Fast Truss Decomposition in Large-scale Probabilistic Graphs

Fatemeh Esfahani, Jian Wu, Venkatesh Srinivasan, Alex Thomo, and Kui Wu
{esfahani,wujian,srinivas,thomo,wkui}@uvic.ca

ABSTRACT

Truss decomposition is popular for finding dense substructures in graphs. Discovering trusses in deterministic graphs has been widely discussed in the literature. However, with the intrinsic uncertainty in many networks such as social, biological, and communication networks, it is of great importance to study truss decomposition in a probabilistic context, but this has received much less attention till now. Furthermore, due to computation challenges of truss decomposition in probabilistic graphs, the state-of-the-art approaches are not scalable to large graphs.

Given a user-defined threshold, we are interested in finding all the maximal subgraphs which are a k-truss with high probability. The most important challenge, which distinguishes truss decomposition in probabilistic graphs from deterministic graphs, is computing tail probabilities of edge supports. We employ a special version of the Central Limit Theorem (CLT) to obtain the tail probabilities efficiently. Based on our CLT approach we propose a peeling algorithm for truss decomposition of a probabilistic graph that scales to very large graphs and offers significant improvement over state-of-the-art. Our extensive experimental results confirm the scalability and efficiency of our approach.

1 INTRODUCTION

Probabilistic graphs are graphs in which each edge has an existence probability [2]. Many real-world networks, such as social, trust, communication, and biological networks, feature uncertainty and thus can be modeled as probabilistic graphs.

Dense subgraph mining is an important way to analyze the structure of networks [7]. A popular notion of cohesive graph is the k-truss, which is defined as a maximal subgraph in which each edge participates in at least $(k-2)$ triangles within that subgraph. The k-truss features a variety of applications [5]. For instance, k-truss is a useful tool for visualization of complex networks [12]. Also, k-trusses are the basis of several community models [8]. It is thus important to discover k-trusses in probabilistic graphs.

Truss decomposition in deterministic graphs is a straightforward task and has been broadly studied in the literature [4, 9, 11]. However, in probabilistic graphs, truss computation is challenging and has received much less attention. We use the notation of local probabilistic (k, η)-truss introduced in [5], and will be explained in more detail in the next section.

Challenges and contributions. Probabilistic truss decomposition is associated with significant challenges due to intrinsic uncertainty in probabilistic graphs. Thus, the general idea of iterative edge removal in deterministic graphs does not work by itself in probabilistic graphs. For instance, counting the number of triangles which contain an edge is straightforward in deterministic graphs. But, in probabilistic graphs, the triangles in which an edge might participate have combinatorial nature [5]. As a result, the most difficult task is computing edge support probabilities efficiently. This becomes particularly important when the input graph is huge. In [5], support probability of an edge $e = (u, v)$ is computed using dynamic programming (DP), which has a complexity of $O((\min\{d(u), d(v)\})^2)$, where $d(u)$ and $d(v)$ is deterministic degree of u and v, respectively. Unfortunately, the values of $d(u)$ and $d(v)$ can be in the millions in many real-world social and web networks, and quadratic time complexity of DP makes it impractical for huge graphs.

Realizing the fact that each triangle in probabilistic graph can be defined as a Bernoulli random variable with an existence probability, we design a novel approach based on Lyapunov's special version of the Central Limit Theorem [6] to approximate probability distribution of the support of an edge. We show that the proposed approximation is accurate for our problem when the number of triangles is big. In addition, we derive an error bound on the approximation to ensure that the output probabilities are very close to the values obtained through exact computation.

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

In summary, our contributions are as follows.

- We introduce an efficient approach based on Lyapunov's central limit theorem to compute support probabilities of edges in the input graph (Section 3.1).
- Using theoretical analysis, we obtain error bound of the approximation, which shows that the higher the number of triangles, the higher the accuracy of the approximation results (Corollary 1).
- We develop a peeling algorithm based on recursive edge deletions (Section 3.2), which, by utilizing central limit theorem and additional data structures, is able to calculate truss decomposition in very big probabilistic graphs not possible with the pure DP approach (Section 4).

2 BACKGROUND

Trusses in deterministic graphs. Let $G = (V, E)$ be an undirected graph with no self-loops. For a vertex v, let $N_G(v)$ be the set of v's neighbors in G. A triangle in the input graph is defined as a cycle of length 3, denoted by $\triangle_{uvw}$, where $u, v, w \in V$. Given an edge $e = (u, v)$, the *support of* e in G is the number of triangles that contain e. Formally, $\text{sup}_G(e) = |N_G(v) \cap N_G(u)|$.

The k-truss of G is defined as the maximal induced subgraph $T_k(G) = (V', E_{V'})$ in which each edge $e \in E_{V'}$ has support of at least $(k-2)$. The set of all k-trusses forms truss decomposition of G, where $2 \leq k \leq k_{\max}$, and $k_{\max}$ is the largest support of any edge.

Probabilistic graphs. A probabilistic graph is defined as $\mathcal{G} = (V, E, p)$, where V and E are as before and $p : E \rightarrow (0, 1]$ is a function that assigns existence probability $p(e)$ to edge e. In the most common probabilistic graph model [2], the existence probability of each edge is assumed to be independent of other edges.

To analyze probabilistic graphs, we use the concept of *possible worlds*, which are deterministic graph instances of $\mathcal{G}$. For each

© 2019 Copyright held by the owner/author(s). Published in Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), March 26-29, 2019, ISBN 978-3-89318-081-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

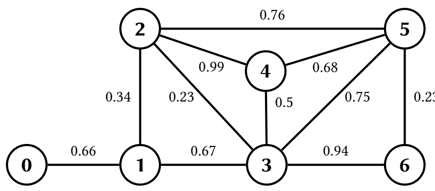

Figure 1: A probabilistic graph.

possible world $G = (V, E_G) \sqsubseteq \mathcal{G}$, where $E_G \subseteq E$, the probability of observing that possible world is obtained as follows: $\Pr(G) = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G}(1 - p(e))$.

Given an edge $e = (u, v)$, let $k_e = |N(u) \cap N(v)|$. We define the notion of *η-support*, denoted by $\eta\text{-}sup_{\mathcal{G}}(e)$, as the maximum k for which $\Pr_{G \sqsubseteq \mathcal{G}}[\sup_G(e) \geq k] \geq \eta$, where $k = 0, \ldots, k_e$, and the probability is taken over all the possible worlds $G \sqsubseteq \mathcal{G}$. In the rest of the paper, we use $\Pr[\sup_{\mathcal{G}}(e) \geq k]$ to denote $\Pr_{G \sqsubseteq \mathcal{G}}[\sup_G(e) \geq k]$.

For instance, consider Figure 1, edge $e = (2, 5)$, and $\eta = 0.5$. With the assumption that e exists (with probability $p(e) = 0.76$), edge e has support at least 2 with probability $0.76 \cdot (0.99 \cdot 0.68) \cdot (0.23 \cdot 0.75) = 0.0883$ (product of the probabilities that triangles $\triangle_{245}$ and $\triangle_{235}$ exist in a possible world), and it has support at least 1 with probability $0.76 \cdot \big(1 - (1 - 0.99 \cdot 0.68) \cdot (1 - 0.23 \cdot 0.75)\big) = 0.5545$ (complementary probability that none of the two triangles are in a possible world). Since 0.5545 is greater than the threshold η, the η-support of edge e is 1.

In probabilistic context, $\sup_{\mathcal{G}}(e)$ is a random variable which can take on integer values form zero to k_e. Furthermore, as k increases, the value of $\Pr[\sup_{\mathcal{G}}(e) \geq k]$ decreases.

Trusses in probabilistic graphs. In order to compute truss decomposition in probabilistic graphs we follow the local (k, η)-*truss* model defined in [5]:

Definition 1. *Let $\mathcal{G} = (V, E, p)$ be a probabilistic graph. Given threshold $\eta \in [0, 1]$, the local (k, η)-truss is the maximal induced subgraph $T_{(k,\eta)}(\mathcal{G}) = (V', E_{V'}, p)$ in which the η-support of each edge $e \in E_{V'}$ is at least $(k - 2)$. The set of all the local (k, η)-trusses forms the local truss decomposition of $\mathcal{G}$.*

Similar to deterministic case, local (k, η)-trusses are unique and nested into each other. The highest value of k for which e belongs to a local (k, η)-truss is called *η-truss* number or probabilistic trussness of e.

Computing η-supports using Dynamic Programming. Support probabilities are computed using $\Pr[\sup_{\mathcal{G}}(e) \geq k] = 1 - \sum_{i=0}^{k-1} \Pr[\sup_{\mathcal{G}}(e) = i]$. One way of calculating $\Pr[\sup_{\mathcal{G}}(e) = i]$ is to use dynamic programming as proposed in [5]. Given an edge $e = (u, v)$, time complexity of this method of computation is $O(k_e^2)$. Since $k_e \in O(\min\{d(u), d(v)\})$, where $d(u)$ and $d(v)$ are deterministic degree of u and v in $\mathcal{G}$, respectively, this method of computation is not practical when the minimum degree of two neighbors is big, say over 100, which is quite common in all our datasets. In addition, web-scale graphs have millions of such edges, and as a result, if DP is applied to each edge the total processing time increases considerably. In the next section we propose an alternative approach for fast computation of η-support of an edge e using Lyapunov central limit theorem.

3 PEELING ALGORITHM FRAMEWORK USING CENTRAL LIMIT THEOREM

We describe a CLT-focused algorithm to compute truss decomposition in probabilistic graphs. The pseudocodes/proofs are omitted due to page limit.

3.1 Computing η-Supports using Central Limit Theorem

We first show how a special version of Central Limit Theorem (CLT) can be applied to accurately estimate $\Pr\big[\sup_{\mathcal{G}}(e) \geq k\big]$. Then, we show theoretical bounds on the accuracy of this approximation. Specifically, we show that CLT can produce a very accurate approximation to tail probabilities of the support edge.

Based on CLT, the distribution of properly scaled sum of a sequence of random variables converges to normal distribution under specific conditions. In this paper, we consider a variant called Lyapunov CLT that can be applied when random variables are independent, but not necessarily identically distributed. Lyapunov CLT can be formally stated in the following:

Theorem 3.1. ***Lyapunov CLT.*** *Let $\zeta_1, \zeta_2, \cdots, \zeta_n$ be a sequence of independent, but non-identically distributed random variables, each with finite expected value μ_k and variance σ_k. Let*

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

Lyapunov CLT states that if

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

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

Equation (2) is called Lyapunov's condition which in practice is usually tested for the special case $\delta = 1$. The proof for this theorem can be found in [3].

Computing η-supports using Lyapunov CLT. Given an edge e, to compute $\Pr\big[\sup_{\mathcal{G}}(e) \geq k\big]$ we assume that e exists. Thus, the true edge support probability is obtained by multiplication of $\Pr\big[\sup_{\mathcal{G}}(e) \geq k\big]$ with $p(e)$.

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

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

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

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

Let $p_i = p(u, t_i) \cdot p(v, t_i)$. Since X_i is a Bernoulli random variable, we know that $\mathrm{E}[X_i] = \mu_i = p_i$ and $\mathrm{Var}[X_i] = p_i(1 - p_i)$. Since $\sup_{\mathcal{G}}(e) = \sum_{i=1}^{k_e} X_i$, we have:

$$\Pr[\sup_{\mathcal{G}}(e) \geq k] = \Pr\left[\sum_{i=1}^{k_e} X_i \geq k\right]. \tag{3}$$

Bernoulli random variables X_i's are independent, but may not be identically distributed. Thus, if condition (2) is satisfied and if k_e is large enough, we can conclude that $\frac{1}{s_{k_e}} \sum_{i=1}^{k_e}(X_i - \mu_i)$ has standard normal distribution, where $s_{k_e} = \sqrt{\sum_{i=1}^{k_e} p_i(1 - p_i)}$. In order to compute the right-hand side of equation (3), we can subtract $\sum_{i=1}^{k_e} \mu_i$ from both sides of the inequality, and then divide by s_{k_e} which results in:

$$\Pr\Big[\sum_{i=1}^{k_e} X_i \geq k\Big] = \Pr\Big[\frac{1}{s_{k_e}} \sum_{i=1}^{k_e}(X_i - \mu_i) \geq \frac{1}{s_{k_e}}(k - \sum_{i=1}^{k_e} \mu_i)\Big]. \tag{4}$$

Using Lyapunov CLT and setting

$$Z = \frac{1}{s_{k_e}} \sum_{i=1}^{k_e}(X_i - \mu_i), \tag{5}$$

we can conclude that Z has standard normal distribution. Thus

$$\Pr\big[\sup(e) \geq k\big] \cong \Pr\big[Z \geq z\big], \tag{6}$$

where $z = \frac{1}{s_{k_e}}(k - \sum_{i=1}^{k_e} \mu_i)$. Using the complementary cumulative distribution function [13] of standard normal variable Z, we can simply evaluate the right-hand side of Equation (6) for each value of k. Thus, to find the η-support for an edge, we start with $k = 1$, approximate $\Pr[\sup(e) \geq k]$ using Lyapunov CLT, and find the maximum k for which the probability multiplied by $p(e)$ is above threshold η. For an edge e, the obtained value of k, which can be in range from one to k_e, is set as initial η-support for that edge. Given an edge $e = (u, v)$, time complexity of finding η-support is $O(k_e)$, where $k_e = |N(u) \cap N(v)|$. Recall that DP required $O(k_e^2)$ for this step.

In the following we show that Lyapunov's condition in Theorem 3.1 is satisfied for our problem. We set $\delta = 1$ in Equation (2) to show that this condition holds for a sequence of non-identically distributed Bernoulli random variables.

Theorem 3.2. *Given a sequence of random variables $X_i \sim$ Bernoulli(p_i), where $1 \leq i \leq n$, the Lyapunov's condition (2) for $\delta = 1$ is satisfied whenever $s_n^2 = \sum_{k=1}^{n} p_k(1 - p_k) \to \infty$.*

Accuracy of the Approximation. Using Berry–Esseen theorem [14], in the following corollary we show how to obtain an upper-bound on the maximal error while approximating the true distribution of the sum of X_i's with the normal distribution.

Corollary 1. *For each edge e in the probabilistic graph $\mathcal{G}$ with X_i's being Bernoulli random variables defined as above in this section, where $i = 1, \ldots, k_e$, the error bound on the approximation of the right-hand side of Equation (6) to the standard normal distribution is given as follows:*

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

3.2 Peeling Algorithm (PA)

In [10], a peeling algorithm was proposed to calculate the k-truss in deterministic graphs. While the algorithm is not applicable to probabilistic graphs, its optimized array-based data structures for storing edge information of the graph are useful. Our new peeling algorithm, termed as *CLT_based-PA* algorithm, is built on the same array-based data structures but utilizes central limit theorem to compute and update support probabilities of edges which participate in more than 100 triangles.[1]

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

In initial support computation step, the η-support of each edge e is computed using CLT and Equation (6), if k_e is greater than 100. Otherwise, DP can be used safely. The details on DP approach can be found in [5]. The initial phase can be executed in parallel, since probabilistic support of each edge can be computed independently of other edges.

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

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

At the third step, the η-support of (u, w) and (v, w) is updated if their η-supports are greater than e's η-support. In the updating part, if the number of remaining triangles which contain edges (u, w) and (v, w) is greater than 100, we perform update phase using CLT approach. Otherwise, we apply DP. Since the η-support has been changed, we change the position of edges (v, w) and (u, w) in *sortedEdge* array in constant time [10]. The algorithm continues until all the edges in the graph are removed. Then, the trussness of each edge is obtained by adding 2 to the final η-support.

4 EXPERIMENTS

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

The statistics for the datasets are shown in Table 1. We obtained flicker, dblp, and biomine from the authors of [2], and the rest of the datasets from Laboratory of Web Algorithmics.[2] Each horizontal line in the table categorizes the datasets according to their size, small (S), medium (M), and large (L). We use the Webgraph framework [1] to store these datasets. The flickr, dblp, and biomine datasets already contained edge probability values. For the other datasets we generated probability values uniformly distributed in $[0, 1]$.

[1]This value was chosen because it was large enough to keep the approximation error obtained from Corollary 1 small.
[2]http://law.di.unimi.it/datasets.php

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

Table 1: Dataset Statistics

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

Table 2: Running times (sec) of DP_Pure and CLT_based–PA. The column "gain (%)" reports the gain of CLT_with_DP algorithm over DP_Pure algorithm. We use N.P. for "Not Possible".

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

Table 3: Maximum η-support, maximum probabilistic trussness, value of the threshold η.

Table 2 represents the running times of our proposed approach, versus the running times of the state-of-the-art, which uses dynamic programming (DP) only and is referred as *DP_Pure*. The last column shows the gain of *CLT_based–PA* algorithm over *DP_Pure*. For ljournal-2008, we present the results for different values of η ranging from 0.1 to 0.5. However, for the other datasets, we only show the results for $\eta = 0.1$, and omit results for $\eta = 0.2, \dots, 0.5$, since they are similar to those for $\eta = 0.1$ and their performance trend is similar to what we see for ljournal-2008. As can be seen, *CLT_based–PA* algorithm is significantly faster than *DP_Pure*. For instance, for biomine, which is a large dataset, the gain of our algorithm is 67 percent, making *CLT_-based–PA* truss decomposition algorithm three times faster than *DP_Pure*.

CLT_based–PA produced the results in about 1.5 minutes and about 34 seconds for flickr and dblp, respectively. Although flickr is smaller than dblp in terms of the number of vertices and edges, its probabilistic maximum truss is much greater at value 47 compared to 14 in dblp. We represent the maximum probabilistic truss and maximum probabilistic support in Table 3. *These values are the same as those obtained by DP_Pure.* On biomine which is a large dataset, our proposed algorithm completed in about 43 minutes; which is quite impressive. In contrast, *DP_Pure* produced the results in more than 2 hours. The running time for ljournal-2008 increases, which is quite reasonable, because this graph has 49 million edges with probabilistic support of 1030 when $\eta = 0.1$. The same argument holds for cnr-2000 which has probabilistic support of 4672 which is significantly big. *DP_Pure wasn't able to run to completion in our machine after one day.*

Effect of η values. The running time of both algorithms increases as η becomes small. This is because as η decreases, the chance for support probabilities to pass the threshold increases, resulting in larger values of η-supports. This is particularly important in performance evaluation of *DP_Pure* algorithm– as η decreases the DP algorithm approaches its worst case time complexity, $O(k_e^2)$, for an edge e. As a result, for larger values of η, the running time of *DP_Pure* improves, but is still by far slower than *CLT_based–PA*. In terms of the effect of η on truss decomposition and support values, as can be seen in Table 3, the maximum truss and maximum initial probabilistic support decrease as η increases. As before, we report the maximum truss, and the maximum η-support for ljournal-2008 for $\eta = 0.1, \dots, 0.5$, whereas for the other datasets we only show the values for $\eta = 0.1$.

5 CONCLUSIONS

We presented a peeling algorithm for computing truss decomposition in probabilistic graphs at web scale. Our peeling algorithm uses Lyapunov's central limit theorem to obtain the probabilistic support for an edge. Unlike the dynamic programming approach, the computation does not rely on incremental evaluation of support probabilities. In addition, it can efficiently update probabilistic support when a triangle is removed from the input graph, without the need of storing all the previously computed support probabilities. We evaluated our algorithm and showed that it is significantly faster than state-of-the-art for large datasets. For large and medium datasets our algorithm obtained approximately 50 percent gain over the proposed algorithm in the literature, completing the biomine dataset in less than one hour.

REFERENCES

[1] P. Boldi and S. Vigna. 2004. The webgraph framework I: compression techniques. In *Proc. WWW'04*. ACM, 595–602.
[2] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. 2014. Core decomposition of uncertain graphs. In *Proc. SIGKDD*. ACM, 1316–1325.
[3] H. Cramér. 1946. *Mathematical Methods of Statistics*. PUP.
[4] X. Huang, H. Cheng, L. Qin, W. Tian, and J. Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proc. SIGMOD*. ACM, 1311–1322.
[5] X. Huang, W. Lu, and L. V. Lakshmanan. 2016. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proc. SIGMOD*. ACM, 77–90.
[6] H. Kobayashi, B.L. Mark, and W. Turin. 2011. *Probability, Random Processes, and Statistical Analysis*. Cambridge University Press.
[7] V. Lee, N. Ruan, R. Jin, and Ch. Aggarwal. 2010. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*. Springer, 303–336.
[8] Y. Li, T. Kuboyama, and H. Sakamoto. 2013. Truss decomposition for extracting communities in bipartite graph. In *Proc. IMMM*. 76–80.
[9] J. Wang and J. Cheng. 2012. Truss decomposition in massive networks. *Proc. VLDB* 5, 9 (2012), 812–823.
[10] J. Wu, A. Goshulak, V. Srinivasan, and A. Thomo. 2018. K-Truss Decomposition of Large Networks on a Single Consumer-Grade Machine. In *Proc. ASONAM*. IEEE, 873–880.
[11] Y. Zhang and S. Parthasarathy. 2012. Extracting analyzing and visualizing triangle k-core motifs within networks. In *Proc. ICDE*. IEEE, 1049–1060.
[12] F. Zhao and A. Tung. 2012. Large scale cohesive subgraphs discovery for social network visual analysis. In *Proc. VLDB*, Vol. 6. VLDB Endowment, 85–96.
[13] D. Zwillinger and S. Kokoska. 2000. *CRC Standard probability and statistics tables and formulae*. Chapman and Hall/CRC, USA.
[14] D. Zwillinger and S. Kokoska. 2013. *Probability Theory*. Springer-Verlag, London.