, detecting modules is highly important and valuable as it helps "assess the disease relevance of certain genes" [11], which further help critical clinical diagnosis of diseases such as cancer. Last but not the least, probabilistic k-truss can be directly applied to task-driven team formation in uncertain social networks, e.g., DBLP collaboration network, LinkedIn (with predicted links), etc. We conduct a case study on DBLP and report detailed results in Section 6.

In light of the above, mining k-trusses in probabilistic graphs is a pressing need. However, to the best of our knowledge, *truss decomposition over probabilistic graphs has not been studied yet.* In this paper, we present a principled extension of k-truss definitions in the presence of uncertainty, and propose efficient k-truss decomposition algorithms. Given a probabilistic graph $\mathcal{G}$, integer k and parameter $\gamma \in (0, 1]$, we define *local (k, γ)-truss* to be a connected subgraph $\mathcal{H} \subseteq \mathcal{G}$, in which for each edge, the probability that it is contained in at least $(k - 2)$ triangles is no less than γ. For example, consider the probabilistic graph $\mathcal{G}$ in Figure 1(a), the probability that the edge (q_1, v_1) is contained in two triangles is $0.5 \cdot (0.5 \cdot 1) \cdot (0.5 \cdot 1) = 0.125$. Figure 2(a) shows a local $(4, 0.125)$-truss $\mathcal{H}_1$ in $\mathcal{G}$. For each edge $e \in \mathcal{H}_1$, the probability that e is contained in at least 2 triangles is at least 0.125.

As with truss decompositions over deterministic graphs, we are interested in finding maximal local (k, γ)-trusses. However, efficient extraction of local (k, γ)-trusses raises challenges. In probabilistic graphs, to check if an edge belongs to a local (k, γ)-truss, the basic operation is to check if the edge has a large enough probability to be contained in at least $(k-2)$ triangles. A straightforward extension of the deterministic triangle counting operation leads to combinatorial blow-ups and is inefficient. Therefore, existing truss decomposition algorithms for deterministic graphs do not work for probabilistic graphs. Fortunately, local (k, γ)-truss has many desirable features. The key point is that the probability of edge support can be computed in polynomial time, relying only on the local structure. In other words, we can avoid its evaluation for every possible world of $\mathcal{G}$. Specifically, we are able to exploit this "locality" and develop an efficient dynamic programming algorithm for finding a decomposition of a probabilistic graph into maximal local (k, γ)-trusses, for various k, given a parameter $\gamma \in (0, 1]$.

The above definition of truss is *local* in that the "witness" graphs having the $(k - 2)$ triangles containing an edge in $\mathcal{H}$ may be quite different for different edges. To mitigate this, we explore another, "global" notion: in a *global* (k, γ)-truss $\mathcal{H} \subseteq \mathcal{G}$, for each edge in $\mathcal{H}$, the probability that it is contained in a k-truss must be at least γ. We show that unlike local (k, γ)-truss, the global (k, γ)-truss decomposition of a probabilistic graph is intractable: specifically, even computing the exact probability that a given subgraph contains a k-truss connecting its nodes is in general #P-hard. Given this, we tackle this issue with a novel sampling scheme that enables an efficient estimation of the said probability. We also show that there are instances where the number of maximal global (k, γ)-trusses can be exponential. Thus, we develop a heuristic approach combined with the sampling technique above for obtaining global (k, γ)-trusses. Each solution reported by our algorithm is guaranteed to be a maximal global (k, γ)-truss with high probability.

To summarize, we make the following contributions:

- We define local and global (k, γ)-truss over probabilistic graphs and motivate the problem of probabilistic truss decomposition associated with these notions (Section 3.1).
- We show that every global (k, γ)-truss is also a local (k, γ)-truss. We also show that computing the probability that a given probabilistic graph contains a k-truss connecting all its nodes is #P-hard and that there are instances where a given probabilistic graph has exponentially many maximal (k, γ)-trusses even when k and γ are fixed (Section 3.2).
- We show that the support probability of any edge, on which local (k, γ)-truss is based, is monotone w.r.t. k. Leveraging this and the structural locality, we develop an efficient dynamic programming algorithm for finding all maximal local (k, γ)-trusses of a probabilistic graph, given γ (Section 4).
- We develop a sampling scheme for approximating the probability of a graph containing a connected k-truss. We present an exact search algorithm and an efficient heuristic for finding approximate maximal global (k, γ)-trusses (Section 5).
- Using extensive experiments on eight real datasets, we test our proposed algorithms for finding maximal local and global (k, γ)-trusses. The results show that our algorithms significantly outperform natural baselines and alternative techniques based on probabilistic extensions to k-cores [4] (Section 6).

2. RELATED WORK

There are mainly two categories of previous work that are related to ours: querying and mining over probabilistic graphs and dense subgraph mining.

2.1 Probabilistic Graphs

The body of work on querying and mining probabilistic graphs is the closest to our work. In the literature, the studies on uncertain graphs have mainly focused on querying [17, 27, 30] and mining, specifically frequent subgraph mining [38, 39], dense subgraph mining [4, 23] and clustering [18, 21]. Jin et al. [17] study the distance-constraint reachability problem, which is a generalization of the classic two terminal reliability problem: for a pair of nodes find the probability that their shortest path distance is under a threshold. This problem is #P-hard and they propose sampling schemes and experimentally show that they are efficient and produce accurate estimates. Potamias et al. [27] study the k-nearest neighbors problem over uncertain graphs and propose sampling strategies for answering k-nearest neighbor queries efficiently.

Zou et al. [38, 39] consider the problem of discovering frequent pattern subgraphs in an input probabilistic graph. One of the key properties that helps efficient discovery of frequent pattern graphs is monotonicity: if a pattern graph is frequent then any subgraph of that pattern is also frequent. Jin et al. [16] investigate the problem

of discovering highly reliable subgraphs of probabilistic graphs, in which the connectivity holds with a high probability. This problem is intractable and the authors develop an efficient approach using a combination of sampling and frequent cohesive set discovery. Bonchi et al. [4] extend the notion of k-core to probabilistic graphs. Since k-cores are a kind of dense subgraphs, a comparison with [4] appears in the next subsection. The problem of finding truss decomposition over probabilistic graphs has not been studied before, to our knowledge. Specifically, the notions of local and global (k,γ)-trusses proposed in this paper are novel. None of the techniques proposed in the aforementioned related work can be directly used to find maximal local/global (k,γ)-trusses. One unique challenge we face in case of global (k,γ)-trusses, unlike some of the related work, is the lack of any monotonicity properties (see Section 5): given two probabilistic graphs $\mathcal{H}_1 \subset \mathcal{H}_2$, if one of them is a global (k,γ)-truss, then the other may or may not be a (k,γ)-truss. This makes their discovery particularly challenging. Furthermore, unlike highly reliable subgraphs, our global (k,γ)-trusses needs to be not just connected, but also be densely connected with high probability.

2.2 Dense Subgraph Mining

In deterministic graphs, there are various definitions of dense subgraph patterns, including clique [5, 7, 34, 35], quasi-clique [31], k-core [2, 6] and k-truss [9, 33]. Truss decomposition has been studied in various settings, including in-memory algorithms [9,36], external-memory algorithms [33], and MapReduce algorithm [10]. However, as far as we know, probabilistic truss decomposition has not been studied yet. Based on the k-truss definition [9], we propose two novel types of local and global (k,γ)-truss definitions appropriate for probabilistic graphs. Core decomposition has also been studied in both in-memory and external memory [6] settings. Recently, Bonchi et al. [4] extend the core decomposition from deterministic graphs to probabilistic graphs. They define a (k,η)-core as a subgraph of a probabilistic graph in which the degree of each node is at least k, with probability no less than η. Our local (k,γ)-truss is similar to (k,η)-core, since both are defined on the local structure. (k,γ)-Truss emphasizes the number of triangles supporting an edge, but (k,η)-core focuses on node degree. It is well-known that k-trusses enjoy a higher density (e.g., clustering coefficient) than k-cores and are more cohesive. This property carries over to probabilistic graphs and is also borne out by our experiments (see Section 6). On the other hand, in this paper, we also propose a global (k,γ)-truss, that is based on the probability of each edge belonging to a connected k-truss possible world. This is a graph-level, holistic constraint, which has no parallel in any of the previous work, to the best of our knowledge.

3. TRUSSES IN PROBABILISTIC GRAPHS

k-Truss in deterministic graphs. Let $G = (V, E)$ be an undirected, unweighted simple graph, with vertices V and edges $E \subseteq V \times V$. A cycle of length 3 is called a *triangle*; we let $\triangle_{uvw}$ denote a triangle with vertices $u, v, w \in V$. Let $H = (V_H, E_H)$ be an induced subgraph of G, i.e., $V_H \subseteq V$ and $E_H = \{(u,v) \in E \mid u, v \in V_H\}$. Henceforth, by subgraph we mean an induced subgraph. For any edge $e = (u,v) \in E_H$, we define the *support* of e in H as the number of triangles in H that contain e: $\mathsf{sup}_H(e) = |\{\triangle_{uvw} \mid (u,w),(v,w) \in E_H\}|$. A subgraph H of G is called a *k-truss* iff the support of every edge in this subgraph is at least $k-2$. Formally,

DEFINITION 1 (k-TRUSS). *Let $H = (V_H, E_H)$ be a subgraph of $G = (V, E)$ and $k \geq 2$ be any integer. Then, H is a* k-truss *if and only if for all $e \in E_H$, $\mathsf{sup}_H(e) \geq k-2$.*

A k-truss H is *maximal* if it is not a subgraph of any other k-truss. Given a deterministic graph G, the task of *truss decomposition* is to compute all maximal k-trusses of G for all $2 \leq k \leq k_{\max}$, where $k_{\max}$ is the largest support of any edge [33]; we do not need to know this value beforehand, since the decomposition process will automatically reveal it. A useful notion in truss decomposition is the *trussness* of an edge e in a graph G, defined as the maximum k for which e is contained in a k-truss subgraph of G.

Truss decomposition can be done in polynomial time using an iterative removal algorithm [9,33]. Observe that any connected graph is itself a 2-truss by definition, and thus we start from $k = 3$. The algorithm iteratively removes edges whose support is smaller than $k-2$ and updates the support of affected edges, until the remaining graph is a k-truss (or empty). The update is crucial for correctness, since when (u,v) is removed, all triangles $\triangle_{uvw}$ disappear and the support of (u,w) and (v,w) should be decreased by 1. As long as the remaining graph is not empty, we increment k by 1 and repeat the iterative edge removal process. We stop when the graph is empty, and the current k is thus $k_{\max}$. This algorithm is due to [9], which is then improved by [33] using hashing and sorting, achieving $O(m^{1.5})$ time and uses $O(m+n)$ space complexity.

Probabilistic graphs. Let $\mathcal{G} = (V, E, p)$ denote a *probabilistic* graph, where $p\colon E \to [0,1]$ is a function that maps each edge $e \in E$ to its existence probability $p(e)$. Each edge is typically assumed to exist independently [16]. A well-known approach to analyzing and reasoning about probabilistic graphs is to use *possible worlds*: each possible world is a deterministic graph instantiation of $\mathcal{G}$, where only a subset of edges exist for certain. Notice that by definition, *a possible world retains all nodes of $\mathcal{G}$*.

For a deterministic graph $G = (V, E_G)$ where $E_G \subseteq E$, the probability that G is observed as a possible world of $\mathcal{G}$ can be calculated as follows.

$$\Pr[G|\mathcal{G}] =_{\text{def}} \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)). \quad (1)$$

We use $G \sqsubseteq \mathcal{G}$ to indicate that G is a possible world of $\mathcal{G}$.

EXAMPLE 1. *Consider the probabilistic graph $\mathcal{G}$ in Figure 1(a) and the possible world $G_1 \sqsubseteq \mathcal{G}$ in Figure 1(b). By applying Eq.* (1)*, we have* $\Pr[G_1|\mathcal{G}] = 0.7^4 \times 0.5^6 = 0.0037515625$.

3.1 Probabilistic Trusses

Local (k,γ)-truss. Next, we extend the concept of k-truss to probabilistic graphs. Let $\mathsf{sup}_{\mathcal{G}}(e)$ denote the support of $e = (u,v)$ in $\mathcal{G}$, and let $N(u) \subset V$ denote the set of *structural neighbors* of u in $\mathcal{G}$. Here, by a structural neighbor of u, we mean any node v that is adjacent to u according to the graph structure of $\mathcal{G}$, ignoring any probabilities. Thus, $N(u)$ is a deterministic set. Clearly, $\mathsf{sup}_{\mathcal{G}}(e)$ is a random variable which can take on any integer value from zero to $k_e =_{\text{def}} |N(u) \cap N(v)|$, the maximum possible support of e in any possible world of $\mathcal{G}$.

By definition of possible worlds, given any $t \in [0, k_e]$, the probability that $\mathsf{sup}_{\mathcal{G}}(e) \geq t$ is the sum of the probability mass of all possible worlds $G \sqsubseteq \mathcal{G}$ such that the (deterministic) support of e in G is no less than t. Mathematically,

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) \geq t] = \sum\nolimits_{G \sqsubseteq \mathcal{G}} \Pr[G|\mathcal{G}] \cdot \mathbf{I}(\mathsf{sup}_G(e) \geq t), \quad (2)$$

where $\mathbf{I}(\mathsf{sup}_G(e) \geq t)$ is an indicator function which takes on 1 if $\mathsf{sup}_G(e) \geq t$, and 0 otherwise. If $e \notin E_G$, $\mathbf{I}(\mathsf{sup}_G(e) \geq t) =_{\text{def}} 0$.

Intuitively, a subgraph $\mathcal{H}$ of $\mathcal{G}$ [1] can be regarded as a cohesive subgraph of $\mathcal{G}$ if the support of every edge in $\mathcal{H}$ is no less than

[1]Note that $\mathcal{H}$ is still a probabilistic graph.

a threshold *with high probability*. Following this intuition, we define *local* (k,γ)*-truss* as follows, where by a connected subgraph $\mathcal{H} \subseteq \mathcal{G}$, we mean any subgraph $\mathcal{H}$ which is connected as a graph, ignoring probabilities.

DEFINITION 2 (LOCAL (k,γ)-TRUSS). *Let $\mathcal{G} = (V,E,p)$ be a probabilistic graph and $\mathcal{H} \subseteq \mathcal{G}$ any* connected *subgraph. Given a threshold $\gamma \in [0,1]$, and an integer $k \geq 2$, we say $\mathcal{H}$ is a* local (k,γ)-truss*, iff for every $e \in E_{\mathcal{H}}$, $\Pr[\mathsf{sup}_{\mathcal{H}}(e) \geq k-2] \geq \gamma$, i.e., for each edge, the probability that it has a support no less than $(k-2)$ in $\mathcal{H}$ is at least γ.*

The name "local" reflects that the support of each edge is evaluated individually, and there is no graph-wise global constraint that $\mathcal{H}$ must satisfy. The requirement that $\mathcal{H}$ is *connected* is natural, since it is possible to have two totally-separated connected components while still all edges have high support with high probability. For instance, as mentioned in Section 1, the subgraph $\mathcal{H}_1$ shown in Figure 2(a) is a local (k,γ)-truss where $k=4$ and $\gamma = 0.125$.

We are interested in finding all local (k,γ)-trusses that are *maximal*, i.e., those that are not proper subgraphs of any other local (k,γ)-truss. We formulate this problem as *Local Probabilistic Truss Decomposition*, dubbed LOCALDECOMP.

PROBLEM 1 (LOCALDECOMP). *Given a probabilistic graph $\mathcal{G} = (V,E,p)$ and a threshold $\gamma \in [0,1]$, find all* maximal local (k,γ)-trusses *of $\mathcal{G}$, for all $2 \leq k \leq k_{\max}$, where $k_{\max}$ is the maximum support of any edge in $\mathcal{G}$.*

Global (k,γ)-truss. As noted earlier, a local (k,γ)-truss does not have to satisfy any graph-wise global constraints: specifically, while Definition 2 ensures that for each edge, there are enough possible worlds where its support is $\geq (k-2)$, it does not ensure that the $(k-2)$ supporting triangles of different edges in $\mathcal{H}$ are present *together* in a fraction $\geq \gamma$ of the possible worlds. To mitigate this, we propose a stronger definition of a (k,γ)-truss which naturally incorporates a global constraint, thus strengthening the cohesiveness of a probabilistic truss. In the definition below, notice that all nodes of $\mathcal{H}$ are required to be connected in any connected possible world H of $\mathcal{H}$. Formally,

DEFINITION 3 (GLOBAL (k,γ)-TRUSS). *Let $\mathcal{G} = (V,E,p)$ be a probabilistic graph. Given a threshold $\gamma \in [0,1]$, and an integer $k \geq 2$, a connected subgraph $\mathcal{H} = (V_{\mathcal{H}}, E_{\mathcal{H}})$ of $\mathcal{G}$ is a* global (k,γ)-truss*, iff for each edge $e \in E_{\mathcal{H}}$,*

$$\alpha_k(\mathcal{H},e) =_{\text{def}} \sum_{H \sqsubseteq \mathcal{H}} \Pr[H|\mathcal{H}] \cdot \mathbf{I}(H,k,e) \geq \gamma, \qquad (3)$$

where $\mathbf{I}(H,k,e)$ is an indicator function taking on 1 if the possible world H (thus $V_H = V_{\mathcal{H}}$) is a connected, deterministic k-truss containing e, and 0 otherwise.

Every global (k,γ)-truss is also a local (k,γ)-truss, as we show below. Hence, global (k,γ)-truss is a stronger and stricter notion, which intuitively corresponds to more cohesive (probabilistic) subgraphs.

LEMMA 1. *Let $\mathcal{G} = (V,E,p)$ be a probabilistic graph and $\mathcal{H} \subseteq \mathcal{G}$ be a global (k,γ)-truss. Then $\mathcal{H}$ is also a local (k,γ)-truss, for the same k and γ.*

PROOF. Every connected k-truss containing an edge e clearly has at least $(k-2)$ triangles containing that edge. The lemma follows from this and the fact that $\mathcal{H}$ is connected. □

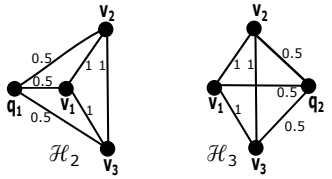


Figure 3: Two global (k,γ)-trusses of $\mathcal{G}$ in Figure 1. Here, $k=4$ and $\gamma = 0.125$.

EXAMPLE 2. Figure 3 depicts two global $(4, 0.125)$-trusses $\mathcal{H}_2, \mathcal{H}_3 \subseteq \mathcal{G}$, where $\mathcal{G}$ is shown in Figure 1. In both cases, the only supporting possible world is the one where all edges exist, whose existence probability is $(0.5)^3 \cdot 1^3 = 0.125$. Moreover, both $\mathcal{H}_2$ and $\mathcal{H}_3$ are maximal global $(4, 0.125)$-trusses and there exists no other global $(4, 0.125)$-truss in $\mathcal{G}$. To compare global and local (k,γ)-trusses, first, we note that $\mathcal{H}_2$ and $\mathcal{H}_3$ are also local $(4, 0.125)$-trusses, but the local $(4, 0.125)$-truss $\mathcal{H}_1$ in Figure 2(a) is not a global one. In fact, $\mathcal{H}_1$ is a global $(4, 0.5^6)$-truss, because the only possible world of $\mathcal{H}_1$ that is a connected k-truss containing all nodes in $\mathcal{H}_1$ is the one shown in Figure 2(b), and its existence probability is 0.5^6.

We are interested in the problem of finding all maximal global (k,γ)-trusses of a given probabilistic graph. Formally:

PROBLEM 2 (GLOBALDECOMP). *Given a probabilistic graph $\mathcal{G} = (V,E,p)$ and a threshold $\gamma \in [0,1]$, find all* maximal global (k,γ)-trusses *of $\mathcal{G}$, for all $2 \leq k \leq k_{\max}$, where $k_{\max}$ is the maximum support of any edge in $\mathcal{G}$.*

3.2 Hardness Results

As we shall see in Section 4, LOCALDECOMP can be solved in polynomial time. However, GLOBALDECOMP is generally intractable, since even computing $\alpha_k(\mathcal{H},e)$ is #P-hard.

THEOREM 1. *Given a probabilistic graph $\mathcal{H}$ and an edge e in $\mathcal{H}$, computing $\alpha_k(\mathcal{H},e)$ as in Eq.* (3) *is #P-hard.*

PROOF. The reduction is from the NETWORK RELIABILITY problem [16,32]. For a probabilistic graph $\mathcal{G}$, its network reliability, i.e., the probability that $\mathcal{G}$ is connected, is defined as

$$\mathsf{conn}(\mathcal{G}) = \sum\nolimits_{G \sqsubseteq \mathcal{G}} \Pr[G|\mathcal{G}] \cdot \mathbf{C}(G), \qquad (4)$$

where G is a possible world of $\mathcal{G}$, and $\mathbf{C}(G)$ is an indicator function taking on 1 if G is connected, and 0 otherwise. It is known that computing $\mathsf{conn}(\mathcal{G})$ is #P-hard [16].

Given an instance of NETWORK RELIABILITY with $\mathcal{G}$, we construct an instance of the problem of computing $\alpha_k(\cdot,\cdot)$ as follows. Let v be an arbitrary vertex in $\mathcal{G}$. We create a dummy node w and an edge (w,v) with $p(w,v) = 1$. Let the resulting graph be $\mathcal{H}$.

We show the following claim: a possible world $G = (V_G, E_G) \sqsubseteq \mathcal{G}$ is connected iff $H = (V_G \cup \{w\}, E_G \cup \{(w,v)\}) \sqsubseteq \mathcal{H}$ is a connected 2-truss that contains (w,v). The "if" direction is trivial. Now consider "only if". Since G is connected, then there exists a path between v and any other node in V_G. Also, since the edge (w,v) is in H, then there is also a path connecting w with any other node in V_G. Thus H is connected, contains (w,v) by construction, and is a 2-truss by definition.

Due to the one-to-one correspondence established between the possible worlds of $\mathcal{G}$ and $\mathcal{H}$ in the above claim, and the fact that $\Pr[G|\mathcal{G}] = \Pr[H|\mathcal{H}]$ as $p(w,v) = 1$, we can see that $\mathsf{conn}(\mathcal{G}) = \alpha_k(H,(w,v))$. The theorem follows. □

GLOBALDECOMP is as least as intractable as evaluating $\alpha_k(\mathcal{H}, k, e)$: since γ can be any real number in $[0, 1]$, checking if $\alpha_k(\mathcal{H}, e) \geq \gamma$ requires that $\alpha_k(\mathcal{H}, e)$ be computed to arbitrary accuracy. In addition, we show that the number of solutions to GLOBALDECOMP can in fact be exponential in graph size, adding another major computational challenge.

LEMMA 2. *There are probabilistic graphs $\mathcal{G}$ and parameter γ, such that the number of maximal global (k, γ)-trusses is exponential in $|V_\mathcal{G}|$. This is true even for fixed k.*

PROOF. Please refer to Appendix. □

Both Theorem 1 and Lemma 2 indicate the intrinsic hardness of GLOBALDECOMP. Although GLOBALDECOMP may have an exponential number of answers, it is still important to study for the following reasons. First, compared with the local (k, γ)-truss, global (k, γ)-truss is a more stringent and holistic definition, which ensures that the *entire* subgraph has a certain probability to exist as a k-truss. In critical applications such as identifying functional modules from protein-protein interaction networks, modules with a high probability of existence as a whole are more useful for making sure that any medical or clinical assessments are correct. In addition, as an analogy, consider the seminal Frequent Item-set (Pattern) Mining problem, where the number of satisfying item-sets is exponential in the number of items. However, this is still an extremely useful problem with wide applications. Besides, the number of frequent item-sets can be controlled by adjusting parameters such as support and confidence. In our global (k, γ)-truss decomposition problem, we can control the size of output by fine-tuning the value of k and γ. For a reasonably large k and γ, the number of satisfying subgraphs is unlikely to be enormous in real applications. This claim is experimentally verified on real datasets in Section 6.3. Finally, in Section 5.1, we make use of Hoeffding's inequality to develop a Monte Carlo sampling method, which lays the foundations of an approximation algorithm for finding maximal global (k, γ)-trusses.

4. LOCAL PROBABILISTIC TRUSS DECOMPOSITION

We now describe an algorithm for solving LOCALDECOMP. The general idea is based on the iterative edge removal process for truss decomposition in deterministic graphs [9, 33]. However, there are significant computational challenges brought about by probabilistic graphs. For example, we must combat the complications introduced by the *combinatorial nature* of the triangles in which an edge may participate. We will first give an outline of the algorithm and then describe two challenging tasks, namely computing and updating edge support probabilities, in detail.

4.1 Algorithmic Framework

For an edge $e = (u, v) \in E_\mathcal{G}$, let $\rho_\mathcal{G}(e, t)$ denote the probability that the support of edge e is at least t in $\mathcal{G}$, i.e., $\sigma_\mathcal{G}(e, t) =_{\text{def}} \Pr[\mathsf{sup}_\mathcal{G}(e) \geq t]$. Furthermore, let $\sigma_\mathcal{G}(e) =_{\text{def}} [\sigma_\mathcal{G}(e, 1), \sigma_\mathcal{G}(e, 2), \ldots, \sigma_\mathcal{G}(e, k_e)]$ be the vector of support probabilities of e for all possible t from 1 to k_e. We refer to $\sigma_\mathcal{G}(e)$ as the *edge support probability vector*. When it is clear from the context, we drop the subscript $\mathcal{G}$.

Note that the edge support probabilities in $\sigma(e)$ are well-defined only when e actually exists. Hence, in what follows, we assume that e does exist for the purpose of computing $\sigma(e)$. The "true" edge support probabilities can be easily derived by multiplying $\sigma(e)$ with $p(e)$ element-wise.

Algorithm 1 Local (k, γ)-truss decomposition

Require: $\mathcal{G} = (V, E, p)$; $\gamma \in [0, 1]$
Ensure: trussness score $\tau(e)$ of each edge $e \in E$
1: **for all** $e \in E$ **do**
2: compute $\sigma(e)$ using Algorithm 2
3: **for** $k \leftarrow 2$ **to** n **do**
4: **while** $\exists e = (u, v)$ such that $\sigma(e, k-1)p(e) < \gamma$ **and** $\sigma(e, k-2)p(e) \geq \gamma$ **do**
5: $\tau(e) \leftarrow k$
6: remove e from $\mathcal{G}$
7: **for** $w \in N(u) \cap N(v)$ **do**
8: update $\sigma_{(w,u)}$ and $\sigma_{(w,v)}$ {*cf.* Section 4.3}
9: **if** $\mathcal{G}$ is empty **then**
10: **break**

Monotonicity of $\sigma(e)$. In designing the decomposition algorithm, we make use of the fact that the edge support probabilities in $\sigma(e)$ are monotonically non-increasing. That is, for all $e \in E$, we have $\sigma(e, 1) \geq \sigma(e, 2) \geq \ldots \geq \sigma(e, k_e)$. This holds true by definition, because

$$\sigma(e, k) = \Pr[\mathsf{sup}_\mathcal{G}(e) = k] + \sigma(e, k+1) \geq \sigma(e, k+1).$$

Overview. Algorithm 1 presents the pseudo-code for our local (k, γ)-truss decomposition procedure. We begin by computing the edge support probability vector $\sigma(e)$ for all $e \in E$ (lines 1-2). Then, the algorithm iteratively finds all local (k, γ)-trusses starting from $k = 2$. For any particular k, if there exists an edge e such that $\sigma(e, k-1)p(e) < \gamma$ and $\sigma(e, k-2)p(e) \geq \gamma$, then e belongs to a local (k, γ)-truss, but does not belong to any local $(k+1, \gamma)$-truss. Then e will be assigned a trussness score of k, denoted by $\tau(e)$, and removed from the graph (lines 4-6). After the removal of e, all triangles in which this edge participates shall also be removed, and thus the support of the other two edges in each such triangle shall be updated (lines 7-8). This iterative process goes on until all edges are removed from the graph (lines 9-10). The value of k at which we stop is thus $k_{\max}$.

After obtaining the trussness score of all edges, we can easily construct all maximal local (k, γ)-trusses, for a particular k, by piecing together all edges e with $\tau(e) \geq k$. Note that by Definition 2, the output trusses for LOCALDECOMP must be connected subgraphs of $\mathcal{G}$. Thus, after Algorithm 1, we perform a post-processing step to extract all connected components. We have the following:

THEOREM 2. *Algorithm 1, with post-processing on connectivity, finds all and only maximal local (k, γ)-trusses of $\mathcal{G}$, for all $2 \leq k \leq k_{\max}$.*

PROOF. Please refer to Appendix. □

Although computing and updating edge support in deterministic graphs can be considered straightforward, it is far from trivial to compute and update edge support probabilities in probabilistic graphs. Next we give a dynamic programming algorithm for this task. We defer time complexity analysis of Algorithm 1 to the end of this section.

4.2 Computing Edge Support Probabilities

By definition, $\sigma(e, t) = \Pr[\mathsf{sup}_\mathcal{G}(e) \geq t]$, and thus

$$\sigma(e, t) = 1 - \sum_{i=0}^{t-1} \Pr[\mathsf{sup}_\mathcal{G}(e) = i]$$

Therefore, it is helpful for us to determine the formulas for computing $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$, given any i.

Zero triangles. Consider the special case of edge e having zero support, i.e., $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = 0]$. Two cases arise: (i). e does not even exist, which happens with probability $1 - p(e)$; (ii). e exists but does not participate in any triangles, which happens with probability $p(e) \cdot \prod_{w \in N(u) \cap N(v)} (1 - p(w,u)p(w,v))$. Thus,

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) = 0] = (1 - p(e)) + p(e) \prod_{w \in N(u) \cap N(v)} (1 - p(w,u)p(w,v)). \quad (5)$$

The case of e not existing is not interesting, but for our dynamic programming algorithm in Section 4.2.1, we need to consider case (ii) above. Thus, the probability of zero support reduces to $\prod_{w \in (N(u) \cap N(v))} (1 - p(w,v)p(w,u))$ *under the assumption that* e *exists.*

Multiple triangles. Now, we consider the probability that edge e participates in one or more triangles. The most naive approach is to consider, for each $1 \leq i \leq k_e$, all sets of i triangles, and sum up all the probabilities as a final result.

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i] = \sum_{W \subseteq N(u) \cap N(v), |W| = i} \prod_{w \in W} p(w,u)p(w,v) \prod_{w \in N(u) \cap N(v) \setminus W} (1 - p(w,u)p(w,v)).$$

However, enumerating all subsets of $N(u) \cap N(v)$ of size i can be prohibitively expensive. The time complexity is $\binom{n}{i}$ where $|N(u) \cap N(v)| \in O(n)$, and thus is potentially exponential in n. To avoid this exponential blow-up, we devise a dynamic programming algorithm to compute $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$, $1 \leq i \leq k_e$, in polynomial time, as described next.

4.2.1 *Dynamic Programming for* $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$

Our purpose is to identify the structure of the problem so that we can derive a recursive formula for dynamic programming. Consider an edge $e = (u,v)$ and a fixed common neighbor $w \in N(u) \cap N(v)$. Note that if e participates in i triangles in $\mathcal{G}$, then either (i). $\triangle_{uvw}$ exists and e participates in $i-1$ other triangles in $\mathcal{G}$ excluding $\triangle_{uvw}$, or (ii). $\triangle_{uvw}$ does not exist and e participates in i other triangles. Thus, $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$ can be computed as a linear combination of the probabilities that e has $i-1$ or i triangles in $\mathcal{G}$ excluding $\triangle_{uvw}$. According to this rule, we can take advantage of the results of subproblems $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = j]$, $0 \leq j \leq i-1$, to calculate $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$.

The recursion. Given a probabilistic graph $\mathcal{G} = (V, E, p)$ and an edge $e = (u,v) \in E$, we denote by $W =_{\text{def}} N(u) \cap N(v) = \{w_1, ..., w_{k_e}\}$, the set of all common neighbors of u and v in $\mathcal{G}$, ordered arbitrarily. Given a subset $W_\ell = \{w_1, ..., w_\ell\} \subseteq W$, where $1 \leq \ell \leq k_e$, we denote by $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i | W_\ell]$ the probability that e participates in i triangles with common neighbors from W_ℓ. For clarity of exposition, we use $f(i,\ell)$ as a shorthand for $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i | W_\ell]$. Now, for any two consecutive subsets $W_{\ell-1}$ and W_ℓ, the following recursive formula holds:

$$f(i,\ell) = p(w_\ell,u)p(w_\ell,v)f(i-1,\ell-1) + (1 - p(w_\ell,u)p(w_\ell,v))f(i,\ell-1), \quad (6)$$

where $-1 \leq i \leq k_e$ and $0 \leq \ell \leq k_e$. Note that $i = -1$ and $\ell = 0$ are "dummy" cases to set up the base cases:

Algorithm 2 Dynamic programming for computing $\sigma(e)$

Require: $\mathcal{G} = (V, E, p)$, an edge $e = (u,v) \in E$.
Ensure: $\sigma(e) = [\sigma(e,1), \sigma(e,2), \ldots, \sigma(e,k_e)]$
1: $f(0,0) \leftarrow 1$
2: $f(-1,\ell) \leftarrow 0$, for all $0 \leq \ell \leq k_e$
3: **for** $\ell \leftarrow 0$ **to** k_e **do**
4: **for** $i \leftarrow \ell + 1$ **to** k_e **do**
5: $f(i,\ell) \leftarrow 0$
6: **for** $i \leftarrow 0$ **to** k_e **do**
7: **for** $\ell \leftarrow 1$ **to** k_e **do**
8: $f(i,\ell) \leftarrow p(w_\ell,u)p(w_\ell,v)f(i-1,\ell-1) + (1 - p(w_\ell,u)p(w_\ell,v))f(i,\ell-1)$
9: $\sigma(e,0) \leftarrow 1$
10: **for** $t \leftarrow 1$ **to** k_e **do**
11: $\sigma(e,t) \leftarrow \sigma(e,t-1) - f(t,k_e)$

- $f(0,0) = 1$;
- $f(-1,\ell) = 0$, for all $0 \leq \ell \leq k_e$;
- $f(i,\ell) = 0$, whenever $i > \ell$.

For any i, the value $f(i,k_e)$ is the desired edge support probability $\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i]$.

Lines 1-8 of Algorithm 2 describe the dynamic programming procedure. After these steps, we can get the desired edge support vector $\sigma(e)$ via a single linear scan (lines 9-11), thanks to the following equations:

$$\begin{aligned}\sigma(e,t) &= \sigma(e,t-1) - \Pr[\mathsf{sup}_{\mathcal{G}}(e) = t-1] \\ &= \sigma(e,t-1) - f(t-1,k_e). \end{aligned} \quad (7)$$

Time complexity of Algorithm 2. For each edge $e = (u,v)$, Algorithm 2 runs in $O((\min\{d(u),d(v)\})^2)$ time, where $d(u)$ is the degree of u in $\mathcal{G}$. The entire dynamic programming procedure (lines 1-8) takes $O(k_e^2)$ time, where $k_e = |N(u) \cap N(v)| \in O(\min\{d(u),d(v)\})$. Calculating $\sigma(e)$ (lines 9-11) takes $O(k_e)$ time. As a result, the total time complexity is $O((\min\{d(u),d(v)\})^2)$.

4.3 Updating Edge Support Probabilities

We now describe how to update $\sigma(e)$ for edge $e = (u,v)$ if the triangle $\triangle_{uvw}$ is deleted from $\mathcal{G}$ where $w \in N(u) \cap N(v)$, owing to the removal of $e_1 = (u,w)$ or $e_2 = (v,w)$. Recall that the update step is fundamental and crucial in our (k,γ)-truss decomposition algorithm (line 8 of Algorithm 1). A naive approach is to compute $\sigma(e)$ from scratch by Algorithm 2 whenever the update is needed, but it will again incur $O((\min\{d(u),d(v)\})^2)$ overhead.

Assume, without loss of generality, that $e_1 = (u,w)$ was removed from $\mathcal{G}$ (the case of $e_2 = (v,w)$ being removed can be similarly analyzed). The following formula holds:

$$\begin{aligned}\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i | W] = (1 - p(e_1)p(e_2)) \Pr[\mathsf{sup}_{\mathcal{G}}(e) = i | W \setminus \{w\}] \\ + p(e_1)p(e_2) \Pr[\mathsf{sup}_{\mathcal{G}}(e) = i-1 | W \setminus \{w\}]\end{aligned}$$

Re-arranging terms, we have:

$$\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i | W \setminus \{w\}] = \frac{\Pr[\mathsf{sup}_{\mathcal{G}}(e) = i | W] - p(e_1)p(e_2) \Pr[\mathsf{sup}_{\mathcal{G}}(e) = i-1 | W \setminus \{w\}]}{1 - p(e_1)p(e_2)}$$

Thus, to update $\sigma(e)$, we only need to update $f(t,k_e)$ using the following rule:

$$f^{new}(i,k_e) = \frac{f^{old}(i,k_e) - p(e_1)p(e_2)f^{new}(i-1,k_e)}{1 - p(e_1)p(e_2)}, \quad (8)$$

where the superscripts old and new correspond to before and after the edge e_1 is removed. Then, $\sigma(e)$ can be efficiently updated via a linear scan using f^{new} in $O(k_e)$ time as per Eq. 7. Furthermore, we can see that the update rule in Eq. 8 applies regardless of which edge, e_1 or e_2, is the one being removed.

Time complexity of Algorithm 1. The cost of computing $\sigma(e)$ for all $e \in E$ (lines 1-2) is $O(\sum_{(u,v)\in E}(\min\{d(u),\ d(v)\})^2) \subseteq O(d_{\max}\sum_{(u,v)\in E}\min\{d(u), d(v)\}) \subseteq O(d_{\max}\rho|E|)$, where $d_{\max}$ is the maximum degree and ρ is the arboricity of graph $\mathcal{G}$, i.e., the minimum number of spanning forests needed to cover all edges of $\mathcal{G}$. Notice that $\rho \leq \min\{d_{max}, \sqrt{|E|}\}$ [8]. The entire decomposition (lines 3-10) eventually will remove all edges from $\mathcal{G}$, and for each removal of an edge (u, v), we need to update $\sigma((v, w))$ and $\sigma((u, w))$ for all $w \in N(u) \cap N(v)$. For each removal, the number of updates is at most $2|N(u) \cap N(v)| \in O(\min\{d(u), d(v)\})$. Since a single update can be done in $O(k_e) \subseteq O(\min\{d(u), d(v)\})$ time as discussed, the total cost of doing updates pertaining to one edge removal is $O(\sum_{(u,v)\in E}(\min\{d(u), d(v)\})^2) \subseteq O(d_{\max}\rho|E|)$. In addition, edge selection (line 4) and edge removal (line 6) both can be done in $O(1)$ time by a variant of bin sort table [33]. Thus, the total time complexity of Algorithm 1 is $O(d_{\max}\rho|E|)$.

Space complexity of Algorithm 1. For each $e = (u, v) \in E$, we maintain a 2-dimensional table for dynamic programming in theory. In the implementation, for computing each $f(i, \ell)$, we only need to keep two arrays as $f(i-1, *)$ and $f(i, *)$, which consumes $O(\min\{d(v), d(u)\})$ space. Moreover, these two array can be released after obtaining $\sigma(e)$. In addition, the edge support vector $\sigma(e)$ only uses $O(\min\{d(v), d(u)\})$ space. The whole graph itself takes $O(|E|)$ space. Thus, the total space complexity is $O(|E| + \sum_{(u,v)\in E}\min\{d(v), d(u)\}) \subseteq O(\rho|E|)$.

5. GLOBAL PROBABILISTIC TRUSS DECOMPOSITION

5.1 Monte Carlo Sampling

First, we show how to use Monte Carlo sampling to estimate $\alpha_k(\mathcal{H}, e)$ (Eq. (3)) to circumvent the #P-hardness of its exact computation (Theorem 1). We apply a special case of the well-known Hoeffding's inequality [14], stated as follow. It specifies the minimum number of possible world samples required to get a bounded error for estimating $\alpha_k(\mathcal{H}, e)$.

PROPOSITION 1. *Let $X_1, X_2, \ldots, X_N$ be independent random variables such that $\Pr[0 \leq X_i \leq 1] = 1$. Let $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$ and $\mu = \mathbb{E}[\bar{X}]$. Then,*

$$\Pr[|\bar{X} - \mu]| \geq \epsilon] \leq 2\exp\left(-2N\epsilon^2\right). \quad (9)$$

From Eq. (9), we have that for any $\delta \in (0, 1]$ and any $\epsilon \in (0, 1]$, $\Pr[|\bar{X} - \mu| \geq \epsilon] \leq \delta$, as long as $N \geq \frac{1}{2\epsilon^2}\ln(\frac{2}{\delta})$.

Given an error upper bound ϵ and a probability guarantee δ, a naive strategy would be to sample $N = \lceil \frac{1}{2\epsilon^2}\ln(\frac{2}{\delta})\rceil$ possible worlds of $\mathcal{H}$, for every subgraph $\mathcal{H} \subseteq \mathcal{G}$, and estimate $\alpha_k(\mathcal{H}, e)$ using the samples. This would be prohibitively expensive. Instead, our overall strategy is to sample $N = \lceil \frac{1}{2\epsilon^2}\ln(\frac{2}{\delta})\rceil$ possible worlds of $\mathcal{G}$, denoted $D_{\mathcal{G}} =_{\text{def}} \{G_i\}_{i=1}^{N}$. As the existence of each edge is independent, the resulting possible worlds are pairwise independent. Consider any subgraph $\mathcal{H} \subseteq \mathcal{G}$, we then obtain the N possible worlds of $\mathcal{H}$ by "projecting" each G_i to $\mathcal{H}$: $G_i \downarrow_{\mathcal{H}} =_{\text{def}} (V_{\mathcal{H}}, E_{G_i} \cap E_{\mathcal{H}})$. We also denote the projected world by H_i for simplicity. An estimation of $\alpha_k(\mathcal{H}, e)$ is then obtained via Monte Carlo sampling:

$$\widehat{\alpha}_k(\mathcal{H}, e) = \sum\nolimits_{i=1}^{N} \mathbf{I}(H_i, e, k)/N, \quad (10)$$

where the indicator function $\mathbf{I}(H_i, e, k)$ is defined in Definition 3. Notice that by Proposition 1, an estimate of $\alpha_k(\mathcal{H}, e)$ obtained by sampling possible worlds of $\mathcal{H}$ is an unbiased estimate. Later, we shall show that the estimate $\widehat{\alpha}_k(\mathcal{H}, e)$ obtained by sampling (projected) possible worlds of $\mathcal{G}$ (as in Eq. (10)) gives the same result as the estimate obtained by sampling possible worlds of $\mathcal{H}$.

We say that $\mathcal{H}$ is an (ϵ, δ)-*approximate global* (k, γ)-*truss* if for any $e \in E_{\mathcal{H}}$, $\widehat{\alpha}_k(\mathcal{H}, e) \geq \gamma$. The following theorem establishes the theoretical foundation for our strategy that first samples N possible worlds of $\mathcal{G}$ and then projects them on to various subgraphs $\mathcal{H}$ throughout the decomposition process.

THEOREM 3. *Given a probabilistic graph $\mathcal{G}$ and parameters $\epsilon \in (0, 1]$ and $\delta \in (0, 1]$, let $D_{\mathcal{G}} = \{G_1, ..., G_N\}$ be N independently sampled possible worlds of $\mathcal{G}$, where $N \geq \frac{1}{2\varepsilon^2}\ln(\frac{2}{\delta})$. Then, for any given $\mathcal{H} \subseteq \mathcal{G}$ and for any given $e \in E_{\mathcal{H}}$, we have*

$$\Pr[|\widehat{\alpha}_k(\mathcal{H}, e) - \alpha_k(\mathcal{H}, e)| \geq \varepsilon] \leq \delta,$$

where $\widehat{\alpha}_k(\mathcal{H}, e)$ is computed according to Eq. (10) using the projected possible worlds $D_{\mathcal{H}} =_{\text{def}} \{G_i \downarrow_{\mathcal{H}}\}_{i=1}^{N}$.

This is a non-trivial result, and to prove it, we first introduce a few notations and then give a useful lemma. Consider any subgraph $\mathcal{H} \subseteq \mathcal{G}$ and an arbitrary $H \sqsubseteq \mathcal{H}$. Let $X(H)$ denote the set of possible worlds of $\mathcal{G}$ whose projection to $\mathcal{H}$ results in H, i.e., $X(H) =_{\text{def}} \{G \sqsubseteq \mathcal{G} : G \downarrow_{\mathcal{H}} = H\}$.

LEMMA 3. *Given a probabilistic graph $\mathcal{G}$, a subgraph $\mathcal{H} \subseteq \mathcal{G}$, and any possible world $H \sqsubseteq \mathcal{H}$, we have*

$$\Pr[H|\mathcal{H}] = \sum\nolimits_{G\in X(H)} \Pr[G|\mathcal{G}].$$

PROOF. Please refer to Appendix. □

COROLLARY 1. *Given any $\mathcal{H} \subseteq \mathcal{G}$, $\forall e \in E_{\mathcal{H}}$, we have*

$$\alpha_k(\mathcal{H}, e) = \sum_{G\sqsubseteq\mathcal{G}} \Pr[G|\mathcal{G}] \cdot \mathbf{I}(G \downarrow_{\mathcal{H}}, k, e).$$

PROOF. Please refer to Appendix. □

Note that Theorem 3 follows upon applying Proposition 1 to the result of Corollary 1. In particular, note that the probability of sampling a given possible world $H \sqsubseteq \mathcal{H}$ is the same as the probability of sampling some possible world $G \sqsubseteq \mathcal{G}$ such that G projects to H. Thus, we only need to sample N possible worlds of $\mathcal{G}$ which can be used to estimate $\alpha_k(\mathcal{H}, e)$ for any subgraph $\mathcal{H} \subseteq \mathcal{G}$ and edge $e \in E_{\mathcal{H}}$.

In what follows, we describe two algorithms for finding all *maximal* (ϵ, δ)-approximate global (k, γ)-trusses for a given γ and all possible k. For simplicity, we refer to such trusses as "satisfying trusses" in our algorithm descriptions.

5.2 Theoretical Analysis on Global (k, γ)-Truss

The search space for maximal global (k, γ)-trusses is huge. This is exacerbated by two challenging factors: (i) even computing the probability that a given subgraph $\mathcal{H} \subseteq \mathcal{G}$ contains a k-truss that connects all nodes of $\mathcal{H}$ is intractable (Theorem 1) and (ii) the number of maximal global (k, γ)-trusses can be exponential. In the previous section, we tackled the first factor using sampling. As for the second, a useful property that is often employed in efficient

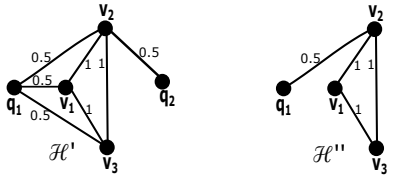


Figure 4: Two non-global (4, 0.125)-truss $\mathcal{G}'$ and $\mathcal{G}''$

discovery of frequent subgraphs is *monotonicity*. In our context, this reduces to asking, given graphs $\mathcal{H}_1 \subset \mathcal{H}_2$, whether one of them being a global (k, γ)-truss has any implications for the other graph. Unfortunately, this is not the case here.

EXAMPLE 3. In Figure 4, neither $\mathcal{H}'$ nor $\mathcal{H}''$ is a global (4, 0.125)-truss. For $\mathcal{H}'$, none of its possible worlds is a connected 4-truss containing node q_2 that is incident with just one edge. The same reasoning can be applied to $\mathcal{H}''$. Recall that as shown in Figure 3, $\mathcal{H}_2$ is a global (4, 0.125)-truss. Since $\mathcal{H}'' \subset \mathcal{H}_2 \subset \mathcal{H}'$, we can see that any monotonicity-driven pruning of search space does not work.

On the other hand, global (k, γ)-trusses do satisfy a monotonicity property w.r.t. k: every $(k+1, \gamma)$-truss is also a (k, γ)-truss. This follows from the fact that every $(k+1)$-truss is also a k-truss, although it is not guaranteed to be maximal. This observation can be exploited to develop a search framework and a bottom-up exploration strategy. The idea of the search framework is to start with a small k and find maximal global (k, γ)-trusses. Then we remove edges from them to find global trusses with larger k. In the bottom-up exploration strategy, we start with a single edge and then expand it by adding common neighbors of its endpoints and recursively following up until we obtain a maximal global (k, γ)-truss.

One of the challenges in case of maximal global trusses is that their number can be exponential (see Lemma 2 and the appendix). The reason for this is that different maximal global trusses can overlap. In case of maximal local trusses, the edge support probability is determined independently for each edge, so maximal local trusses (for any given k) are always disjoint. This is the reason they can be found efficiently.

5.3 Decomposition Algorithms

Overview. The decomposition process first samples N possible worlds from $\mathcal{G}$ (Theorem 3). Then, starting from $k = 2$, we apply local (k, γ)-truss decomposition (Algorithm 1) to construct a candidate graph, in which we search for maximal (ϵ, δ)-approximate global (k, γ)-trusses. The candidate graph varies with k, and higher k corresponds to smaller candidate graph (details to follow), thanks to the monotonicity property w.r.t. k discussed in the previous section. Since exhaustive search is prohibitive, we propose two more efficient search algorithms. The first one is a top-down, exact approach (Algorithm 4) that finds *all* satisfying trusses; The second one is a bottom-up heuristic (Algorithm 5) that is not guaranteed to find all satisfying trusses, but is significantly more efficient.

Backbone algorithm. Algorithm 3 is the backbone decomposition algorithm. It first samples $N = \lceil \frac{1}{2\epsilon^2} \ln(\frac{2}{\delta}) \rceil$ possible worlds from $\mathcal{G}$ to form $D_{\mathcal{G}} = \{G_i\}_{i=1}^{N}$ (lines 1–2), which will be used for estimating edge supports. Next, we apply Algorithm 1 to obtain all maximal local (k, γ)-trusses in $\mathcal{G}$ for all possible k (line 3).

The main decomposition procedure iterates over all possible k, starting from 2. In each iteration, we first generate the candidate set $\mathcal{C}_k$ that contains all edges that *may* be present in an (ϵ, δ)-approximate global (k, γ)-truss (line 5). In order to prune this set effectively, notice the following. An edge cannot belong to a global

Algorithm 3 Global (k, γ)-truss decomposition

Require: $\mathcal{G} = (V, E, p)$; γ, ϵ, δ
Ensure: All (ϵ, δ)-approximate global (k, γ)-truss $\mathcal{H}$
1: $N \leftarrow \lceil \frac{1}{2\epsilon^2} \ln(\frac{2}{\delta}) \rceil$
2: $D_{\mathcal{G}} \leftarrow \{G_1, \ldots, G_N\}$, sampled independently from $\mathcal{G}$
3: Apply Algorithm 1 on $\mathcal{G}$ for local (k, γ)-truss decomposition.
4: **for** $k \leftarrow 2$ **to** n **do**
5: $\quad \mathcal{C}_k \leftarrow$ Eq. ((11))
6: $\quad$ **if** $k > 2$ **then**
7: $\quad\quad$ Delete all edges with $< k-2$ triangles in $\mathcal{C}_k$ (computed without considering edge probabilities)
8: $\quad$ **if** $\mathcal{C}_k = \emptyset$ **then**
9: $\quad\quad$ **break**
10: $\quad$ **for each** connected component $\mathcal{C}$ of $\mathcal{C}_k$ **do**
11: $\quad\quad$ $\mathsf{S}_k \leftarrow$ all (ϵ, δ)-approximate (k, γ)-trusses in $\mathcal{C}$, found by Algorithm 4 or Algorithm 5
12: **return** all maximal (ϵ, δ)-approximate (k, γ)-trusses in $\mathsf{S}_k, \forall k$

Algorithm 4 Top-down exact search (TopDownSearch)

Require: $D_{\mathcal{G}} = \{G_1, ..., G_N\}$, k, $\mathcal{C} = (V_{\mathcal{C}}, E_{\mathcal{C}})$
Ensure: all (ϵ, δ)-approximate global (k, γ)-trusses in $\mathcal{C}$
1: Ans $\leftarrow \emptyset$.
2: **if** $\forall e \in E_{\mathcal{C}}$ such that $\widehat{\alpha}_k(\mathcal{C}, e) \geq \gamma$ **then**
3: $\quad$ **return** $\mathcal{C}$
4: **else**
5: $\quad$ **for** $e \in E_{\mathcal{C}}$ **do**
6: $\quad\quad$ $\mathcal{C}' \leftarrow \mathcal{C} - \{e\}$ {remove e from $\mathcal{C}$}
7: $\quad\quad$ Delete edges in $\mathcal{C}'$ having $< k-2$ triangles
8: $\quad\quad$ **for each** connected component $\mathcal{C}''$ of $\mathcal{C}'$ **do**
9: $\quad\quad\quad$ Ans $\leftarrow$ Ans $\cup$ TopDownSearch(D, k, $\mathcal{C}''$)
10: **return** Ans

(k, γ)-truss unless it also belongs to a local (k, γ)-truss. It also cannot belong to a maximal (k, γ)-truss unless it belongs to a maximal global $(k-1, \gamma)$-truss. Thus, edges outside this intersection can be safely pruned. Complications arise when dealing with approximate trusses, which are found by sampling. However, w.r.t. a given set of N possible worlds, it is easy to verify that every (ϵ, δ)-approximate maximal global (k, γ)-truss is also an (ϵ, δ)-approximate maximal global $(k-1, \gamma)$-truss. This observation allows us to prune the candidate set $\mathcal{C}_k$ significantly, as follows.

Let S_k denote the set of all (ϵ, δ)-approximate global (k, γ)-trusses produced by Algorithm 3 for any given $k \geq 2$. For convenience, we define $\mathsf{S}_1 = E_{\mathcal{G}}$, i.e., the set of all edges in the original graph $\mathcal{G}$ and we let $\bigcup \mathsf{S}_k$ denote the union of the edge sets associated with the satisfying trusses in S_k, for a given k. Hence, we have:

$$\mathcal{C}_k = \{e \in E_{\mathcal{H}} : \mathcal{H} \text{ is a maximal local } (k, \gamma)\text{-truss in } \mathcal{G}\} \cap (\bigcup \mathsf{S}_{k-1}). \quad (11)$$

For $k = 2$, $\mathcal{C}_2$ reduces to the set of edges present in some maximal local $(2, \gamma)$-truss. When $k > 2$, we perform an additional pruning step by removing from $\mathcal{C}_k$ all edges that are contained in less than $k-2$ triangles (lines 6–7). The computation of triangles in this step does not take edge probabilities into account, which is equivalent to treating as if each edge had a probability of 1. Such removal clearly further prunes our search space, as those removed edges have no chance of being in any (ϵ, δ)-approximate global (k, γ)-trusses.

Algorithm 5 Bottom-up heuristic (BottomUpSearch)

Require: $D_{\mathcal{G}} = \{G_1, ..., G_N\}$, k, $\mathcal{C} = (V_{\mathcal{C}}, E_{\mathcal{C}})$
Ensure: all (ϵ, δ)-approximate global (k, γ)-trusses in $\mathcal{C}$
1: Ans $\leftarrow \emptyset$
2: **for all** $e \in E_{\mathcal{C}}$ **do**
3: $Q \leftarrow \{e\}$ {Q is an induced graph on e.}
4: **while** $\exists e \in Q$ such that $\mathsf{sup}_Q(e) < k-2$ **do**
5: Add $k-2$ triangles of $\mathcal{C}$ that have e to Q
6: **if** $\forall e \in E_Q$, such that $\widehat{\alpha}_k(Q, e) \geq \gamma$ **then**
7: Extend Q to be maximal by adding edges.
8: Ans $\leftarrow$ Ans $\cup\{Q\}$
9: **return** Ans

After that, we search the subgraphs of each each connected component of $\mathcal{C}_k$ to identify all satisfying (ϵ, δ)-approximate global (k, γ)-trusses (lines 10–11). This is done by invoking a searching sub-procedure, Algorithm 4 or Algorithm 5, which we describe shortly. The whole process terminates as soon as the candidate set $\mathcal{C}_{k'}$ becomes empty, for a certain k'. By the monotonicity of global (k, γ)-trusses w.r.t. k mentioned above, for any $k'' > k'$, we know $\mathcal{C}_{k''}$ is also empty. Lastly, for each k such that $\mathcal{C}_k$ is non-empty, we return all maximal subgraphs in S_k as output, which are guaranteed to be the (ϵ, δ)-approximate global (k, γ)-trusses (line 12).

Top-down exact search. Algorithm 4 presents a DFS-based algorithm for finding all satisfying trusses, namely maximal (ϵ, δ)-approximate global (k, γ)-trusses, given as input k and a connected component $\mathcal{C}$ of candidate graph $\mathcal{C}_k$. The pseudo-code is mostly self-explanatory. It first checks if $\mathcal{C}$ is itself a satisfying truss. If so, then $\mathcal{C}$ is also maximal by construction, and thus is returned (lines 2–3). Otherwise, we remove an edge from $\mathcal{H}$, and recursively remove all edges whose support is less than $k-2$ as a result (lines 6–7). Let the resulting graph be $\mathcal{C}'$. We then recursively run this search algorithm on each connected component of $\mathcal{C}'$ (lines 8–9). Finally, all satisfying trusses are gathered and returned to Algorithm 3.

Bottom-up heuristic search. The above top-down approach may suffer from inefficiency, as the search process that removes one edge at a time can be expensive. To combat this, we propose a bottom-up heuristic search method that provides significant speedup, in exchange for incompleteness. That is, it may not discover all satisfying trusses.

The pseudo-code is presented in Algorithm 5. This search method grows a potential satisfying truss from Q that is initialized to a single edge from the candidate component. For each edge $e \in Q$, we repeatedly add into Q the edges that can form triangles with e, until all edges in Q have a support of at least $k-2$ (lines 3–5). We then check if Q satisfies the definition of (ϵ, δ)-approximate global (k, γ)-truss w.r.t. $D_{\mathcal{G}}$: if so, it will be included in a candidate solution. Since it may not be maximal, the algorithm will extend it by adding local edges to achieve maximality (lines 6–8).

The above bottom-up building process is repeated for all edges. As a heuristic, we rank edges in $\mathcal{C}$ in descending order of their edge probability. Also, when we grow Q, it is possible that an edge e may participate in more than $k-2$ triangles, in which case we randomly select $k-2$ of these. The reason for not adding all such triangles is that the more edges a subgraph includes, the more likely there is an edge that violates the constraint for a global (k, γ)-truss. There are other options such as choosing those triangles whose edges participate in the largest number of triangles. Such "look-ahead" strategies involve additional overhead and we do not explore them further in this paper. A careful study of the trade-off introduced by such additional heuristics is an interesting question for future work.

Network	$\lvert V_{\mathcal{G}}\rvert$	$\lvert E_{\mathcal{G}}\rvert$	d_{max}	$\lvert V_{\mathcal{C}}\rvert$	$\lvert E_{\mathcal{C}}\rvert$	#comp
FruitFly	3751	3692	27	2400	2771	435
WikiVote	7118	103689	1065	7066	103663	24
Flickr	24125	300836	546	21398	296202	840
DBLP	684911	2284991	611	581539	2169186	34132
BioMine	1008200	6742939	139624	961760	6688520	16242
LiveJournal	4847571	42851237	20333	4843953	42845684	914
Orkut	3072441	117185083	33313	3072441	117185083	1
Wise	58655849	261321033	278489	58655820	261321018	15

Table 1: Network statistics

6. EXPERIMENTS

We conduct extensive experiments to test the effectiveness and efficiency of our proposed algorithms for LocalDecomp and GlobalDecomp. All algorithms are implemented in C++, and all the experiments are conducted on a Linux Server with Intel Xeon CUP X5570 (2.93 GHz) and 100GB main memory.

6.1 Dataset and Experimental Setup

We use eight real-world probabilistic graphs, whose basic statistics are summarized in Table 1. For each network $\mathcal{G}$, besides the number of vertices $|V_{\mathcal{G}}|$, number of edges $|E_{\mathcal{G}}|$ and maximum degree d_{max}, we also report the size of the largest connected component $\mathcal{C}(V_{\mathcal{C}}, E_{\mathcal{C}})$ and the number of connected components (#comp).

Flickr (`https://www.flickr.com/`) is a popular online community for sharing photos. The network data contains 24.1K nodes (representing users) and 301K edges, where the probability of an edge between two users is calculated by the Jaccard coefficient of the interest groups of the two users [4, 27].

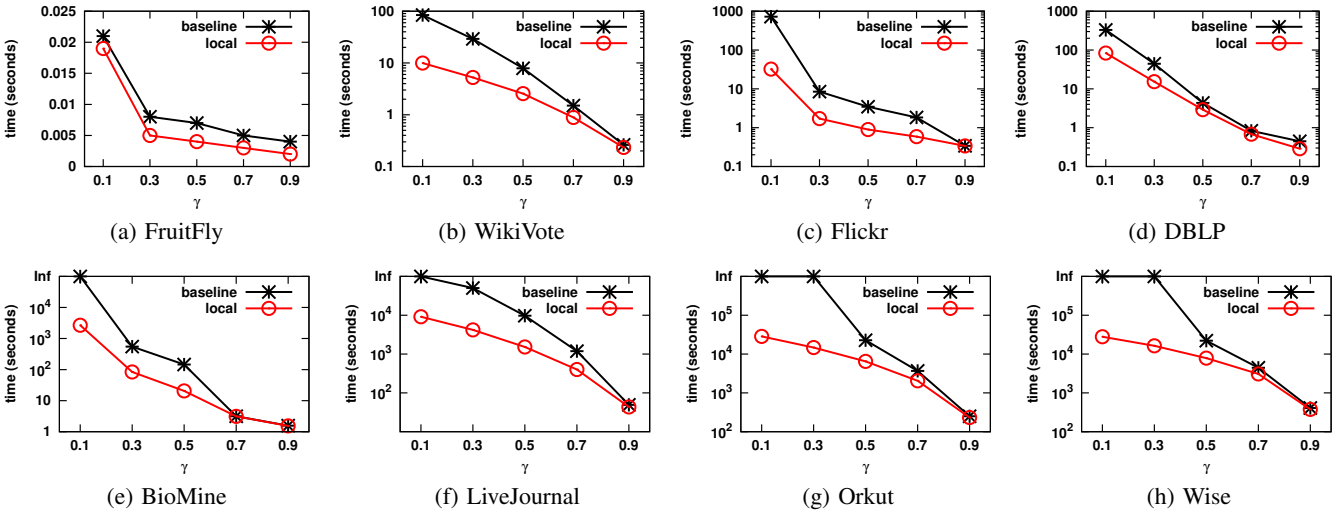
DBLP (`http://dblp.uni-trier.de/`) is a computer science bibliography website. The probabilistic graph consists of 685K nodes and 2.3M edges. Here, each node corresponds to an author, and edges represent co-authorship relationships. Precisely, previous work [4, 27] measures the probability of each edge based on an exponential function of the number of collaborations.

BioMine is a snapshot of the database of the BioMine project [13] containing biological interactions (`http://BioMine.cs.helsinki.fi/search/`). The graph contains 1.01M nodes and 6.74M edges, where the probability of an edge corresponds to the confidence that the interaction actually exists [4, 27].

FruitFly is a protein-protein interaction (PPI) network [23], obtained by integrating data from the BioGRID (`http://thebiogrid.org/`) database and data from the STRING database [27]. WikiVote, LiveJournal and Orkut are social networks downloaded from the Stanford Network Analysis Project (`http://snap.stanford.edu/`); Wise (`http://www.wise2012.cs.ucy.ac.cy/challenge.html`) is a micro-blogging network from WISE 2012 Challenge. For these four networks, the edge probabilities are assigned uniformly at random from the interval $[0, 1]$.

Comparison Methods. To evaluate the efficiency and effectiveness of local and global (k, γ)-truss decomposition methods, we test and compare three algorithms proposed in this paper, namely, Local, GTD, and GBU. Here, Local is the algorithm for LocalDecomp by combining Algorithm 1 and Algorithm 2 using dynamic programming strategies. GTD is the algorithm for GlobalDecomp by combining Algorithm 3 and Algorithm 4. GBU is also an algorithm for GlobalDecomp, which uses heuristic search strategies by combining Algorithm 3 and Algorithm 5.

Evaluation Metrics. For efficiency, we report running time in seconds. To evaluate the quality of an output truss, we use the follow-



(a) FruitFly (b) WikiVote (c) Flickr (d) DBLP

(e) BioMine (f) LiveJournal (g) Orkut (h) Wise

Figure 5: Local (k,γ)-truss decomposition: dynamic programming vs. baseline (re-computing edge support probabilities from scratch after each edge removal) in terms of running time (in seconds)

ing two metrics. We define the *density* (cohesiveness) of a probabilistic graph $\mathcal{H}$ as

$$\mathsf{density}(\mathcal{H}) = \frac{\sum_{e\in E_{\mathcal{H}}} p(e)}{\frac{1}{2}|V_{\mathcal{H}}|\cdot(|V_{\mathcal{H}}|-1)}, \tag{12}$$

where the numerator can be interpreted as the weighted sum of existing edges (where weights are existence probabilities) and the denominator is the maximum number of possible edges $\mathcal{H}$ can have. The second metric is the *probabilistic clustering coefficient* [26]:

$$\mathsf{PCC}(\mathcal{H}) = \frac{3\sum_{\triangle_{uvw}\in\mathcal{H}} p(u,v)p(v,w)p(w,u)}{\sum_{(u,v),(u,w)\in E(\mathcal{H}),v\neq w} p(u,v)p(u,w)}. \tag{13}$$

Note that, a graph $\mathcal{H}$ containing only a single edge is not considered for PCC in experiments. For GTD and GBU, we set the parameters $\epsilon = 0.1$ and $\delta = 0.1$, and randomly sample a total of $N = 150$ graphs where $N \geq \frac{1}{2\varepsilon^2}\ln(\frac{2}{\delta})$ by Theorem 3.

6.2 Efficiency Evaluation

Local (k,γ)-truss decomposition. For LocalDecomp, we test the proposed method Local, and compare its efficiency with a naive baseline, which still uses Algorithm 1 as its backbone, but whenever an update of edge support probabilities is needed (after an edge removal), it computes the entire vector $\sigma(\cdot)$ from scratch. The running time results are illustrated in Figure 5 by varying γ from 0.1 to 0.9. Note that the Y-axis for Figure 5(b) – 5(h) is in log-scale.

Overall, as γ increases, the requirement for a subgraph to be a local (k,γ)-truss becomes stricter, and the running time decreases. This is expected, as more edges can be pruned quickly and the algorithm enjoys working with a smaller graph. As can be seen, on all datasets, the version that uses dynamic programming for updating the probabilities is faster than the naive baseline. In particular, on large datasets (from WikiVote to Wise), the dynamic programming version is often more than one order of magnitude faster than the baseline, indicating its superior efficiency and scalability.

Global (k,γ)-truss decomposition. We next report the running time of our decomposition for (ϵ,δ)-approximate global (k,γ)-truss. As mentioned in Section 5, the top-down search algorithm GTD may suffer from inefficiency issues due to the inherent hardness of GlobalDecomp, and a bottom-up heuristic method GBU is proposed to alleviate the issue.

As can be seen from Figure 6, on a small probabilistic graph FruitFly, the GTD cannot finish in a reasonable amount of time for

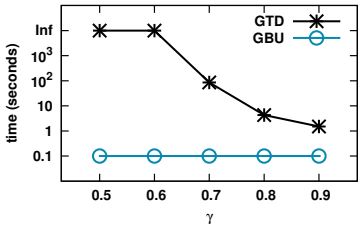


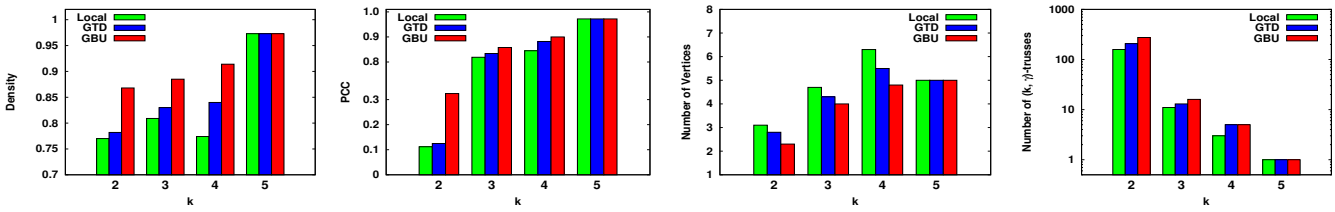
Figure 6: Running time (in secs) of GTD and GBU on FruitFly

Network	$\gamma=0.1$	$\gamma=0.3$	$\gamma=0.5$	$\gamma=0.7$	$\gamma=0.9$
FruitFly	0.1	0.1	0.1	0.1	0.1
WikiVote	27.2	10.2	6.2	1.4	0.3
Flickr	40.9	11.7	11.4	11.1	11.0
DBLP	168	42.8	16.2	7.5	3.5
BioMine	2660	164	43.8	43.6	25.0
LiveJournal	24397	11828	4429	1173	146
Orkut	69870	33671	6178	2659	421
Wise	66085	49512	22861	8036	843

Table 2: Running time (in seconds) of GBU on all networks

$\gamma = 0.5$ and 0.6. GTD can accomplish the task in reasonable for $\gamma \geq 0.7$, but is often orders of magnitude slower than GBU. This indicates the scalability limitation of GTD. Thus, in subsequent experiments, we keep using the bottom-up heuristic for GlobalDecomp, whose running time on all datasets are shown in Table 2. As can be seen, similar to local (k,γ)-truss decomposition, the running time decreases as γ increases. The running time of GBU increases essentially linearly with graph size. Since all graphs tested have a single dominant connected component (see Table 1), these findings are reliable and attest to the scalability of GBU over graphs with millions of edges.

Memory Usage. In this experiment, we report the memory usage of proposed methods with $\gamma = 0.5$ in Figure 8. The black bar represents the space required for storing the graph on disk. As can be seen, the memory usage of GBU is the same memory as for Local. Both methods consume less than 20 times of graph size (see Figure 8). Local (k,γ)-truss decomposition takes $O(\rho|E_{\mathcal{G}}|)$ space complexity to store the edge support vectors, $|E_{\mathcal{G}}|$ being the number of edges. It can be inferred from Figure 8 that ρ tends to be small for real sparse networks. The memory consumption of GBU mainly consists of the edge support vectors and N sampled graphs. For an edge $e \in \mathcal{G}$, we use one bit '1' or '0' to record whether e exists in each sampled graph G_j, $1 \leq j \leq N$. It takes 192 bits



(a) Density (b) PCC (c) Number of Vertices (d) Number of (k, γ)-trusses

Figure 7: Quality Comparison on Fruit-Fly with $\gamma = 0.7$ varying k

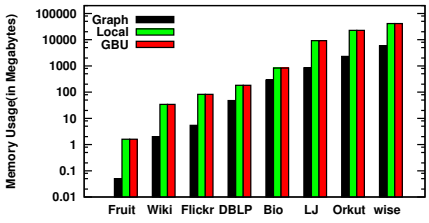
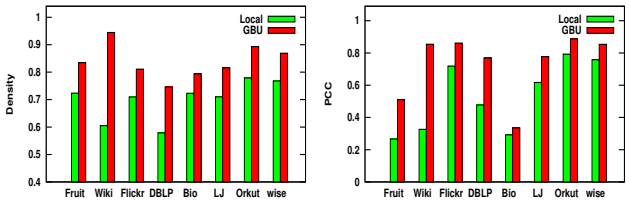


Figure 8: Memory Usage of proposed methods on all networks



(a) Density (b) PCC

Figure 9: Quality Comparison between Local (k, γ)-truss and Global (k, γ)-truss with $\gamma = 0.5$ on all networks

(6 bytes) to record all sample results for one edge. Thus, the total memory cost of sampled graphs is significantly less than the cost of edge support vectors. Once we obtain the local truss-ness of all edges, we can release the memory used for storing edge support vectors, before creating sample graphs. As a result, GBU still costs the same memory as Local. Both methods show a good space scalability to large real networks.

6.3 Quality Evaluation

Recall that our motivation of proposing global (k, γ)-truss is to extract more cohesive subgraphs through more stringent definitions. To validate our proposal, in this experiment we directly compare the cohesiveness of these two types of probabilistic trusses using density and probabilistic clustering coefficient (PCC).

Figures 7(a)–(d) report the average density, the average PCC, the average vertex number of (k, γ)-truss and the number of (k, γ)-truss respectively found by all three methods on FriutFly[2], for varying k, with γ fixed at $\gamma = 0.7$. Figures 7(a)-(b) show that the output global (k, γ)-trusses consistently achieve higher density and PCC than the local ones, as expected. An exception is $k = 5$. For $k = 5$, all three methods find the same *one* (k, γ)-truss, which structurally is a 5-clique. The heuristic method GBU achieves higher density and PCC than GTD. The reason is two-fold. GBU always starts the exploration from edges with high probability to form (k, γ)-truss, which potentially leads to a higher density and PCC than GTD. Secondly, note that GBU cannot find all (k, γ)-trusses as GTD, and some of (k, γ)-trusses found by GBU may not be maximal, which may also increase density. Generally, for both local and global (k, γ)-truss, density and PCC become larger as k increases, as denser (k, γ)-trusses will be found by removing edges of low support and small probability. In Figure 7(c), GTD and GBU both found smaller (k, γ)-trusses than Local, reflecting that global (k, γ)-truss is a stricter definition than the local one. The (k, γ)-trusses of GBU are the smallest ones. In Figure 7(d), the number of output (k, γ)-trusses decreases as k goes up: indeed, increasing k leads to stricter a requirement by definition. For GBU, as a heuristic it may not discover some of the truly maximal satisfying trusses as found by GTD; instead it may find some other non-maximal ones. Thus, GBU sometimes outputs more trusses than GTD.

Next, we compare Local and GBU on larger networks. GTD is not compared here owing to its efficiency limitation. We report the results on average density and average PCC of all (k, γ)-trusses for all possible k with $\gamma = 0.5$ in Figure 9 (a)-(b). The results of the vertex size and the number of (k, γ)-trusses on these datasets are similar as FriutFly, and are thus omitted. As we can see, GBU achieves higher density and PCC than Local on all networks. Once again the superiority of global (k, γ)-truss is established.

6.4 Comparisons with (k, η)-Cores

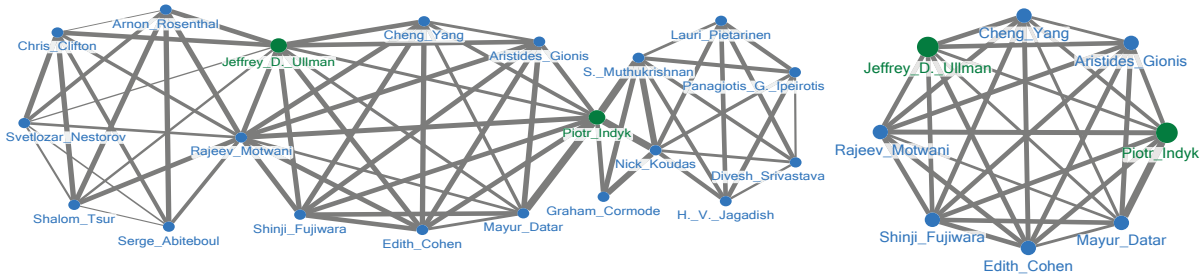
In this experiment, we show that local (k, γ)-truss is superior to (k, η)-core [4] as a type of cohesive probabilistic subgraphs when we take into account, not just graph structure, but also the *probability of existence* of that structure. For fairness of comparison, we use our local definition, since (k, η)-core is defined in a similar fashion to local (k, γ)-truss. For a given η, we use k_{cmax} to denote the maximum core number. Then, we set $\gamma = \eta$, and use k_{tmax} to denote the maximum truss number. For simplicity, we denote the (k_{cmax}, η)-core by C and the (k_{tmax}, γ)-truss by T. Table 3 reports the statistics of T and C on WikiVote, DBLP and Biomine. The results of other datasets are similar and are omitted. We vary the parameter $\eta = \gamma \in \{0.1, 0.5\}$. In terms of both the number of vertices and edges, we can see that the size of T is significantly smaller than that of C, which shows that the (k_{tmax}, γ)-truss and (k_{cmax}, η)-core are indeed quite different in terms of the kinds of cohesive subgraphs they extract. Note that k_{tmax} is always smaller than k_{cmax} for all $\gamma = \eta \in \{0.1, 0.5\}$.

First, ignore the edge probabilities and consider only the graph structure. The clustering coefficient of T is comparable to that of C on all datasets, and on Wikivote, T has a higher clustering coefficient. In addition, both C and T had nearly clique structure on DBLP and Biomine. However, this is not the true whole story as this ignores the probability of existence of the structure present in the found cores and trusses. As we can see, in terms of PCC and density, the (k_{tmax}, γ)-truss clearly outperforms (k_{cmax}, η)-core on all networks. This is because the (k, γ)-truss takes the probability of an edge contained in triangles into consideration, whereas (k, η)-core does not. As a result, (k, γ)-truss can prune edges with small probability as well as those having a small porbability of being contained in many triangles, and achieve a higher density and PCC. In conclusion, the results demonstrate that (k_{tmax}, γ)-truss is significantly more cohesive and tightly-knit than the (k_{cmax}, η)-

[2]For all other datasets, under our parameter settings, GTD cannot finish. Thus Figure 7 demonstrates using FruitFly.

Network	$\eta = \gamma$	V_T/V_C	E_T/E_C	k_{tmax}/k_{cmax}	CC_T/CC_C	$\mathsf{PCC}_T/\mathsf{PCC}_C$	den_T/den_C
WikiVote	0.1	280/1080	7660/47873	8/21	0.386/0.200	**0.220/0.100**	**0.115/0.041**
WikiVote	0.5	82/985	891/44243	5/20	0.395/0.212	**0.305/0.105**	**0.210/0.046**
DBLP	0.1	34/112	261/6216	14/25	1.0/1.0	**0.619/0.316**	**0.611/0.263**
DBLP	0.5	10/114	45/6441	10/20	1.0/1.0	**0.992/0.319**	**0.992/0.265**
Biomine	0.1	102/199	5127/19701	33/58	0.996/1.0	**0.539/0.280**	**0.539/0.279**
Biomine	0.5	101/201	4996/20100	18/52	0.990/1.0	**0.536/0.280**	**0.536/0.279**

Table 3: Statistics of (k,γ)-truss, T, and (k,η)-core, C, on WikiVote, DBLP and Biomine. the number of vertices (V_T/V_C), the number of edges (E_T/E_C), the maximum truss/core number (k_{tmax}/k_{cmax}), the clustering coefficient (CC_T/CC_C), the probabilistic clustering coefficient ($\mathsf{PCC}_T/\mathsf{PCC}_C$) and the probabilistic density(den_T/den_C) respectively.



(a) Local (k,γ)-truss (b) Global (k,γ)-truss

Figure 10: An example of task-driven team formation with query nodes ={"Jeffrey D. Ullman", "Piotr Indyk"} and keywords ={"data", "algorithm"}. In the plots, a wider line indicates a higher probability on the edge.

core, for $\gamma = \eta$, when we take both structure and its existence probability into account.

6.5 Task-Driven Team Formation

We apply our local and global (k,γ)-truss decomposition algorithms to solve the task-driven team formation problem. The original problem definition [4] is w.r.t. (k,η)-core specifically. To be able to perform this experiment, we adapt the definition and make it specific w.r.t. (k,γ)-truss: Given a probabilistic graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ derived specifically for a task T, a query (Q,T) with nodes $Q \subset \mathcal{V}$ and task T, as well as $\gamma \in (0,1)$, find a local/global (k,γ)-truss from $\mathcal{G}$ that (i) contains all nodes in Q and (ii) has the highest truss-ness k that satisfies γ. The edge probabilities in $\mathcal{G}$ are defined w.r.t. task T, following the authors of [4].

We use a DBLP collaboration network with meta-data obtained from [4]. Each node represents an author, and an edge is drawn between two authors if they co-authored at least one paper. The resulting graph has 1.1M nodes and 4.1M edges. For each edge (u,v), the data contains the titles of all papers coauthored by u and v. Given a set W of keywords, the edge probability of (u,v) represents the collaboration strength of papers co-authored by u and v related to keywords W. For each edge, [4] takes the bag of words of the titles of all papers coauthored by the two authors, and applies the Latent Dirichlet Allocation (LDA) model [3] to infer its topics and calculates the edge probability. The probabilistic graph on keywords W is denoted $\mathcal{G}_W$.

We set $\eta = \gamma = 10^{-11}$ as suggested by authors of [4] due to the low edge probabilities in the data. The sample query is ({"Jeffrey D. Ullman", "Piotr Indyk"}, {"data", "algorithm"}). Figure 10(a) depicts a local $(4, 10^{-11})$-truss containing both authors. The local truss has 20 nodes, 67 edges, with density 0.002 and PCC 0.005. Furthermore, we use the local (k,γ)-truss shown in Figure 10(a) as the input of global (k,γ)-truss decomposition and obtain 17 global (k,γ)-trusses, one of which is shown in Figure 10 (b): It has 8 nodes, 28 edges, density 0.007, and PCC 0.007. Once again, the superiority of global (k,γ)-truss is shown.

In contrast, applying (k,η)-core decomposition [4] to this query results in a $(5, 10^{-11})$-core with 1153 nodes, 13355 edges, with density $6 \cdot 10^{-5}$ and PCC 0.002. As we can see, team formation by (k,γ)-truss produces a much more desirable team, i.e., it is much smaller and denser than (k,η)-core. Intuitively, this is arguably better for the task of writing a research paper related to "data" and "algorithm" as it is unrealistic for 1153 researchers to collaborate.

7. SUMMARY AND FUTURE WORK

Motivated by applications in biological, social, and communication networks, we propose, for the first time, extensions to the definition of a k-truss for probabilistic graphs. Our framework allows for a local as well as a global version of a probabilistic truss, a (k,γ)-truss, to be precise. (k,γ)-trusses correspond to probabilistic cohesive subgraphs and we motivate the truss decomposition problem for the world of probabilistic graphs as that of finding maximal (local or global) (k,γ)-trusses. We develop an elegant and efficient dynamic programming algorithm for finding all maximal local (k,γ)-trusses of a given probabilistic graph. For global (k,γ)-trusses, for a given probabilistic graph, the number of maximal global (k,γ)-trusses can be exponential; also even computing the probability of a given subgraph containing a k-truss that connects all its nodes, a task needed for finding global (k,γ)-trusses, is #P-hard. We thus propose an approach that combines sampling along with heuristic search, to find a subset of approximate (k,γ)-trusses efficiently. We conducted extensive experiments on 6 real datasets. Our results demonstrate that our algorithms significantly outperform natural baselines as well as alternative techniques based on probabilistic extensions to k-cores. The experiments show the efficiency and effectiveness of our proposed algorithms.

This work opens up several interesting questions. First developing further efficient and clever heuristics for finding maximal global (k,γ)-trusses is important. Second, given k, how to find maximal (local or global) (k,γ)-trusses for various possible γ? Notice that this problem is well defined even though γ is a real number, since there are only finitely many k-trusses in a (probabilistic) graph, each of which must be a (k,γ)-truss for some maximum possible γ. Finding efficient solution to this problem is wide open.

ACKNOWLEDGMENTS

This work was supported by a Discovery grant and a Discovery Accelerator Supplements grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

8. REFERENCES

[1] A.-L. Barabasi and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

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

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, pages 1316–1325. ACM, 2014.

[5] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[6] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[7] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *SIGMOD*, pages 447–458, 2010.

[8] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[9] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. Technical report, National Security Agency, 2008.

[10] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science and Engineering*, 11(4):29–41, 2009.

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

[12] J. Dong and S. Horvath. Understanding network concepts in modules. *BMC Systems Biology*, 1(24), 2007.

[13] L. Eronen and H. Toivonen. Biomine: predicting links between biological entities using network models of heterogeneous databases. *BMC Bioinformatics*, 13, 2012.

[14] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[15] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[16] R. Jin, L. Liu, and C. C. Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *KDD*, pages 992–1000. ACM, 2011.

[17] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint reachability computation in uncertain graphs. *PVLDB*, 4(9):551–562, 2011.

[18] G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *TKDE*, 25(2):325–336, 2013.

[19] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, et al. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084):637–643, 2006.

[20] Y. Li, T. Kuboyama, and H. Sakamoto. Truss decomposition for extracting communities in bipartite graph. In *Third International Conference on Advances in Information Mining and Management*, pages 76–80, 2013.

[21] L. Liu, R. Jin, C. Aggarwal, and Y. Shen. Reliable clustering on uncertain graphs. In *ICDM*, pages 459–468. IEEE, 2012.

[22] R. J. Mokken. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.

[23] A. P. Mukherjee, P. Xu, and S. Tirthapura. Mining maximal cliques from an uncertain graph. *arXiv preprint arXiv:1310.6780*, 2013.

[24] J. Pei, D. Jiang, and A. Zhang. Mining cross-graph quasi-cliques in gene expression and protein interaction data. In *ICDE*, pages 353–354, 2005.

[25] J. B. Pereira-Leal, A. J. Enright, and C. A. Ouzounis. Detection of functional modules from protein interaction networks. *PROTEINS: Structure, Function, and Bioinformatics*, 54(1):49–57, 2004.

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

[27] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *PVLDB*, 3(1-2):997–1008, 2010.

[28] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept*. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

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

[30] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.

[31] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112, 2013.

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

[33] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[34] J. Wang, J. Cheng, and A. W.-C. Fu. Redundancy-aware maximal cliques. In *KDD*, pages 122–130, 2013.

[35] J. Xiang, C. Guo, and A. Aboulnaga. Scalable maximum clique computation using mapreduce. In *ICDE*, pages 74–85, 2013.

[36] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, pages 1049–1060, 2012.

[37] F. Zhao and A. K. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment*, 6(2):85–96, 2012.

[38] Z. Zou, H. Gao, and J. Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *KDD*, pages 633–642, 2010.

[39] Z. Zou, J. Li, H. Gao, and S. Zhang. Mining frequent subgraph patterns from uncertain graph data. *TKDE*, 22(9):1203–1218, 2010.

Appendix: Additional Proofs

Proof of Lemma 2. Consider an extension of the graph $\mathcal{G}$ shown in Figure 11, where there are n triangles attached to the central node, instead of 4. Consider $k = 3$ and $\gamma = (1/2)^{3\lceil n/2 \rceil}$. Then it can be easily verified that a maximal global (k, γ)-truss is a subgraph of $\mathcal{G}$ consisting of $\lceil n/2 \rceil$ triangles. There are $\binom{n}{\lceil n/2 \rceil}$ such choices, which is exponential in n.

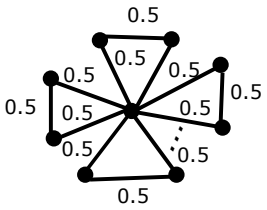


Figure 11: A "windmill" probabilistic graph

Proof of Theorem 2. We prove by contradiction. Suppose a maximal local (k, γ)-truss $\mathcal{H}$ is not in the answer set of Algorithm 1. First, for each edge $e \in \mathcal{H}$ has $\sigma(e, k-2)p(e) \geq \gamma$ by Definition 2, thus $\tau(e) \geq k$. Due to the fact that Algorithm 1 collects together all edges e with $\tau(e) \geq k$, no edges of $\mathcal{H}$ are missed by Algorithm 1. Moreover, all edges of $\mathcal{H}$ are connected, and will be assigned into the same component by the post-processing step. Thus, if $\mathcal{H}$ is not in the answer set of Algorithm 1, the only possibility is that there exists a larger local (k, γ)-truss containing $\mathcal{H}$. But this violates the maximal property of $\mathcal{H}$, a contradiction.

Proof of Lemma 3. First, the probability of a possible world H of $\mathcal{H}$ is

$$\Pr[H|\mathcal{H}] = \prod_{e \in E_H} p(e) \prod_{e \in E_{\mathcal{H}} \setminus E_H} (1 - p(e)).$$

Next, for each possible world $G \sqsubseteq \mathcal{G}$ such that $G \in X(H)$, we have $E_G \cap E_{\mathcal{H}} = E_H$; we can divide the edge set E_G into two disjoint subsets E_1 and E_2, where $E_1 = E_H$ and $E_2 = E_G \setminus E_{\mathcal{H}}$. Since $E_G \cap E_{\mathcal{H}} = E_H$, we also have $E_2 = E_G \setminus E_{\mathcal{H}} = E_G \setminus E_H$ and $E_{\mathcal{H}} \setminus E_G = E_{\mathcal{H}} \setminus E_H$. Then, we obtain the probability

$$\begin{aligned}
\Pr[G|\mathcal{G}] &= \prod_{e \in E_G} p(e) \prod_{e \in E(\mathcal{G}) \setminus E_G} (1 - p(e)) \\
&= \prod_{e \in E_1 \cup E_2} p(e) \prod_{e \in (E_{\mathcal{H}} \cup (E(\mathcal{G}) \setminus E_{\mathcal{H}})) \setminus E_G} (1 - p(e)) \quad (14) \\
&= \prod_{e \in E_1} p(e) \prod_{e \in E_2} p(e) \prod_{e \in E_{\mathcal{H}} \setminus E_G} (1 - p(e)) \\
&\quad \prod_{e \in (E(\mathcal{G}) \setminus E_{\mathcal{H}}) \setminus E_G} (1 - p(e)) \quad (15) \\
&= \prod_{e \in E_H} p(e) \prod_{e \in E_{\mathcal{H}} \setminus E_H} (1 - p(e)) \prod_{e \in E_2} p(e) \\
&\quad \prod_{e \in (E(\mathcal{G}) \setminus E_{\mathcal{H}}) \setminus E_G} (1 - p(e)) \\
&= \Pr[H|\mathcal{H}] \prod_{e \in E_2} p(e) \prod_{e \in (E(\mathcal{G}) \setminus E_{\mathcal{H}}) \setminus E_2} (1 - p(e)) \\
&= \Pr[H|\mathcal{H}] \Pr[G'|\mathcal{G}'], \quad (16)
\end{aligned}$$

where $\mathcal{G}' = (V(\mathcal{G}), E(\mathcal{G}) \setminus E_{\mathcal{H}})$ and $G' = (V(\mathcal{G}), E_2) = (V(\mathcal{G}), E_G \setminus E_{\mathcal{H}})$. Also notice that in going from Eq. (14) to Eq. (15), we have used the fact that $E_{\mathcal{H}} \setminus E_G$ and $(E_{\mathcal{G}} \setminus E_{\mathcal{H}}) \setminus E_G$ are disjoint. Now, clearly, G' has no edges in any possible world of $\mathcal{H}$, and $G' \sqsubseteq \mathcal{G}'$.

For a given possible world $H \sqsubseteq \mathcal{H}$, summing both sides of Eq. (16) over all possible worlds of $\mathcal{G}$ that project to H, we have

$$\sum_{G \in X(H)} \Pr[G|\mathcal{G}] = \sum_{G \in X(H)} \Pr[H|\mathcal{H}] \Pr[G'|\mathcal{G}'].$$

That is,

$$\begin{aligned}
\sum_{G \in X(H)} \Pr[G|\mathcal{G}] &= \sum_{G \in X(H)} \Pr[H|\mathcal{H}] \Pr[G'|\mathcal{G}'] \\
&= \Pr[H|\mathcal{H}] \sum_{G' \sqsubseteq \mathcal{G}'} \Pr[G'|\mathcal{G}'] \\
&= \Pr[H|\mathcal{H}].
\end{aligned}$$

This completes the proof.

Proof of Corollary 1. We define the indicator function $\mathbf{I}(G \in X_H)$ which takes on 1 if $G \in X_H$, and 0 otherwise. From Definition 3 and Lemma 3, we have:

$$\begin{aligned}
\alpha_k(\mathcal{H}, e) &= \sum_{H \sqsubseteq \mathcal{H}} \Pr[H|\mathcal{H}] \cdot \mathbf{I}(H, k, e) \\
&= \sum_{H \sqsubseteq \mathcal{H}} \sum_{G \in \mathbf{X}_H} \Pr[G|\mathcal{G}] \cdot \mathbf{I}(H, k, e) \\
&= \sum_{G \sqsubseteq \mathcal{G}} \sum_{H \sqsubseteq \mathcal{H}} \mathbf{I}(G \in X_H) \cdot \Pr[G|\mathcal{G}] \cdot \mathbf{I}(H, k, e) \\
&= \sum_{G \sqsubseteq \mathcal{G}} \Pr[G|\mathcal{G}] \sum_{H \sqsubseteq \mathcal{H}} \mathbf{I}(G \in X_H) \cdot \mathbf{I}(H, k, e), \quad (17) \\
&= \sum_{G \sqsubseteq \mathcal{G}} \Pr[G|\mathcal{G}] \cdot \mathbf{I}(G \downarrow_{\mathcal{H}}, k, e). \quad (18)
\end{aligned}$$

The last equality follows from the fact that although the second summation in (17) is over all $H \sqsubseteq \mathcal{H}$, the product $\mathbf{I}(G \in X_H) \cdot \mathbf{I}(H, k, e)$ is non-zero for at most one possible world of $\mathcal{H}$, namely $G \downarrow_{\mathcal{H}}$, and for this world, the product reduces to $\mathbf{I}(G \downarrow_{\mathcal{H}}, k, e)$.