Truss Decomposition on Large Probabilistic Networks using H-Index

Fatemeh Esfahani
University of Victoria
Victoria, BC, Canada
esfahani@uvic.ca

Mahsa Daneshmand
University of Victoria
Victoria, BC, Canada
mahsad@uvic.ca

Venkatesh Srinivasan
University of Victoria
Victoria, BC, Canada
srinivas@uvic.ca

Alex Thomo
University of Victoria
Victoria, BC, Canada
thomo@uvic.ca

Kui Wu
University of Victoria
Victoria, BC, Canada
wkui@uvic.ca

ABSTRACT

Truss decomposition is a popular approach for discovering cohesive subgraphs. However, truss decomposition on probabilistic graphs is challenging. State-of-the-art either do not scale to large graphs or use approximation techniques to achieve scalability. We present an exact and scalable algorithm for truss decomposition of probabilistic graphs. The algorithm is based on progressive tightening of the estimate of the truss value of each edge based on h-index computation and novel use of dynamic programming. Our proposed algorithm (1) is significantly faster than state-of-the-art and scales to much larger graphs, (2) is progressive by allowing the user to see near-results along the way, (3) does not sacrifice the exactness of final result, and (4) achieves all these while processing only an edge and its immediate neighbors at a time, thus resulting in smaller memory footprint. Our extensive experimental results confirm the scalability and efficiency of our algorithm.

KEYWORDS

Probabilistic Graphs, Truss Decomposition, Social Networks

1 INTRODUCTION

An important category of problems in network analysis is detecting dense or cohesive components in graphs. Studying cohesive subgraphs can reveal important information about connectivity, centrality, and robustness of the network. Among different notions of cohesive subgraphs in the literature, the notion of *truss* is particularly suited to extracting a hierarchical structure of cohesive subgraphs [23].

In deterministic graphs, the k-truss of a graph G is defined as the largest subgraph in which each edge is contained in at least k triangles (or $(k-2)$ in some works). The highest value of k for which an edge is part of the k-truss is called truss value of that edge. The collection of all k-trusses for different values of k forms the truss decomposition of the graph. This is a hierarchical structure because k-truss is contained in $(k-1)$-truss for all $k > 1$. Truss decomposition has been used in several important applications such as visualization of complex networks [30] and community modeling [10].

Truss decomposition in deterministic graphs has been widely studied in the literature (cf. [10, 22–24, 29]). However, with the intrinsic uncertainty in many networks such as social, biological, and communication networks (cf. [6, 26]), it is of great importance to study truss decomposition in a probabilistic context. However, in probabilistic graphs, truss computation is challenging and has received much less attention. Here, we present an efficient algorithm for computing truss decomposition in probabilistic graphs; the graphs in which each edge has a probability of existence independent of the other edges. We use the notion of (k, η)-truss introduced in [11]. Specifically, we aim to compute the largest subgraph in which each edge is contained in at least k triangles within that subgraph with probability no less than a user specified threshold η. The threshold η defines the desired level of certainty of the output trusses.

Challenges and contributions. The standard approach to computing k-truss decomposition is the edge peeling process, which is based on continuously removing edges with less than k triangles (cf. [11]). This process is repeated after incrementing k until no edges remain [21], which results in finding all k-trusses for different values of k. Edge peeling is associated with a major drawback: the edges have to be kept sorted by their current triangle support (count) at all times which requires maintaining global information of the graph at each step of the algorithm. This affects the scalability of the algorithm considerably. Edge peeling becomes even more challenging in probabilistic graphs, because triangle counting in such graphs has a combinatorial nature [11]. That is, each edge should have enough probability to participate in at least k triangles in the input graph. This probability is called support probability of the edge. The exact computation of the support probability of an edge $e = (u, v)$ is done using dynamic programming (DP) in [11]. The process is repeated each time the edge loses a neighbor during the peeling process. This process does not scale as it involves many recomputations, especially when there are edges with many neighbors. An approximation method is proposed in [8], which is also a peeling algorithm. However, it addresses the problem by *approximating the support probability* using statistical techniques, thus sacrificing the exactness of solution, but without providing approximation guarantees. This leads us to ask whether there is an *exact* and *scalable* approach to truss decomposition in probabilistic graphs.

We answer the above question positively by introducing an algorithm which extends the iterative h-index computation, recently introduced for deterministic graphs in [21], to probabilistic graphs.

In deterministic graphs, triangle support of the edges are obtained at the beginning, and each edge computes the h-index value

for the list of its neighbors' triangle supports. Neighbors of an edge are those edges which form a triangle with the given edge. This process is repeated on these values until convergence to truss values occurs. Upon termination, the final h-index value of each edge equals its truss value. The authors in [21] proves that convergence of support values to the truss values is guaranteed.

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

- We propose an algorithm based on h-index updating which works for probabilistic graphs. Our proposed algorithm is exact with respect to final result, but also progressive allowing the user to see near-results along the way, and it works by processing only one edge and its immediate neighbors at a time, resulting in smaller memory footprint in practice.
- While proving the correctness of the algorithm, we obtain an upper-bound on the number of iterations that the algorithm needs for convergence. It shows that the convergence to truss values can be obtained after a finite number of iterations.
- We evaluate the performance of our approach on a wide range of datasets. Our experimental results confirm the scalability and efficiency of our algorithm, significantly outperforming the exact algorithm in [11] for large datasets. Furthermore, comparisons with the approximate algorithm of [8] show that the running time of our proposed algorithm is very close to that of the approximate algorithm. It is indeed surprising that we can achieve efficiency without sacrificing the exactness of the solution.

2 RELATED WORK

In the literature, much research has been done in the area of mining and querying probabilistic graphs [5, 9, 12, 13, 15, 18, 19, 27, 28, 31], such as the k-nearest neighbor search over probabilistic graphs [20], uncertain graph sparsification [16], and mining top k maximal cliques in probabilistic graphs [32].

Recently k-truss has attracted a lot of attention due to its cohesive structure and the fact that it can be used to compute other definitions of dense subgraphs, such as k-clique. In deterministic graphs, truss decomposition has been studied extensively in different settings (cf. [4, 10, 23, 25]).

For probabilistic graphs, the notion of (k, η)-truss is introduced by Huang, Lu, and Lakshmanan in [11]. Their algorithm for computing (k, η)-truss is based on iterative edge peeling and uses dynamic programming for computing support probability of edges. While this algorithm runs in polynomial time, it does not scale well to large graphs, especially those having a high maximum vertex degree. To address this problem, Esfahani et al. in [8] propose an *approximate* algorithm, also based on edge peeling, which uses statistical arguments to replace the DP part in the algorithm of [11].

In contrast, in the current paper, we propose instead an *exact* algorithm which does not use peeling at all and scales to large probabilistic graphs.

[11] also proposes the notion of global (k, η)-truss based on the probability of each edge belonging to a connected k-truss in a possible world. An algorithm based on sampling is proposed in [11] to find global (k, η)-trusses. This notion of probabilistic truss decomposition falls in the category of #P-hard problems and is not in the scope of our paper.

Probabilistic core decomposition is studied in [3, 7, 14, 17]. Core decomposition can also produce cohesive subgraphs, albeit less so than truss decomposition. In general, truss decomposition is harder to compute than core decomposition.

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

Table 1: Main Notations

3 BACKGROUND

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

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

Probabilistic graphs. A probabilistic graph is a triple $\mathcal{G} = (V, E, p)$, and is defined over a set of vertices V, a set of edges E and a probability function $p : E \rightarrow (0, 1]$ which maps every edge $e \in E$ to an existence probability $p(e)$. In the most common probabilistic graph model [3], 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 $\mathcal{G}$. In each possible world only a subset of edges appears. For each 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:

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

Example 1. *Consider probabilistic graph $\mathcal{G}$ in Figure 1a. A possible world G of $\mathcal{G}$ is shown in Figure 1b. $Pr(G) = 0.5 \cdot 0.4 \cdot 0.3 \cdot 0.65 \cdot (1 - 0.25) = 0.02925$.*

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

$$\Pr[\sup_{\mathcal{G}}(e) = t] = \sum_{G \sqsubseteq \mathcal{G}} \Pr[G] \cdot \mathbb{1}(\sup_G(e) = t), \quad (2)$$

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

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

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

Definition 1. *Let η be a user defined threshold.*

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

Proposition 1. *(k, η)-truss of $\mathcal{G}$ is the subgraph of $\mathcal{G}$ containing all and only the edges e in $\mathcal{G}$ with $\kappa_\eta(e) \geq k$.*

In this paper (as in [8, 11]) we focus on finding the truss values of the edges in an input graph. Then (k, η)-truss for any k is constructed by collecting all edges e with $\kappa_\eta(e) \geq k$.

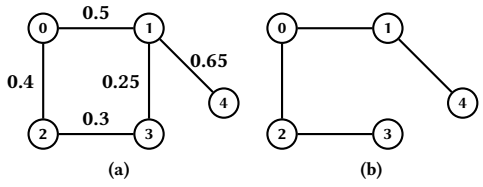

Figure 1: a) Probabilistic graph $\mathcal{G}$, b) A possible world G of $\mathcal{G}$.

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

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

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

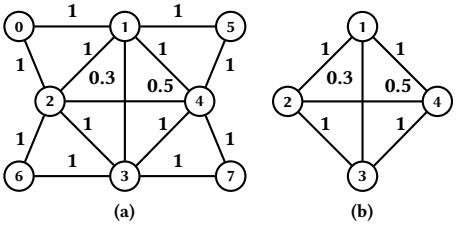

Figure 2: a) Probabilistic graph $\mathcal{G}$, b) (2,0.15)-truss $\mathcal{F}$ of $\mathcal{G}$.

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

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

Computing $\Pr[\sup_{\mathcal{G}}(e) = i]$ for different values of i can be done using dynamic programming as proposed in [11].

4 ALGORITHM FRAMEWORK

Here we propose an algorithm based on h-index updating, which has been introduced in the context of *deterministic* graphs by [21]. Given a set of real numbers, the h-index of the set is defined as the largest number h such that there are at least h elements in the set that are equal to h or higher. For instance, the h-index of the set $\{1, 2, 3, 3, 5\}$ is 3 because the set includes three numbers no less than 3.

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

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

In our algorithm, we refer to this process as *Phase I*. This phase corresponds to the h-index based algorithm of [21] for the deterministic case. In deterministic graphs, once the process terminates, the h-index of each edge becomes equal to the truss value of that edge. However, we show that this does not solve our problem.

Deterministic h-index updating, *Phase I*. In the following we provide explanation of *Phase I* of our algorithm, which is based on [21].

Definition 2. *Given a set K of natural numbers, $\mathcal{H}(K)$ is the largest $k \in \mathbb{N}$ such that at least k elements of K are greater than or equal to k.*

Algorithm 1 Phase I

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

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

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

Example 3. To illustrate how h-index works on deterministic graphs, we refer to Figure 3. The figure shows a deterministic graph with 6 vertices. Initially, the triangle counts of all the edges are computed and are set as initial values on the h-index of the edges. Let h_0 be the list of these initial values, which are shown with blue color in the figure. Then, the algorithm starts updating the h-indices based on the initial values. Let h_1 be the list of updated values at this step (red). Edge $e = (0, 2)$, for instance, participates in 4 triangles and *in each of them*, the algorithm finds the edge neighbor to $(0, 2)$ with minimum h_0 value and records this value in an array. Then the algorithm updates the h-index of e. So, $L = \{\min(h_0(0,1), h_0(1,2)), \min(h_0(0,3), h_0(2,3)), \min(h_0(0,4), h_0(2,4)), \min(h_0(0,5), h_0(2,5))\} = \{1, 2, 3, 2\}$. As a result, $h_1(0,2) = \mathcal{H}(L) = 2$. The h-index of edges $(0, 4)$ and $(2, 4)$ are updated similarly. No more updates happen in the next iteration. Since the given graph is a deterministic graph, at the end, each edge obtains its truss value (green).

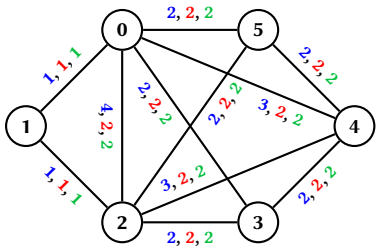

Figure 3: A running example of h-index algorithm on a deterministic graph.

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

Table 2: η-sup$_\mathcal{G}(e)$, values obtained by Phase I (Ph_I) and Phase II (Ph_II), respectively, truss values. $\eta = 0.2$ for Figure 2a.

Algorithm 2 Probabilistic h-index Truss (proHIT)

```
1:  function Probabilistic h-index updating(G, support, η)
2:      for all edge e ∈ E do
3:          h(e) ← η-sup_G(e),   scheduled[e] ← true
4:      Phase I (G, h, scheduled)          ▷ Deterministic h-index
5:      updated ← false                    ▷ True if any h(e) is updated
6:      while true do
7:          Phase II (G, h, scheduled)
8:          if updated is false then break
9:          else
10:             Phase I (G, h, scheduled), updated ← false
11:     for all edge e ∈ E do κ_η(e) ← h(e)
12:     return array of κ_η(·)
```

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

For an example, consider Figure 2a and $\eta = 0.2$. In Table 2 we show the execution of our algorithm at each phase. The first column shows edges in the graph. The last shows their truss values. Column ph_I shows the h-index values at the end of *Phase I*. Initially h-index, $h(e)$, of each edge e is set to η-sup$_\mathcal{G}(e)$ (second column). Now, consider edge $e = (1, 2)$. For each triangle containing e, *Phase I* finds the minimum h-index of the two edges other than e in the triangle, and adds this minimum to a set L. For e, we have $L = \{\min(h(0,1), h(0,2)), \min(h(1,3), h(2,3)),$ $\min(h(1,4), h(2,4))\} = \{1, 2, 2\}$. Since there are two numbers on this list that are equal to 2, *Phase I* sets 2 as the h-index of edge e. Further execution of *Phase I* cannot produce any updates. However, the truss value of edge e is in fact 1 (see last column of Table 2) as we explain later in this section. Therefore, *Phase I* is not able to converge to the truss value of e. Nevertheless, we prove that *Phase I* can be used to provide upper-bounds to true truss values. The proof of this fact, Theorem 1, is presented in Section 5.

Algorithm 3 Phase II

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

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

The major steps of our algorithm, *probabilistic h-index truss* (*pro-HIT*), are summarized in Algorithm 2. At a high level, we maintain an array h indexed by edges where we initially store the h-index value for each edge. Then, we tighten up these values using *Phase I* and *Phase II*, and by the end of the iterations, we have the output truss values in array h.

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

Phase II of our approach is given in Algorithm 3. *This part crucially differentiates our approach from h-index based algorithms for deterministic graphs.*

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

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

$$\Pr\big[\sup(e) \geq h(e) \;\big|\; \Gamma\big] \geq \eta. \tag{3}$$

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

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

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

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

The main challenge in *Phase II* is efficient checking of condition 3 for different values of $h(e)$ until a proper value is obtained. For this, we introduce a modified dynamic programming (DP) process to avoid computation of these probabilities from scratch.

Modified DP. This process is invoked when we check the condition on line 6 of Algorithm 3.

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

The probability $\Pr[\sup(e) \geq H \;|\; \Gamma]$ is computed using DP [11]. However, Γ changes in each iteration of the while loop (see line 9). We would like to avoid the computation of $\Pr[\sup(e) \geq H \;|\; \Gamma]$ from scratch each time. It should be noted that the probability computation is valid if e exists, with existence probability $p(e)$. So, based on statistics we can write:

$$\Pr[\sup(e) \geq H \;|\; \Gamma] = p(e) \cdot \Pr[\sup(e) \geq H \;|\; \Gamma, e \text{ exists}], \tag{4}$$

Initially, the following probabilities are computed

$$\Pr[\sup(e) = 0 \;|\; \Gamma, e \text{ exists}], \cdots, \Pr[\sup(e) = H \;|\; \Gamma, e \text{ exists}], \tag{5}$$

We now cache these probabilities.

Given a Γ set, let $\Pr_{(H,\Gamma)} = \Pr[\sup(e) \geq H \;|\; \Gamma]$. For $H - 1$, we define $\mathcal{T}^{(H-1)} = \{\triangle_1, \cdots, \triangle_j\}$ to be the set of $\triangle$ triangles which contain e, and $\rho_\triangle = H-1$. Let Γ^{new} be the set of all triangles $\triangle$ which contain e, and have $\rho_\triangle \geq H - 1$. Clearly, $\Gamma^{new} = \Gamma \cup \mathcal{T}^{(H-1)}$. Now, we need to compute $\Pr_{(H-1,\Gamma^{new})}$ efficiently using the probabilities in Equation 5. For this, we only need to look at set $\mathcal{T}^{(H-1)}$, which is usually small (i.e., not more than 50 in our tested real graphs). As such, the computation is done very fast.

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

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

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

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

$$\begin{aligned}T(j, k) = p(u, w_l)p(v, w_l)T(j-1, k-1) \\ + (1 - p(u, w_l)p(v, w_l))T(j-1, k).\end{aligned} \tag{6}$$

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

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

Note. The central limit theorem can be used for approximating $\Pr[\sup(e) \ge H \mid \Gamma]$ as well as obtaining an estimate for initial probabilistic support of edges [8]. Approximation can make *proHIT* algorithm faster. However, in this work, we focus on proposing an exact algorithm for solving truss decomposition.

5 PROOFS OF CORRECTNESS

In this section, we present the proofs of correctness of our algorithm, *proHIT*, propsed in Section 4. In particular, we show that convergence can be obtained in a finite number of iterations. We start by showing that the values obtained by *Phase I* are upper-bounds on the truss values.

Theorem 1. *In every iteration, Phase I provides upper-bounds on truss values of edges in the input probabilistic graph.*

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

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

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

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

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

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

□

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

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

$$\Pr[\sup_{\mathcal{G}}(e) \ge \delta_\eta(\mathcal{G})] \ge \eta, \quad \forall e \in E(\mathcal{G}), \tag{7}$$

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

Proposition 2. *Given a subgraph $\mathcal{G}' \subseteq \mathcal{G}$ and an edge $e = (u, v)$ in $\mathcal{G}'$, let W and W' be the sets of all the triangles in $\mathcal{G}$ and $\mathcal{G}'$, respectively, which contain e. We have that $\Pr[sup(e) \ge k \mid W'] \le \Pr[sup(e) \ge k \mid W]$, where $k = 0, \cdots, k_e$. (As mentioned earlier, this is equivalent to $\Pr[sup_{\mathcal{G}'}(e) \ge k] \le \Pr[sup_{\mathcal{G}}(e) \ge k]$) [11].*

The following Lemma is a generalization of a property of truss values in deterministic graphs [21] to the probabilistic context.

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

$$\kappa_\eta(e) = \max_{\mathcal{G}' \subseteq \mathcal{G}} \delta_\eta(\mathcal{G}'), \tag{8}$$

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

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

Now, we show that $\kappa(e) \ge \max_{\mathcal{G}'} \delta_\eta(\mathcal{G}')$. We use proof by contradiction. Let $\mathcal{G}''$ be the largest subgraph of $\mathcal{G}$ that contains e and has $\delta_\eta(\mathcal{G}'') > \kappa_\eta(e)$. Based on Equation 7 we have $\Pr[\sup_{\mathcal{G}''}(e') \ge \delta_\eta(\mathcal{G}'')] \ge \eta$, for any edge $e' \in E(\mathcal{G}'')$, including e. Hence, $\mathcal{G}''$ is a $(\delta_\eta(\mathcal{G}''), \eta)$-truss and contains e. This is a contradiction by the definition of $\kappa_\eta(e)$ which is the largest value of k such that e is contained in a (k, η)-truss. □

Following [21], we define the concept of degree (support) levels of edges in a probabilistic graph. First, we start with some technical definitions. Let $\mathcal{G}$ be a probabilistic graph, and η be a user-defined threshold. Also, let $C(\mathcal{G})$ be the set of edges and their containing

triangles. We define the following features for edges and triangles in $C(\mathcal{G})$:

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

Remark. We could have created two separate sets for edges and triangles, but doing so would significantly complicate the notation and its use in the proof as maintaining the relationship between these two sets would be cumbersome. This definition of $C(G)$ is chosen purely for notational convenience and is similar to the definition used in [21] where it is defined as the set of all r-cliques and s-cliques.

Definition 3. ***Degree Levels.*** *We define degree levels in a recursive way in a probabilistic graph $\mathcal{G}$. Let set L_i denote the i-th degree level. L_0 is defined as the set of edges e which have minimum probabilistic support in $C(\mathcal{G})$. L_1 is defined as the set of edges which have minimum probabilistic support in $C(\mathcal{G}) \setminus L_0$, and so on. In general, L_i contains the set of edges which have minimum probabilistic support in $C(\mathcal{G}) \setminus \bigcup_{j<i} L_j$. The maximum value of i for which L_i can be non-empty is equal to $k_{\max,\eta}$. We recall that $k_{\max,\eta} = \max_e\{\eta\text{-}sup_{\mathcal{G}}(e)\}$.*

Theorem 2. *Given integers i and j such that $i \leq j$ and a threshold η, for any $e_i \in L_i$ and $e_j \in L_j$, $\kappa_\eta(e_i) \leq \kappa_\eta(e_j)$.*

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

We prove the convergence of our proposed algorithm using ideas similar to the proof of deterministic h-index algorithm in [21]. In Theorem 1 we showed that *Phase I* provides upper-bounds on truss values of the input probabilistic graph. In the following, we prove that upper-bounds are monotonically non-increasing and are lower-bounded by truss values.

Theorem 3. *For all t and all edges e in $\mathcal{G}$, we have **(1)** $h_{t+1}(e) \leq h_t(e)$, **(2)** $h_t(e) \geq \kappa_\eta(e)$, where by $h_t(e)$ we denote the h-index of e after the t-th iteration of Phase I and Phase II together.*

Proof. **(1)** We prove this by induction on t. Initially, when $t = 0$, $h_0(e)$ is equal to $\eta\text{-sup}_{\mathcal{G}}(e)$. Let $h_1^p(\cdot)$ be the processed values after completion of *Phase I* at iteration 1. As shown in [21], throughout *Phase I*, the upper-bounds can only decrease, so $h_1^p(e) \leq h_0(e)$, for each edge e. The $h_1^p(\cdot)$ values are passed to *Phase II*. The block of steps 6-9 of *Phase II* (Algorithm 3) checks all the values equal or less than $h_1^p(e)$ for each edge e, and finds the maximum value for which the condition in line 6 holds. Let $h_1(e)$ be the obtained maximum value. Thus, we have $h_1(e) \leq h_1^p(e) \leq h_0(e)$. Assume the property is true up to t. For iteration $t+1$, *Phase I* needs to process the values $h_t(\cdot)$ obtained from the previous iteration (i.e. t) by *Phase II*. Let $h_{t+1}^p(e)$ be the processed values after completion of *Phase I* at iteration $t+1$. For an edge e, by the induction hypothesis, and monotonicity of *Phase I* itself [21], we have $h_{t+1}^p(e) \leq h_t(e) \leq h_{t-1}(e)$. Then, this value is passed through *Phase II*. As discussed above, this value is processed using lines 6-9 in Algorithm 3 which make sure that $h_{t+1}(e) \leq h_t^p(e)$. Thus, we have $h_{t+1}(e) \leq h_t(e)$.

(2) We prove the property by induction on t. For $t = 0$, $h_0(e) = \eta\text{-sup}_{\mathcal{G}}(e) \geq \kappa_\eta(e)$. Let us assume that for t, $h_t(e) \geq \kappa_\eta(e)$. Now, we focus on the computation of $h_{t+1}(e)$. Using the induction step and the fact that *Phase I* provides an upper-bound on $\kappa_\eta(e)$ for each edge e (please refer to Theorem 1), we can write: $h_{t+1}^p(e) \geq \kappa_\eta(e)$. Consider the computation of $h_{t+1}(e)$ by *Phase II* which is based on the value produced by *Phase I* (i.e. $h_{t+1}^p(e)$). Let $\mathcal{F}$ be $(\kappa_\eta(e), \eta)$-truss which contains e. Also, let S be the set of all supporting triangles $\triangle$ in $\mathcal{F}$ for edge e, such that $\forall e', e'' \neq e \in \triangle$, $\min(\kappa_\eta(e'), \kappa_\eta(e'')) \geq \kappa_\eta(e)$. Using the property of truss value we know that $\Pr[\sup(e) \geq \kappa_\eta(e) \mid S] \geq \eta$. To obtain $h_{t+1}(e)$, *Phase II* checks the condition $\Pr[\sup(e) \geq h_{t+1}^p(e) \mid \Gamma] \geq \eta$ (line 6, Algorithm 3), where Γ is the set of all the triangles that contain e, and is detected by *Phase II* since $\rho_\triangle \geq h_{t+1}^p(e)$, for each $\triangle \in \Gamma$, where $\rho_\triangle$ is the minimum h-index value of the edges other than e in $\triangle$ (line 19, Algorithm 3). If $\Pr[\sup(e) \geq h_{t+1}^p(e) \mid \Gamma] \geq \eta$ holds, then $h_{t+1}(e) = h_{t+1}^p(e) \geq \kappa_\eta(e)$. Otherwise, all the k values smaller than $h_{t+1}^p(e)$ are checked. In the worst case, consider the computation of the probability when k becomes equal to $\kappa_\eta(e)$. Let Γ be the updated set to contain all $\triangle$ with $\rho_\triangle \geq k$. We claim that $S \subseteq \Gamma$. For each triangle $\triangle \in S$, and $\forall e', e'' \neq e \in \triangle$, we have $\kappa_\eta(e'), \kappa_\eta(e'') \geq \kappa_\eta(e)$. In addition, based on Theorem 1, $h_{t+1}^p(e') \geq \kappa_\eta(e') \geq \kappa_\eta(e) = k$, and $h_{t+1}^p(e'') \geq \kappa_\eta(e'') \geq \kappa_\eta(e) = k$. Thus, $\rho_\triangle = \min(h_{t+1}^p(e'), h_{t+1}^p(e'')) \geq k = \kappa_\eta(e)$, which results in $\triangle \in \Gamma$. Using Proposition 2, $\Pr[\sup(e) \geq k \mid \Gamma] \geq \Pr[\sup(e) \geq k \mid S]$. If for $k = \kappa_\eta(e)$, $\Pr[\sup(e) \geq \kappa_\eta(e) \mid \Gamma] < \eta$, then $\Pr[\sup(e) \geq \kappa_\eta(e) \mid S] < \eta$, which is a contradiction with the definition of $\kappa_\eta(e)$. As a result, we should have $\Pr[\sup(e) \geq \kappa_\eta(e) \mid \Gamma] \geq \eta$. Thus, $h_{t+1}(e) \geq \kappa_\eta(e)$. □

Theorem 4. *Given any level L_i, for all $t \geq i$, and $e \in L_i$, we have $h_t(e) = \kappa_\eta(e)$.*

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

$$\Pr[\sup(e_a) \geq r \mid \Gamma] \leq \Pr[\sup(e_a) \geq r \mid S_h], \text{ for any } r, \quad (9)$$

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

$$\Pr[\sup_{\mathcal{G}'}(e_a) \geq \delta_\eta(\mathcal{G}')] \geq \eta, \quad (10)$$

$$\Pr[\sup_{\mathcal{G}'}(e_a) \geq r'] < \eta, \text{ for any } r' > \delta_\eta(\mathcal{G}'), \quad (11)$$

The above equations are based on the definition of probabilistic support of an edge: $\eta\text{-sup}_{\mathcal{G}'}(e_a) = \max_k\{\Pr[\sup_{\mathcal{G}'}(e_a) \geq k] \geq \eta\}$. By definition of S_h, edges contained in the triangles of S_h are part of $L' = E(\mathcal{G}')$. Thus, triangles in S_h are contained in $\mathcal{G}'$. As mentioned earlier, since computation of $\Pr[\sup_{\mathcal{G}'}(e_a) \geq r']$ is done by considering triangles in S_h, the values of $\Pr[\sup_{\mathcal{G}'}(e_a) \geq r']$ and $\Pr[\sup(e_a) \geq r' \mid S_h]$ are the same. Therefore, $\Pr[\sup(e_a) \geq r' \mid S_h] < \eta$. Combining this with Equation 9, for $r > \delta_\eta(\mathcal{G}')$ we obtain:

$$\Pr[\sup(e_a) \geq r \mid \Gamma] \leq \Pr[\sup(e_a) \geq r \mid S_h] < \eta, \quad (12)$$

Since $\Pr[\sup(e_a) \geq r \mid \Gamma] < \eta$, the algorithm checks r values less than or equal to $\delta_\eta(\mathcal{G}')$, thus $h_{t+1}(e_a) \leq \delta_\eta(\mathcal{G}')$. In addition, based on Lemma 1, we have $\delta_\eta(\mathcal{G}') \leq \kappa_\eta(e_a)$. Thus, $h_{t+1}(e_a) \leq \kappa_\eta(e_a)$. On the other-hand, based on property (2) in Theorem 3, we have $h_{t+1}(e_a) \geq \kappa_\eta(e_a)$. Combining $h_{t+1}(e_a) \leq \kappa_\eta(e_a)$ and $h_{t+1}(e_a) \geq \kappa_\eta(e_a)$, we conclude that $h_{t+1}(e_a) = \kappa_\eta(e_a)$. Since e_a was an arbitrary edge in L_{i+1}, this concludes the proof by induction. □

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

Corollary 1. *Given a probabilistic graph $\mathcal{G}$, and threshold η, let l be the maximum value for the degree level, such that $L_l \neq \emptyset$. There exists some $t \leq l$ such that $h_t(e) = \kappa_\eta(e)$, for all edges.*

6 COMPLEXITY ANALYSIS

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

Theorem 5. *Given a probabilistic graph $\mathcal{G}$, proHIT computes the truss decomposition of $\mathcal{G}$ in $O(tk_{\max,\eta}\psi m)$, where t is the total number of iterations $k_{\max,\eta} = \max_e\{\eta\text{-sup}_{\mathcal{G}}(e)\}$, ψ is the minimum number of spanning forests needed to cover all edges of $\mathcal{G}$, and m is the number of the edges.*

Proof. The time complexity of Algorithm 2 is dominated by the time complexity of *Phase II*, since h-index computation of edges is done by dynamic programming (DP) algorithm which has quadratic time complexity. In contrast, the h-index computation in *Phase I* can be done in linear time.

To analyze the time complexity of *Phase II* (given in Algorithm 3), we should note that for each edge $e = (u, v)$, the first time computation of the probability $\Pr[\sup(e) \geq H \mid \Gamma]$ in line 6 (Algorithm 3), takes $O(Hj_0)$ time, where $j_0 = |\Gamma|$, $H = h(e)$, and Γ is as given in the algorithm. For the next iterations in the while loop (line 6, Algorithm 3), using *Modified DP*, the computation is performed on $\mathcal{T}^i$ only, where $i = H-1, \cdots, 0$, and $\mathcal{T}^i$ is as before. In the worst case, the while loop is repeated H times. Let us assume that $j_1 = \left|\mathcal{T}^{H-1}\right|$, $j_2 = \left|\mathcal{T}^{H-2}\right|, \cdots, j_{k_e} = \left|\mathcal{T}^0\right|$. It is obvious that $j_0 + j_1 + \cdots + j_{k_e} = k_e$, where k_e is the number of common neighbors of u and v. We have that $k_e \subseteq O(\min\{d(u), d(v)\})$, where $d(u)$ and $d(v)$ are the degree of vertices u and v, respectively. Therefore, the while loop takes $O(j_0H) + O(j_1(H-1)) + \cdots + O(j_{k_e-1}1)$ time. In the worst case then, the time complexity of the while loop is bounded by $O(j_0k_{\max,\eta}) + O(j_1k_{\max,\eta}) + \cdots + O(j_{k_e-1}k_{\max,\eta})$, which is equal to $O(k_{\max,\eta}k_e) \subseteq O(k_{\max,\eta}\min\{d(u), d(v)\})$.

Moreover, the iteration over each neighbor of edge e in line 12 (Algorithm 3), takes $O(\min\{d(u), d(v)\})$. As a resul, the time complexity of *Phase II* is bounded by

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

Thus, the time complexity becomes:

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

It should be noted that $\psi \leq \min\{d_{\max}, \sqrt{m}\}$, where $d_{\max}$ is the maximum degree in the graph. Let t be the total number of iterations. The total time complexity is $O(tk_{\max,\eta}\psi m)$. In the worst case the number of iterations, t, is bounded by the degree levels as discussed in Theorem 4 and Corollary 1 in Section 5. The number of degree levels are bounded by $\beta = k_{\max,\eta}$.

□

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

For example, let us consider the flickr dataset, with $\beta = k_{\max,\eta} = 49$. However, as can be seen in Figure 6, for flickr with $\eta = 0.1$, the

total number of iterations is about 18 which is much smaller than β. This trend is also evident for other datasets.

Graph	$\|V\|$	$\|E\|$	$\|\triangle\|$	Reference
flickr	24,125	300,836	8,857,038	[3]
dblp	684,911	2,284,991	4,582,169	[3]
biomine	1,008,201	6,722,503	93,716,868	[3]
uk-2014-tpd	1,766,010	15,283,718	259,040,749	[1, 2]
itwiki-2013	1,016,867	23,429,644	89,901,299	[1, 2]
in-2004	1,382,908	27,182,946	464,257,245	[1, 2]
ljournal-2008	5,363,260	49,514,271	411,155,444	[1, 2]
enwiki-2013	4,206,785	91,939,728	304,083,160	[1, 2]

Table 3: Dataset Statistics

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

Table 4: $k_{\max,\eta}$, $\max_e\{\kappa_\eta(e)\}$, η.

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

Table 5: The values of $\text{avg}_\eta\{k_{\max,\eta}\}$, and $avg_\eta\{\max_e\{\kappa_\eta(e)\}\}$ over $\eta = 0.1, \cdots, 0.5$.

7 EXPERIMENTS

In this section, we present our experimental results. Our implementations are in Java and the experiments are conducted on a machine with Intel i7, 2.2Ghz CPU, and 12Gb RAM, running Ubuntu 18.04. The statistics for the datasets are shown in Table 3. We report the number of vertices $|V|$, the number of edges $|E|$, and the number of triangles $|\triangle|$. Datasets with real probability values are *flickr*, *dblp*, and *biomine*.

flickr is a popular online community for sharing photos. Nodes are users in the network, and the probability of an edge between two users is obtained based on the Jaccard coefficient of the interest groups of the two users [3, 20].

dblp comes from the well-known bibliography website. Nodes correspond to authors, and there is an edge between two authors if they co-authored at least one publication. The existence probability of each edge is measured based on an exponential function of the number of collaborations between two users [3, 20]. *biomine* contains biological interactions between proteins. The probability of an edge represents the confidence level that the interaction actually exists [3].

The rest of the datasets are social networks and web graphs which are obtained from Laboratory of Web Algorithms [1, 2]. For these datasets we generated probability values uniformly distributed in $(0, 1]$.

7.1 Efficiency Evaluation

In this section we report the running time of our proposed algorithm, *proHIT*, versus the state-of-the-art peeling algorithms, which we refer to as *PDT* (peeling-DP-truss) [11] and *PAPT* (peeling-approximate-truss) [8]. Both *PDT* and *PAPT* algorithms are based on iterative edge removal process which removes edges e with smallest probabilistic support, $\eta\text{-sup}_{\mathcal{G}}(e)$, and updating probabilistic support, $\eta\text{-sup}_{\mathcal{G}\setminus\{e\}}(e')$, of the affected edges e' in $\mathcal{G}\setminus\{e\}$. *PDT* uses dynamic programming for computing and updating probabilistic support of edges. However, *PAPT* uses statistical methods to approximate probabilistic support of edges, and as such, is an approximate algorithm. We use DP as an abbreviation for dynamic programming.

In our experiments, we set threshold $\eta = 0.1, \cdots, 0.5$. The running times in log-scale are shown in Figures 4 and 5. In Figure 4 we present the running times for *flickr*, *dblp*, *biomine*, and *ljournal-2008* using $\eta = 0.1$ as an example. In Figure 5, we separate the running times for the rest of the datasets due to different scales in their plot of running times. For these datasets we show the results for $\eta = 0.2, \cdots, 0.5$, since *PDT* cannot complete in reasonable time for $\eta = 0.1$. Moreover, for each dataset, we obtain the average of the maximum probabilistic support, $\text{avg}_\eta\{k_{\max,\eta}\}$, and the average of maximum truss value, $\text{avg}_\eta\{\max_e\{\kappa_\eta(e)\}\}$, over $\eta = 0.1, \cdots, 0.5$. These statistics are shown in Table 5, second and third columns, respectively.

As can bee seen in Figures 4 and 5, our *proHIT* algorithm is significantly faster than *PDT*, especially on networks containing a large number of triangles, and having large value of $\text{avg}_\eta\{k_{\max,\eta}\}$. For instance, for *biomine* (Figure 4) which is such a dataset, the gain of our algorithm compared to *PDT* is 84%, making *proHIT* six times faster than *PDT*. In fact, for biomine, for η equal to 0.1 and 0.2, *proHIT* is even better than the approximate algorithm *PAPT*, which, we recall, is an approximate algorithm. For the other η's for biomine, *proHIT* is slightly slower than *PAPT*. To reiterate, this is a welcome surprise because our proposed algorithm, *proHIT*, achieves a similar performance as *PAPT*, but without sacrificing the exactness of the solution.

In terms of running time on the smaller datasets, *flickr* and *dblp*, *proHIT* produces the results in 1.5 minutes and 1 minute, respectively. The number of triangles in *flickr* is twice larger than in *dblp* while having much less edges. We observe that *proHit* has a similar performance as *PAPT*. Both *proHit* and *PAPT* are faster than *PDT*, except on *dblp*. We recall that *dblp* is the smallest dataset in terms of probabilistic support and truss value of its edges, and as such it does not cause too much work for Dynamic Programming needed for *PDT*. As we see in the rest of the charts in Figure 5 *proHIT* significantly outperforms *PDT* and *PAPT* as the datasets get larger.

The running times of all algorithms increase for *ljournal-2008*, which is reasonable, because this graph has 49 million edges with

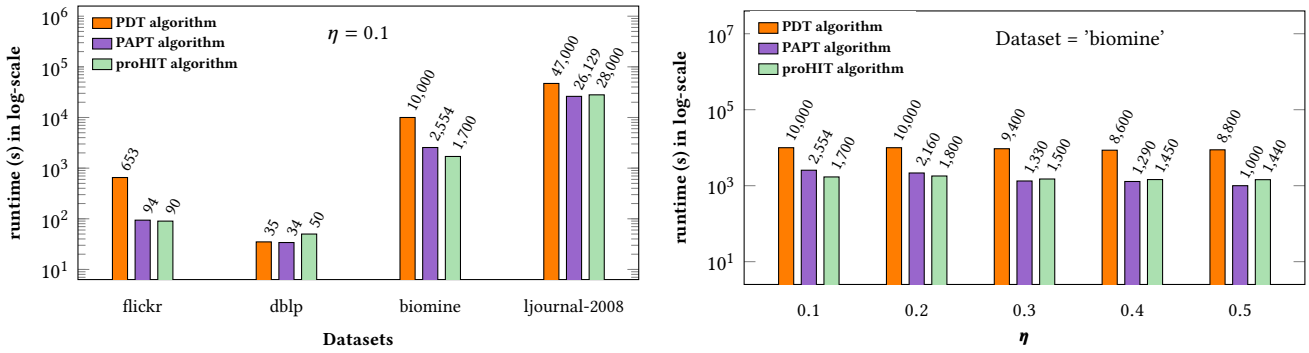

Figure 4: Running time of our proposed algorithm, proHIT, versus PDT and PAPT (baselines) for truss decomposition in probabilistic graphs.

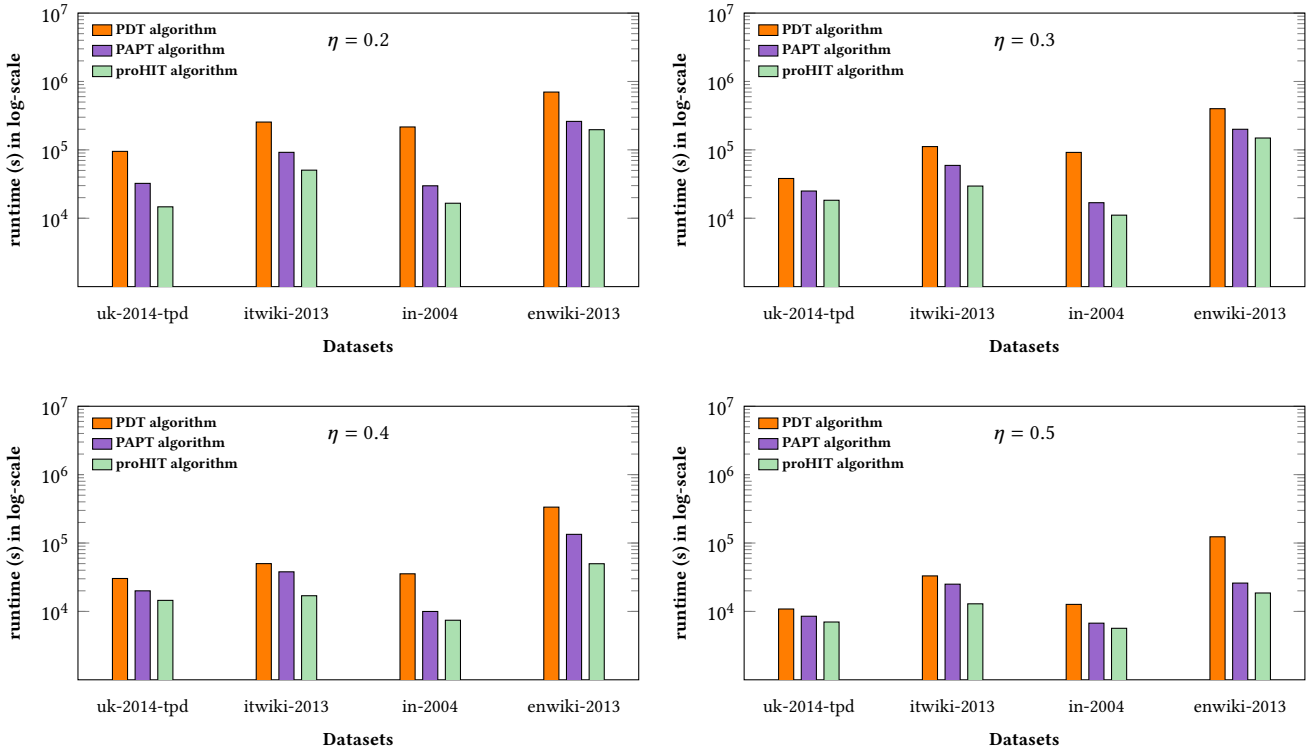

Figure 5: Running time of our proposed algorithm, proHIT, versus PDT and PAPT edge peeling (baselines) for truss decomposition on larger datasets with different values of η.

$\text{avg}_\eta\{k_{\max,\eta}\}$ equal to 911. For *ljournal-2008*, *proHIT* computes truss decomposition faster than *PDT* with a gain of 40 percent.

The running times continue to increase for the remaining datasets. This is because for these datasets $\text{avg}_\eta\{k_{\max,\eta}\}$ is much larger as shown in Table 5. For instance, for *itwiki-2013*, $\text{avg}_\eta\{k_{\max,\eta}\}$ is 4574. Moreover, for *uk-2014-tpd* and *in-2004* the ratio of the number of triangles to the number of edges is much higher than *ljournal-2008*.

As can be seen, for these graphs, *proHIT* is again significantly faster than its counterpart *PDT* (as an exact method). For instance, for *uk-2014* and *itwiki-2013* with $\eta = 0.3$, *proHIT* is about 2 and 3 times faster than *PDT*. Comparing *proHIT* with *PAPT* (which is an approximate method) shows that *proHIT* is on average 24% faster than *PAPT* without sacrificing the exactness of the solution. For *itwiki-2013* with $\eta = 0.5$, *proHIT* can complete truss decomposition in about 4 hours, while *PAPT* takes about 7 hours. Also, truss decomposition of *in-2004* using *proHIT* is 30 min faster than the one using *PAPT*. A similar trend can be observed for other values of η.

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

Note. It should be noted that as η increases, probabilistic support and truss values of edges decrease which lead to decrease in the running times of the algorithms. In Table 4, we show the trend for one of our dataset, *biomine*, in which $k_{\max,\eta}$ and $\max_e\{\kappa_\eta(e)\}$ decrease as η increases.

Next, we discuss why *proHIT* is faster than *PDT*. The most expensive part of both algorithms is executing DP routines, with quadratic run-times in the number of triangles containing each edge. However, their number and sizes are different in *proHIT* and *PDT*. Step 6 in *Phase II* (Algorithm 3) of *proHIT* uses DP to check the validity of the upper-bounds on the truss value of edges at each iteration of the algorithm. Also, at the beginning of *proHIT*, the upper-bound of each edge e is set to its η-$\sup_{\mathcal{G}}(e)$ which is obtained using DP (Algorithm 2, line 3). In *PDT*, DP is used after each edge removal, and all the edges that are neighbors of a peeled edge need to have their probabilistic support recomputed using DP.

Given a probabilistic graph $\mathcal{G}$, and edge $e = (u, v)$, let k_e be the number of common neighbors of u and v used for computing probabilistic support of e in $\mathcal{G}$. The time complexity of the computation by DP is $O(k_e^2)$ [11]. We refer to k_e as the *size* of DP. In *proHIT*, in Phase II, not all neighbors of u and v are used for DP but rather only those neighbors that can contribute to the final truss value of e (recall set Γ and Equation 3 in Section 4). As such, in *proHIT*, the size of DP is typically smaller than the total number of all the common neighbors of u and v. This is in contrast to *PDT*, which runs DP using all the remaining neighbors of an edge at that point in the peeling process. In essence, *proHIT* performs DP on smaller and only the effective set of neighbours for each edge, resulting in a considerable speedup.

We report the average and maximum sizes of DP for both algorithms in Table 6, for *flickr*, *dblp*, *biomine*, and *ljournal-2008*. As can be seen, for all the selected datasets, these sizes are much smaller for *proHIT* than for *PDT*. This is particularly important in large datasets, *biomine* and *ljournal-2008*, in which the average size for *proHIT* is about 3.5 and 4 times smaller than for *PDT*. In addition, in the last column of Table 6, we report the number of times DP is performed for both *PDT* and *proHIT* algorithms. The difference is noticeable for large datasets. For instance, on *ljournal-2008*, the number of executions of DP by *proHIT* is half of those performed by *PDT*.

Memory Usage. In Figure 7, we compare the memory consumption of *proHIT* versus that of *PDT* and *PAPT* on selected datasets. The trend can be verified for other datasets as well. As can be seen, for *biomine* the memory consumption of *proHIT* is 12 times and 6 times smaller than those of *PAPT* and *PDT*, respectively. This also holds for other datasets. For instance, for *ljournal-2008*, which is a large dataset, *PAPT* requires 90% more space than *proHIT*. Thus, Figure 7 confirms that *proHIT* consumes a smaller amount of memory for computing truss decomposition. This is because *PDT* and *PAPT* are edge peeling based algorithms which require maintaining the global information of the graph at each step of the algorithm, while *proHIT* uses local information only.

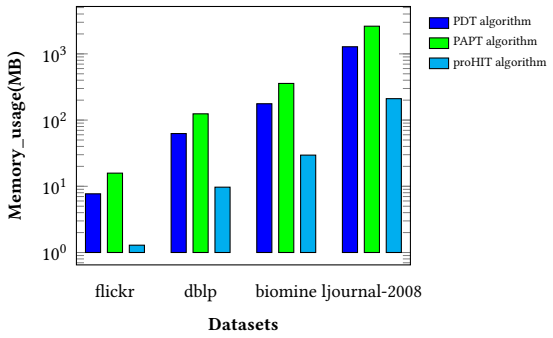

Figure 7: Memory usage of proposed algorithm versus the state-of-the-art edge peeling algorithms

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

Table 6: Average and maximum sizes of dynamic programming (DP), as well as the number of executions of DP for PDT and proHIT.

7.2 Convergence Speed

In this section we further evaluate the execution of *proHIT* as it unfolds with time. We look at the average distance from the truss values over the sequence of iterations for selected small to large datasets (see Figure 6). The average distance decreases fast for *flickr*, *dblp*, and *biomine*, and more gradually for ljournal-2008, in-2004, and uk-2014-tpd. These results show that *proHIT* can produce high-quality near-results in only a fraction of iterations needed for completion. For instance, for ljournal-2008 with $\eta = 0.1$, the average distance becomes less than 0.01 at iteration 20, about one third of the total number of required iterations (about 60, see the end of the curve). This can be a desirable property in graph mining where the user would like to see near-results as the execution progresses.

8 CONCLUSIONS

We presented a novel algorithm, *proHIT*, for computing truss decomposition in large probabilistic graphs. Our algorithm is based on an h-index updating approach. Unlike the edge peeling strategy, *proHIT* accesses the edges in a local fashion which makes it memory efficient. *proHIT* includes two main phases. *Phase I* is responsible for updating the edges' h-index without considering edge probabilities. This phase can provide a fast-to-compute upper-bound on truss values of the edges. *Phase II* takes care of the probabilistic nature of truss decomposition and further tightens the upper-bounds obtained in the previous phase. *proHIT* is an exact algorithm and is significantly faster than the state-of-the-art exact algorithm of [11]. While being an exact algorithm, *proHIT* can also produce

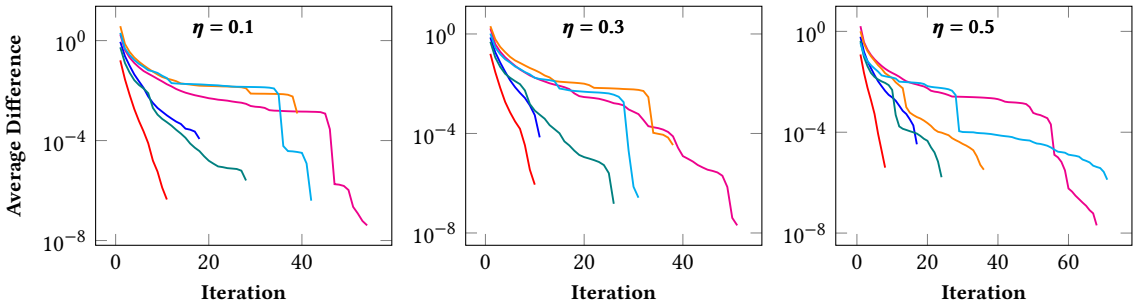

Figure 6: Average difference between the truss value and the upper bound over iterations for different values of η, for DBLP—, Flickr—, biomine—, ljournal-2008—, in-2004, uk-2014-tpd (best viewed in color).

near-results in only a fraction of iterations needed for computing the full exact solution.

REFERENCES

[1] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th International Conference on World Wide Web*. 587–596.

[2] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *Proceedings of the 13th International Conference on World Wide Web*. 595–602.

[3] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. 2014. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*. 1316–1325.

[4] Yulin Che, Zhuohang Lai, Shixuan Sun, Yue Wang, and Qiong Luo. 2020. Accelerating truss decomposition on heterogeneous processors. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1751–1764.

[5] Lei Chen and Xiang Lian. 2012. Query processing over uncertain databases. *Synthesis Lectures on Data Management* 4, 6 (2012), 1–101.

[6] Y. Cheng, Y. Yuan, L. Chen, and G. Wang. 2015. The reachability query over distributed uncertain graphs. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. 786–787.

[7] Fatemeh Esfahani, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. 2019. Efficient Computation of Probabilistic Core Decomposition at Web-Scale.. In *Proceedings of the 22nd International Conference on Extending Database Technology*. 325–336.

[8] F. Esfahani, J. Wu, V. Srinivasan, A. Thomo, and K. Wu. 2019. Fast Truss Decomposition in Large-scale Probabilistic Graphs.. In *Proceedings of the 22nd International Conference on Extending Database Technology*. 722–725.

[9] Nasrin Hassanlou, Maryam Shoaran, and Alex Thomo. 2013. Probabilistic graph summarization. In *International Conference on Web-Age Information Management*. Springer, 545–556.

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

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

[12] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. 2011. Distance-constraint reachability computation in uncertain graphs. *Proceedings of the VLDB Endowment* 4, 9 (2011), 551–562.

[13] Arijit Khan, Francesco Bonchi, Francesco Gullo, and Andreas Nufer. 2018. Conditional reliability in uncertain graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 11 (2018), 2078–2092.

[14] A. P. Mukherjee, P. Xu, and S. Tirthapura. 2015. Mining maximal cliques from an uncertain graph. In *2015 IEEE 31st International Conference on Data Engineering*. 243–254.

[15] Panos Parchas, Francesco Gullo, Dimitris Papadias, and Franceseco Bonchi. 2014. The pursuit of a good possible world: extracting representative instances of uncertain graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 967–978.

[16] Panos Parchas, Nikolaos Papailiou, Dimitris Papadias, and Francesco Bonchi. 2018. Uncertain graph sparsification. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2435–2449.

[17] You Peng, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Lu Qin. 2018. Efficient probabilistic k-core computation on uncertain graphs. In *2018 IEEE 34th International Conference on Data Engineering*. 1192–1203.

[18] Diana Popova, Akshay Khot, and Alex Thomo. 2018. Data Structures for Efficient Computation of Influence Maximization and Influence Estimation.. In *EDBT*. 505–508.

[19] Diana Popova, Naoto Ohsaka, Ken-ichi Kawarabayashi, and Alex Thomo. 2018. Nosingles: a space-efficient algorithm for influence maximization. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 1–12.

[20] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. 2010. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 997–1008.

[21] A. Sarıyüce, C. Seshadhri, and A. Pinar. 2019. Local Algorithms for Hierarchical Dense Subgraph Discovery. *VLDB* (2019).

[22] Shaden Smith, Xing Liu, Nesreen K Ahmed, Ancy Sarah Tom, Fabrizio Petrini, and George Karypis. 2017. Truss decomposition on shared-memory parallel systems. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–6.

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

[24] 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.

[25] Y. Xing, N. Xiao, Y. Lu, R. Li, S. Yu, and S. Gao. 2017. Fast Truss Decomposition in Memory. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. 719–729.

[26] Y. Yuan, L. Chen, and G. Wang. 2010. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In *International Conference on Database Systems for Advanced Applications*. 155–170.

[27] Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. 2013. Efficient keyword search on uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering* 25, 12 (2013), 2767–2779.

[28] Ye Yuan, Guoren Wang, Haixun Wang, and Lei Chen. 2011. Efficient subgraph search over large uncertain graphs. *Proceedings of the VLDB Endowment* 4, 11 (2011), 876–886.

[29] Y. Zhang and S. Parthasarathy. 2012. Extracting analyzing and visualizing triangle k-core motifs within networks. In *2012 IEEE International Conference on Data Engineering*. IEEE, 1049–1060.

[30] Feng Zhao and Anthony KH Tung. 2012. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment* 6, 2 (2012), 85–96.

[31] Zhaonian Zou, Hong Gao, and Jianzhong Li. 2010. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 633–642.

[32] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010. Finding top-k maximal cliques in an uncertain graph. In *2010 IEEE 26th International Conference on Data Engineering*. IEEE, 649–652.