

Sampling Subgraphs with Guaranteed Treewidth for Accurate and Efficient Graphical Inference

Jaemin Yoo
Seoul National University
Seoul, Republic of Korea
jaeminyoo@snu.ac.kr

U Kang*
Seoul National University
Seoul, Republic of Korea
ukang@snu.ac.kr

Mauro Scanagatta
Fondazione Bruno Kessler
Trento, Italy
mscanagatta@fbk.eu

Giorgio Corani
IDSIA
Lugano, Switzerland
giorgio@idsia.ch

Marco Zaffalon
IDSIA
Lugano, Switzerland
zaffalon@idsia.ch

ABSTRACT

How can we run graphical inference on large graphs efficiently and accurately? Many real-world networks are modeled as graphical models, and graphical inference is fundamental to understand the properties of those networks. In this work, we propose a novel approach for fast and accurate inference, which first samples a small subgraph and then runs inference over the subgraph instead of the given graph. This is done by the bounded treewidth (BTW) sampling, our novel algorithm that generates a subgraph with guaranteed bounded treewidth while retaining as many edges as possible. We first analyze the properties of BTW theoretically. Then, we evaluate our approach on node classification and compare it with the baseline which is to run loopy belief propagation (LBP) on the original graph. Our approach can be coupled with various inference algorithms: it shows higher accuracy up to 13.7% with the junction tree algorithm, and allows faster inference up to 23.8 times with LBP. We further compare BTW with previous graph sampling algorithms and show that it gives the best accuracy.

CCS CONCEPTS

• **Mathematics of computing** → Probabilistic inference problems; Markov networks; Probabilistic representations; • **Theory of computation** → Graph algorithms analysis.

KEYWORDS

Markov networks, graph sampling, treewidth, graphical inference

ACM Reference Format:

Jaemin Yoo, U Kang, Mauro Scanagatta, Giorgio Corani, and Marco Zaffalon. 2020. Sampling Subgraphs with Guaranteed Treewidth for Accurate and Efficient Graphical Inference. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20)*, February

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WSDM '20, February 3–7, 2020, Houston, TX, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-6822-3/20/02...\$15.00

<https://doi.org/10.1145/3336191.3371815>

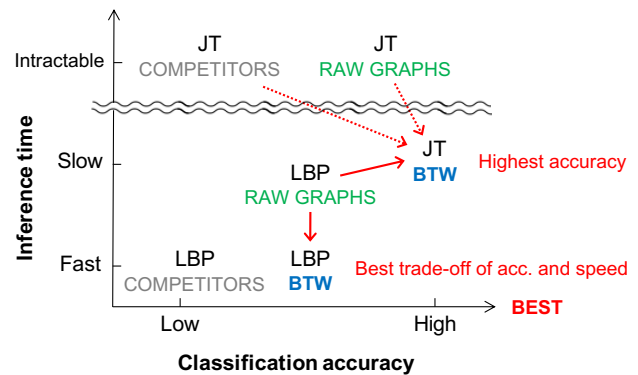


Figure 1: Advantages of BTW with two kinds of inference algorithms. BTW a) gives the best accuracy when the junction tree (JT) algorithm is used and b) speeds up the inference without hurting accuracy when LBP is used.

3–7, 2020, Houston, TX, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3336191.3371815>

1 INTRODUCTION

Given a large graph whose nodes represent discrete random variables, how can we compute their posterior marginals efficiently? Graphical inference is a crucial task in data mining and machine learning, which has been applied to solve various node classification problems such as malware detection [36], social network analysis [3, 12, 13, 15, 40], and recommender systems [5].

Loopy belief propagation (LBP) [39] is an inference algorithm which has been used widely for node classification. Yet, LBP has crucial limitations such that 1) it performs approximate inference rather than exact inference, and 2) its convergence is not guaranteed for general graphs; the conditions that LBP converges have been found only for restricted settings [11, 25]. These limitations make LBP difficult to be applied when the stability of inference is required. There are previous works [9, 10] that aim to solve the convergence problem by approximating LBP by a series of linear operations, but the amount of approximations is still not bounded.

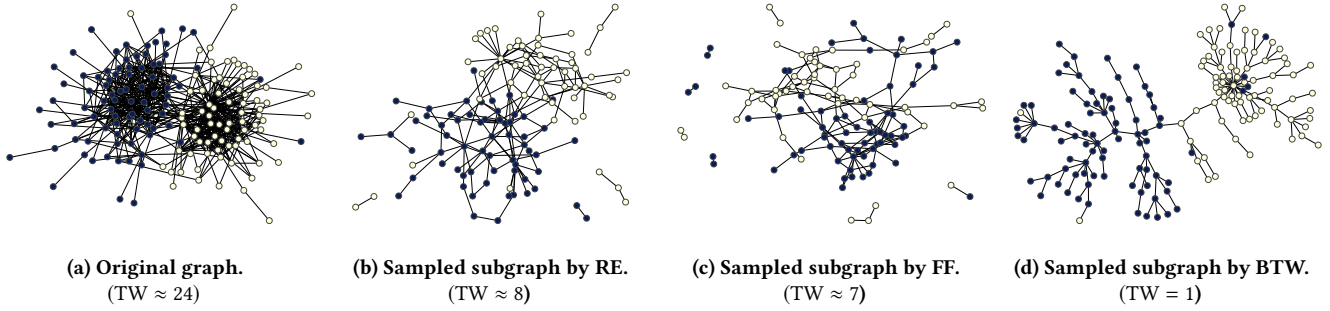


Figure 2: Guaranteed treewidth by BTW. (a) A web network of 163 blogs which are divided into two political groups. (b) The random edge sampling and (c) the forest fire sampling [19] generate subgraphs that have disconnected components and large treewidth (≥ 7). (d) Our proposed bounded treewidth (BTW) sampling generates a connected subgraph with the same number of edges as in (b) and (c), but with bounded treewidth of 1 while preserving the separate political groups.

The junction tree algorithm [17] is an exact inference algorithm that does not suffer from the unstable convergence. However, this approach requires exponential time and space with the treewidth of a graph [17]; thus, efficient inference requires graphs with *bounded treewidth*. Although it is NP-complete to compute the treewidth of a graph [4], recent advances [27, 28, 32] in structure learning of Bayesian networks are able to learn from data a high-scoring graph that respects a treewidth bound; this allows exact inference to be tractable. However, there are many applications where the graph is given, rather than learned. In such cases, exact inference is intractable if there is no control of treewidth.

In this work, we propose a novel approach which allows efficient inference on large graphs. Our idea comprises two steps. First, we sample the original graph obtaining a bounded-treewidth subgraph. This is done by our proposed bounded treewidth (BTW) sampling, a novel algorithm to generate a subgraph with bounded treewidth while retaining as many edges as possible. Once we obtain the subgraph, we run on it a) exact inference by the junction tree algorithm or b) approximate inference by LBP based on the purpose. If we run the junction tree algorithm, our approach is more accurate than running LBP on the original graph, despite its slow speed. We also have guaranteed convergence as the algorithm runs exact inference. If we run LBP, our approach shows similar accuracy with much faster computational time. These results are due to the success of our BTW sampling that maintains essential edges of the original graph with guaranteed bounded treewidth.

To the best of our knowledge, BTW is the first graph sampling algorithm that controls the treewidth of subgraphs and allows efficient graphical inference. The problem of generating subgraphs similar to an original graph has been studied widely in the data mining community [2, 19, 21], but none of the previous methods controls the treewidth of subgraphs. Furthermore, BTW considers cliques as minimal units of sampling, unlike other approaches that sample an edge at each iteration until the desired size is achieved. This gives every node a chance to preserve its comprehensive context which is essential for accurate graphical inference.

We evaluate our approach with the junction tree algorithm and LBP, which are the representative inference algorithms, on various real-world networks. As a result of extensive experiments, we demonstrate the following strengths of our approach:

- The junction tree algorithm on the subgraph generated from BTW achieves the highest accuracy, which is up to 13.7% higher than that from LBP on the original graph.
- LBP on the subgraph generated from BTW is up to 23.8× faster than LBP on the original graph, providing similar or even higher accuracy in some datasets.
- When we run LBP on subgraphs generated from BTW and other sampling algorithms, BTW shows up to 5.6% higher accuracy than the best competitors do.

These observations are summarized as Figure 1. BTW enables the junction tree algorithm which is intractable in raw graphs, leading to the highest accuracy. BTW also allows faster inference when LBP is used, while maintaining the original accuracy. The detailed experimental results are presented in Section 5.

Figure 2 shows that BTW generates a subgraph with guaranteed treewidth on a web network of 163 blogs with two political groups. The previous random edge (RE) sampling and forest fire (FF) sampling generate subgraphs of Figures 2b and 2c, respectively, which contain disconnected components, noisy connections between the different groups, and large treewidth (≥ 7). On the other hand, BTW generates a connected subgraph of Figure 2d, which has low treewidth of 1 and preserves the separate political groups with the same number of edges as in Figures 2b and 2c. As a result, the junction tree algorithm is tractable only in the graph of Figure 2d, and LBP shows the best performance in that graph. The approximate treewidth is reported in (a) to (c) by the *min-fill-in* and *min-degree* heuristics, since the exact computation is intractable [6].

2 RELATED WORKS

We introduce related works for our proposed approach, which are categorized into treewidth, inference algorithms, and sampling algorithms. We denote an undirected graph by $G = (\mathcal{V}, \mathcal{E})$ where \mathcal{V} and \mathcal{E} represent the sets of nodes and edges, respectively.

2.1 Treewidth

Intuitively, the treewidth of a graph quantifies the extent to which it resembles a tree structure. We introduce the concepts of a clique and triangulation before defining formally the treewidth.

Definition 2.1 (Clique). Given a graph G , a set C of nodes is a *clique* if all of its nodes are pairwise connected. A clique of k nodes is called a k -clique and it contains $k(k-1)/2$ edges. A clique C is *maximal* if it is not a subset of a larger clique in G .

Definition 2.2 (Triangulated graph). An undirected graph T is *triangulated* if all the chordless cycles existing in T are of length less than or equal to 3. Note that a chordless cycle is a cycle such that no two nodes of the cycle are connected by an edge that does not belong to the cycle. *Triangulation* is the process of generating a triangulated graph from a graph G by inserting additional edges cutting all the cycles of length greater than 3 in G .

Definition 2.3 (Treewidth). The *treewidth* of a triangulated graph T is the number of nodes in its largest maximal clique minus one. The treewidth of a general graph G is the minimum treewidth among all possible triangulations of G .

Given a graph, it is intractable to compute its treewidth because it requires all possible triangulations [6]. Instead, the treewidth can be bounded to k if the graph is shown as a subgraph of a k -tree.

Definition 2.4 (k -tree). An undirected graph $K = (\mathcal{V}, \mathcal{E})$ is a k -tree of treewidth k if the addition of any edge $(u, v) \notin \mathcal{E}$ among its nodes $u, v \in \mathcal{V}$ increases its treewidth k , or it is a $(k+1)$ -clique and thus no edge can be added. A subgraph of a k -tree is called a *partial k -tree*, and its treewidth is smaller than or equal to k .

We can generate a k -tree by the following inductive process [30]:

- **Base case:** a clique with $k+1$ nodes is a k -tree.
- **Inductive step:** given a k -tree K_n on n nodes, a new k -tree K_{n+1} is obtained by connecting a new node u to a k -clique C in K_n and generating k additional edges between u and C . This adds a new $(k+1)$ -clique of nodes $\{u\} \cup C$ to K_n .

Learning Markov networks that best represent given observations or probability distributions has been studied widely [16, 35]. On the other hand, score-based learning of Bayesian networks with bounded treewidth has been recently addressed [32, 33]. However, none of the previous approaches has used the idea of bounded treewidth learning to sample a subgraph that is suitable for accurate and efficient graphical inference.

2.2 Inference Algorithms

Belief propagation (BP) is an inference algorithm for Markov networks [39], which performs variable elimination efficiently. Given potentials of the variables in a graph, BP efficiently computes their posterior marginals by passing messages between the variables. BP computes the exact marginals when the graph is a tree. Otherwise, we run its variations described below.

Loopy belief propagation (LBP) is an approximate inference algorithm for cyclic graphs [39]. Unlike BP that passes the messages sequentially and only once following the tree structure, LBP updates all messages iteratively until convergence. However, since its convergence is not guaranteed, one needs to adjust the parameters

to stabilize the termination of the algorithm, which are represented as a potential or an affinity matrix [8, 29]. There exist various approaches to guarantee its convergence by linear approximations, but the amount of approximation is not bounded [9, 10].

The *junction tree algorithm* is an exact inference algorithm for cyclic graphs [17]. It triangulates a given graph G and generates a supergraph T . Then, it finds the maximal cliques on T and builds a maximum weight spanning tree over the cliques where the weight between two cliques is the number of shared variables. We get the exact marginals by running BP on the resulting tree. However, its time and space complexities are exponential with the size of the largest maximal clique in T , which depends on how we triangulate G . Since finding an optimal triangulation is intractable, a typical approach is to revert to a heuristic approach whose quality is not guaranteed [7]. As a result, a tractable computation of the junction tree algorithm is not guaranteed on a general graph.

2.3 Graph Sampling

Graph sampling is a task of generating subgraphs that preserve the properties of a graph. Given an undirected graph $G = (\mathcal{V}, \mathcal{E})$, we aim to generate a subgraph $U = (\mathcal{V}', \mathcal{E}')$ such that $\mathcal{V} = \mathcal{V}'$ and $\mathcal{E}' \subset \mathcal{E}$. In this work, we focus on the *edge-sampling* which samples only the edges not changing the nodes. This is because our objective is to run graphical inference efficiently. Each node is an essential variable which cannot be removed from the graph.

Graph sampling has been studied widely in data mining community. Leskovec and Faloutsos [19] compared different graph sampling algorithms and showed that the forest fire sampling [20] and the random walk sampling work well in maintaining the original properties. Li et al. [21] showed drawbacks of existing random walk based algorithms and proposed two novel methods. Wang et al. [38] compared various sampling algorithms on directed social networks. Voudigari et al. [37] proposed a sampling algorithm on social networks. Several sampling methods for triangle counting [14, 22, 23] have been proposed as well.

However, there has been no sampling algorithm that focuses on graphical inference. Most of the previous approaches sample subgraphs by selecting individual edges iteratively until the desired size is achieved, missing complex relationships between multiple nodes. On the other hand, we select cliques as minimal units to guarantee the bounded treewidth of subgraphs, maintaining the comprehensive neighborhood of each node. This leads to improved accuracy of graphical inference on the generated subgraphs.

3 PROPOSED METHOD

We propose the bounded treewidth (BTW) sampling, a novel algorithm for generating subgraphs with bounded treewidth. BTW selects cliques as minimal units instead of edges to bound the treewidth, supporting tractable and accurate inference by the junction tree algorithm and preserving the context of each node.

Figure 2a shows a simple graph of 163 nodes with binary states, which is generated by slicing randomly a principal submatrix from the adjacency matrix of the PolBlogs network which we use in our experiments (Section 4). The two groups of nodes are separated clearly satisfying the property of homophily. BTW generates the

Algorithm 1 Bounded treewidth (BTW) sampling

Require: a graph $G = (\mathcal{V}, \mathcal{E})$ and a bound k

Ensure: a sampled subgraph U and a k -tree K

```
1:  $K, U, H \leftarrow \text{initialize}(G, k)$ 
2: while  $|\mathcal{V}_U| < |\mathcal{V}|$  do
3:    $u, C \leftarrow \text{next\_node}(H)$ 
4:    $U \leftarrow (\mathcal{V}_U \cup \{u\}, \mathcal{E}_U \cup \{(u, v) | v \in N_G(u) \cap C\})$ 
5:    $K \leftarrow (\mathcal{V}_K \cup \{u\}, \mathcal{E}_K \cup \{(u, v) | v \in C\})$ 
6:    $H \leftarrow \text{update\_heap}(H, u, C)$ 
7: end while
8: return  $U, K$ 
```

subgraph in Figure 2d bounding its treewidth to 1, enabling accurate and efficient inference with only 39% of the edges.

3.1 Algorithm

The main objective of BTW is to guarantee the bounded treewidth of sampled subgraphs, which in turn leads to accurate and efficient inference. In other words, given a treewidth bound k , the treewidth of a subgraph U generated from BTW should be smaller than or equal to k . To achieve the objective, we keep track of a k -tree K along with U such that U is a subgraph of K . Thus, it is possible to bound its treewidth without exact computations.

BTW first selects $k + 1$ nodes and creates a $(k + 1)$ -clique, which is an initial k -tree K , and creates the induced subgraph U from the same set of nodes. Then, BTW takes iterations of selecting a new node u by a score-based approach: given a score function $m(v, C)$ between a candidate node v and a clique C in K , it selects the node with the maximum score. BTW adds the selected node u to both K and U until all nodes are included in U .

Algorithm 1 describes the whole process of BTW. It initializes a k -tree K , a subgraph U , and a max-heap H in line 1. The max-heap H is introduced to speed up the process and is discussed in Section 3.2. Then, BTW adds a new node u iteratively until all nodes are included in U , in lines 2 to 7. At each iteration, it selects u in line 3 and then updates U , K , and H in lines 4 to 6. BTW returns K along with U as a result in line 8, since K is a junction tree that is needed when running the junction tree algorithm over U .

3.1.1 Score Function. The simplest choice of the score function m is the uniform random function:

$$m(v, C) = \text{uniform}(0, 1). \quad (1)$$

Since it scores randomly all candidate nodes, using this function equals to first ordering all nodes randomly and then adding one node at a time following the order. However, it generates a sparse subgraph that maintains very few edges, because it does not take into account the structure of the original graph G .

Our main objective is to generate subgraphs that preserve the properties of G to support efficient graphical inference. Thus, we propose the following score function that is designed to maximize the number of edges in the sampled subgraphs:

$$m(v, C) = \sum_{x \in C \cap N_G(v)} (w(v, x) + \text{uniform}(0, \alpha)), \quad (2)$$

where $N_G(u)$ is the set of neighbors of u in G , α is a small constant, and $w(v, x)$ is a positive weight of the edge (v, x) .

This score function maximizes greedily the sum of edge weights in the subgraph U at every iteration. It simply maximizes the number of edges if w is a constant function, while one can design w to treat the edges differently. The uniform random function with the parameter α gives a randomness to the sampling process when the scores of multiple nodes are similar. For our experiments, we set w to a constant function as we deal with simple graphs, and α to a small value 10^{-4} to use it as a tie-breaker.

3.1.2 Subgraph Generation. BTW chooses randomly the first variable $u_1 \in \mathcal{V}$ and initializes the set \mathcal{I} of sampled nodes:

$$\mathcal{I} = \{u_1\}. \quad (3)$$

Then, until \mathcal{I} contains $k + 1$ variables we add to it a new node $u_i \in \mathcal{V} \setminus \mathcal{I}$ that maximizes the score as in Equation (4). Since we initially do not have a k -tree, we use \mathcal{I} instead of a clique: the score function m can be used generally with a node set.

$$u_i = \arg \max_{v \in \mathcal{V} \setminus \mathcal{I}} m(v, \mathcal{I}). \quad (4)$$

After that, BTW generates a complete graph K_{k+1} over the sampled $k + 1$ nodes, which is a k -tree by construction. BTW generates also the induced subgraph U_{k+1} by selecting all the original edges whose incident nodes are both in \mathcal{I} . U_{k+1} is the initial subgraph which we update in the following iterations, and its treewidth is at most k since it is a partial k -tree; it is a subgraph of K_{k+1} .

Then, BTW takes iterations of adding a subsequent node u_{i+1} to the current subgraph U_i and k -tree K_i of i nodes, generating new graphs U_{i+1} and K_{i+1} of $i + 1$ nodes. As in the initialization, BTW selects a node that maximizes the score function m as follows:

$$u_{i+1}, C_k^* = \arg \max_{v, C_k \in K_i} m(v, C_k), \quad (5)$$

where C_k is a k -clique contained in K_i .

Given u_{i+1} and C_k^* , BTW first updates K_i by connecting u_{i+1} to C_k^* . This is the same as adding a new clique $C_k^* \cup \{u_{i+1}\}$ to K_i by considering a clique as a minimal sampling unit. However, it is not possible to add the same clique to U_i since some of the edges in the clique may not be included in the original graph G . Thus, we instead connect u_{i+1} only to $N_G(u_{i+1}) \cap C_k^*$ to guarantee that U is a subgraph of G . Note that K is not necessarily a subgraph of G as it is used only for bounding the treewidth of U . We repeat the iteration until all nodes are sampled and return K and U .

3.1.3 Number of Sampled Edges. BTW aims to maximize greedily the number of edges in U while bounding its treewidth, using the score function of Equation (2). The function counts the number of edges that are added to U if u is connected to C . Thus, the number of edges in U_{i+1} is maximized as we find the maximum score for all pairs of nodes and cliques. The number of sampled edges will be much less if the score function of Equation (1) is used.

It is possible to further limit the number of edges in U by any sampling algorithm as U already has bounded treewidth; the treewidth of any of its subgraphs is bounded to k . On the other hand, it is very difficult to increase the number of sampled edges, since it is likely that more edges lead to increased treewidth. Recall that it takes exponential time even to simply compute the treewidth of a

graph. Thus, we leave it as a future work to increase the number of sampled edges from our subgraph U . It can be done from designing a new score function or by implementing a whole new algorithm coupled with more advanced techniques.

3.2 Optimization

It is redundant to compute the scores of all candidate nodes at each iteration since the scores of most nodes remain unchanged. Thus, we store in a max-heap the maximum score $s(v) = \max_{C \in K} m(v, C)$ of every candidate node v and optimize the algorithm by selecting the next node from the max-heap at each iteration.

We initialize $s(v)$ of every node v as zero. Then, at each iteration where a new clique C_{k+1} is added to the current k -tree K , we update the score $s(v)$ of every candidate node v as follows:

$$s(v) \leftarrow \max \left\{ s(v), \max_{C_k \subset C_{k+1}} m(v, C_k) \right\}. \quad (6)$$

If $s(v)$ is updated, we store also the clique C_k with the maximum score since it is needed when adding v to the subgraph.

A naive approach is to update the scores of all nodes neighboring in C_{k+1} , since such nodes' scores with regard to C_{k+1} are greater than zero. The new scores need to be compared with their previous maximum scores. To expedite the update process, our optimization updates the scores of *only the neighbors of a new node u* which has been added to the current subgraph U . This results in the same update as in the naive approach, but is more efficient in orders of magnitude: it makes BTW run in linear time with the number of edges and thus scalable to large graphs.

LEMMA 3.1 (OPTIMIZATION). *When a new node u and a clique C_{k+1} of $k + 1$ nodes are added to the current subgraph U and k -tree K , respectively, the score $m(v, C_k)$ of a node $v \notin N_G(u)$ and a clique $C_k \subset C_{k+1}$ is less than or equal to the current score $s(v)$.*

PROOF. We assume that α in Equation (2) is used only as a tie-breaker. The following holds because we assume $v \notin N_G(u)$:

$$\max_{C_k \subset C_{k+1}} m(v, C_k) = m(v, C_{k+1} \setminus \{u\}). \quad (7)$$

Then, Equation (6) for computing the new maximum score $s(v)$ of node v is given as follows with regard to C_{k+1} :

$$s(v) \leftarrow \max \{s(v), m(v, C_{k+1} \setminus \{u\})\}. \quad (8)$$

Recall that C_{k+1} is formed by connecting u to a k -clique $C_k = C_{k+1} \setminus \{u\}$, existing already in K . Since all cliques in K have been used already to update the score of every candidate node which is not included in the current K , C_k has also been considered for updating $s(v)$ before we add u to K in this step. Thus, Equation (8) is a redundant operation that does not change $s(v)$. \square

3.3 Theoretical Analysis

We further present theoretical analyses of BTW about its time complexity and preservation of connectivity. Lemma 3.2 shows that BTW has a linear scalability with the number of edges of the given graph. The effect of k^2 is negligible since we normally use a small treewidth bound k such as 2 or 4. Lemma 3.3 shows that BTW preserves the connectivity of the given graph in sampled subgraphs, which is especially useful for graphical inference, because disconnected nodes are considered independent from each other.

Table 1: A summary of four real-world networks that we use in our experiments. All datasets are publicly available.

Network	Nodes	Edges	Labels
Wikipedia ¹	35,579	495,357	16
CoRA ²	23,567	91,965	10
PubMed ³	19,717	44,324	3
PolBlogs ⁴	1,222	16,714	2

LEMMA 3.2 (TIME COMPLEXITY). *Given a graph $G = (\mathcal{V}, \mathcal{E})$ and a bound k , the time complexity of BTW is $O(|\mathcal{E}|(k^2 + \log |\mathcal{V}|))$ with the optimization of Lemma 3.1.*

PROOF. BTW selects a new node u at each iteration. The number of heap updates after selecting u is $O(|N_G(u)|)$, and each update is $O(\log |\mathcal{V}|)$. The computation of Equation (6) to update each score is $O(k^2)$. Thus, the complexity of all heap updates is given as

$$O\left(\sum_{u \in \mathcal{V}} |N_G(u)|(k^2 + \log |\mathcal{V}|)\right) = O(|\mathcal{E}|(k^2 + \log |\mathcal{V}|)).$$

The computational cost for heap deletions is safely ignored with a natural assumption of $|\mathcal{V}| \leq |\mathcal{E}|$, since it is $O(|\mathcal{V}| \log |\mathcal{V}|)$. \square

LEMMA 3.3 (CONNECTIVITY PRESERVATION). *When a given graph G is connected, all intermediate subgraphs during the iterations of BTW are connected, including the one to be returned.*

PROOF. We prove the lemma by mathematical induction:

- (1) **Base case.** The initial nodes in \mathcal{I} are guaranteed to be connected because every node is selected for maximizing the number of edges of the subgraph U when added to \mathcal{I} .
- (2) **Inductive step.** We assume that the subgraph U_i of i nodes is connected. Then, the next subgraph U_{i+1} with an additional node u_{i+1} becomes disconnected only if there exists no edge between U_i and the rest of G , since u_{i+1} has been selected for its maximum score. However, that cannot happen since G is connected. Thus, U_{i+1} is connected.

Thus, every subgraph generated by BTW is connected. \square

4 EXPERIMENTAL SETTINGS

In this section, we present experimental settings: datasets, competitors, etc. Our experiments were done by a workstation with Intel Xeon E5-2630 2.20GHz CPU and 50GB memory.

4.1 Datasets

We use four real-world networks summarized in Table 1. Wikipedia is a crawled dump of Wikipedia pages from 16 categories [26]. The nodes represent pages of certain categories, and the edges represent hyperlinks. CoRA and PubMed are citation networks whose nodes represent research articles, and edges represent citations [26, 34]. Each node is labeled according to the research topic. PolBlogs is a

¹<https://github.com/sharadnandanwar/snb>

²<https://github.com/sharadnandanwar/snb>

³<https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz>

⁴<http://www-personal.umich.edu/~mejn/netdata/polblogs.zip>

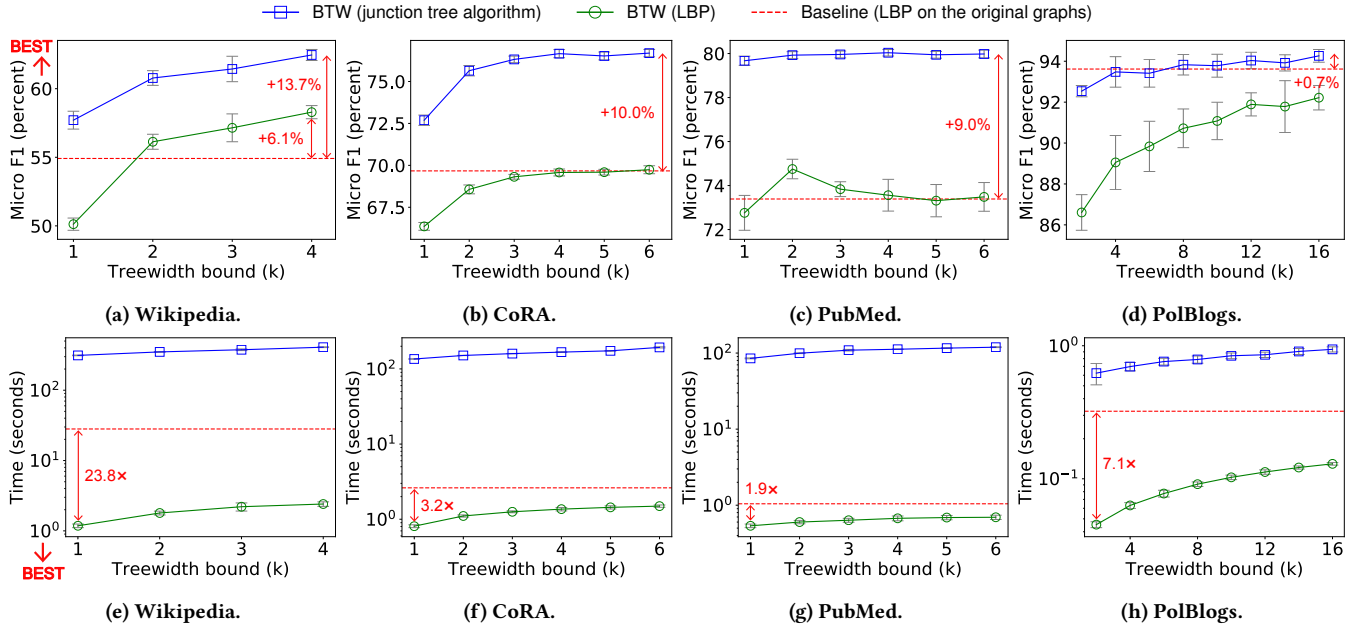


Figure 3: Micro F1 of node classification and inference time by the junction tree algorithm (the solid blue lines) and LBP (the solid green lines) on the subgraphs sampled by BTW, and the baseline (LBP on the original graph, shown by the red dashed lines). BTW with the junction tree algorithm achieves the highest accuracy, despite its slow inference time. On the other hand, BTW with LBP shows comparable accuracy to the baseline with much faster inference.

Table 2: The average numbers of edges in subgraphs generated from BTW, with respect to the treewidth bound k .

Network	$k = 1$	$k = 2$	$k = 4$	$k = 8$	$k = 16$
Wikipedia	35,578	57,896	84,213	126,234	165,620
Cora	23,566	36,307	46,158	51,310	54,479
PubMed	19,716	23,109	25,169	26,641	27,619
PolBlogs	1,221	2,004	3,086	4,477	6,374

webgraph whose nodes represent political blogs, which are liberal or conservative, and edges represent hyperlinks [1].

Some of the networks were originally directed. Thus, we model them as undirected by removing the edge directions as done in [24]. We also remove the isolated components since they are independent in terms of inference. The numbers of nodes and edges in Table 1 are counted after these preprocessing steps.

4.2 Graph Sampling Algorithms

We compare BTW with other graph sampling algorithms. We focus on edge sampling approaches that sample only edges, since every node is a target variable which we aim to classify.

Random edge (RE) sampling is the simplest algorithm that samples repeatedly an edge uniformly at random. However, it selects most edges from high-degree nodes, ignoring the importance of low-degree nodes. Random node-edge (RNE) sampling solves the problem by selecting a node uniformly at random and sampling

one of its adjacent edges. Hybrid (HYB) approach [18] combines the two approaches: it performs either a step of RE or RNE randomly with a probability p at each iteration. p is set to 0.8 as in [18].

Random walk (RW) sampling uniformly at random picks a node and then simulates a random walk with restart. It selects all edges that its random walker passes through. Random jump (RJ) sampling is similar to RW, but the random walker restarts randomly in any node instead of the starting one. Frontier sampling (FS) [31] uses multiple random walkers simultaneously to generate more stable subgraphs. Forest fire (FF) sampling [20] selects n neighbors instead of one at each step, where n follows a probability distribution.

Because none of these methods guarantees to keep the original connectivity in subgraphs, we propose two additional algorithms BTW-W and BTW-J by modifying RW and RJ, respectively. They guarantee the connectivity of a sampled subgraph by generating a spanning tree of nodes using BTW of $k = 1$ and then selecting the remaining edges by their original algorithms (RW and RJ). We use them to demonstrate that the superior performance of BTW comes from not only its ability to keep the connectivity, but also its idea of sampling cliques as minimal units instead of edges.

4.3 Node Classification by Inference

The real-world networks in Table 1 are not graphical models and thus contain no probabilistic information. For solving node classification by graphical inference, we model each dataset as a pairwise Markov random field (MRF), an undirected graphical model that has been widely used for node classification [8]. Compared with a standard MRF that requires a potential for every clique existing in

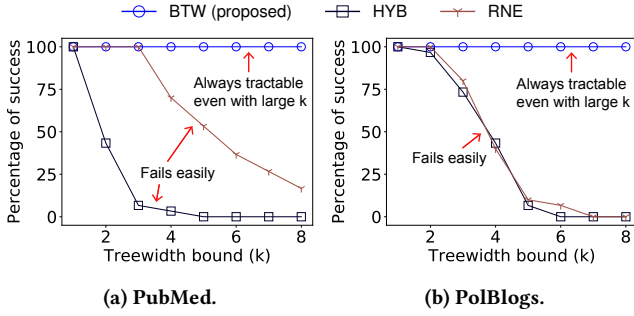


Figure 4: Ratios of successful inference (no out of memory) of the junction tree algorithm for the subgraphs from BTW and major competitors (HYB and RNE). BTW maintains the success ratio of 100% regardless of k , while the others easily fall into memory shortages as k increases.

the graph, a pairwise MRF uses only pairwise potentials, which we call *edge potentials*, and thus the modeling process is simple.

We generate an edge potential matrix ψ of size $n \times n$ based on the observed evidence of each dataset, where n is the number of states. First, we initialize ψ as a zero matrix. Then, for each connected pair of observed nodes in the graph, we add one to the corresponding element of ψ based on their labels. After that, we normalize ψ by the sum of its elements to make it represent a probability distribution. The resulting potentials are shown to follow the homophily (details in Appendix A). However, we have found that LBP does not converge with the computed potentials since they are too skewed. For this reason, we introduce a new parameter d that alleviates the skewness, and use it in all experiments where LBP is used (details in Appendix B). We use the original potentials when applying the junction tree algorithm since it has no convergence issue.

For each network and sampling algorithm, we generate 10 subgraphs with different random seeds. Then, we adopt the 3-fold cross validation for each subgraph: we hide the labels of 1/3 of the nodes and predict them by running graphical inference using the rest as evidence. We give the maximum-a-posteriori (MAP) assignments as predicted labels. We evaluate the result using micro and macro F1 scores as done in [26] but report only the micro F1 due to the lack of space. We report the averages in all experiments.

5 EXPERIMENTAL RESULTS

In this section, we present experimental results which answer the following questions about BTW:

- Q1. **Accuracy and inference time (Section 5.1).** What are the accuracy and inference time on the subgraphs from BTW compared with those on the original graphs?
- Q2. **Comparison with other sampling algorithms (Section 5.2).** Is BTW more suitable for the graphical inference compared with the other sampling algorithms?
- Q3. **Optimization of BTW (Section 5.3).** How much speedup can be obtained by the optimization of BTW? How does it scale with the number of edges and the treewidth k ?

Table 3: Micro F1 scores of LBP on the subgraphs from BTW and the competitors when the treewidth bound k is 2. BTW shows the best accuracy in all datasets.

Method	Wikipedia	CoRA	PubMed	PolBlogs
RE	35.3 \pm 0.2	57.9 \pm 0.3	61.8 \pm 0.4	75.8 \pm 1.0
RNE	51.3 \pm 0.3	65.2 \pm 0.2	71.1 \pm 0.1	84.4 \pm 0.6
HYB	49.4 \pm 0.2	64.5 \pm 0.2	69.8 \pm 0.3	83.4 \pm 0.4
RW	26.5 \pm 2.7	43.0 \pm 1.7	56.5 \pm 1.2	65.2 \pm 3.4
RJ	36.6 \pm 0.4	55.4 \pm 0.3	63.2 \pm 0.4	75.6 \pm 0.5
FS	29.8 \pm 0.2	47.9 \pm 0.2	56.2 \pm 0.5	72.4 \pm 0.8
FF	49.4 \pm 0.2	63.7 \pm 0.3	62.8 \pm 0.4	79.8 \pm 0.9
BTW-W	50.9 \pm 1.4	66.8 \pm 2.4	69.7 \pm 0.7	82.4 \pm 1.7
BTW-J	53.0 \pm 0.3	67.5 \pm 0.2	73.6 \pm 0.8	85.0 \pm 1.3
BTW	56.1 \pm 0.5	68.6 \pm 0.3	74.8 \pm 0.4	86.6 \pm 0.9

5.1 Accuracy and Inference Time

Figure 3 shows how the classification accuracy and inference time change if we run the inference algorithms on the subgraphs from BTW instead of the original graphs. The junction tree algorithm achieves up to 13.7% higher accuracy than the baseline (LBP on the original graph), showing consistent improvements in most cases; the exception is PolBlogs with small k , but the accuracy improves with k and outperforms the baseline from $k = 8$. Although the inference is slower than the baseline due to the overhead of computing the joint probability of each clique by the junction tree algorithm, the inference is still tractable because BTW guarantees to generate subgraphs with bounded treewidth.

The accuracy of LBP on the subgraphs is generally similar to the baseline. This implies that BTW successfully captures essential relationships between nodes, and thus LBP can recover the original accuracy with fewer edges: the subgraphs of Wikipedia when $k = 2$ contain only 11.7% of the original edges as shown in Table 2. At the same time, LBP on the subgraphs is consistently faster than the baseline from 1.5 \times to 23.8 \times based on the value of k .

We use small treewidth bounds in CoRA and Wikipedia (up to 6 and 4, respectively) in order to keep the junction tree algorithm tractable, which requires large memory of $O(n^k)$, where n represents the number of states [17]. Recall that n in both graphs are much larger than in the other graphs. However even in this case BTW yields an important improvement over the baseline.

5.2 Comparison with Other Approaches

We compare BTW and the other graph sampling algorithms by the classification accuracy on generated subgraphs. Most competitors require to know in advance the number of edges that the subgraph should contain, while BTW finds the subgraph automatically once k is specified. Thus, we first run BTW with a given bound k , record the number of edges that are present in the subgraph, and let the others generate subgraphs with the same numbers of edges.

Recall that the time and space complexities of the junction tree algorithm are both exponential with k . Usually, it is not possible to run exact inference on subgraphs generated without any control of treewidth, since it causes the junction tree algorithm to fail with

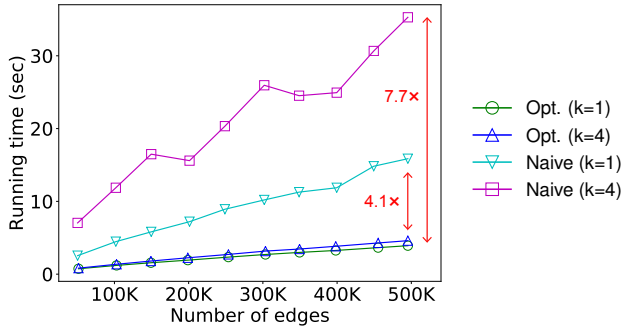


Figure 5: Sampling time of BTW on graphs of different numbers of edges and treewidth bound k . *Opt.* and *Naive* represent BTW with and without the optimization, respectively. BTW runs up to $7.7 \times$ faster with the optimization, showing near-linear scalability as presented in Lemma 3.2.

an out-of-memory error. The percentages of successful inference (no out-of-memory) on PubMed and PolBlogs are shown in Figure 4. They decrease sharply for all methods other than BTW, especially when $k > 3$, while the inference is always successful on the subgraphs returned by BTW due to the bounded treewidth.

We then perform node classification on such subgraphs reverting to LBP and obtain the accuracy in Table 3. BTW consistently shows the best accuracy compared with all competitors. It is notable that BTW performs better than BTW-J and BTW-W that keep the original connectivity in subgraphs by running BTW of $k = 1$ before RW and RJ. This implies that the superior performance of BTW does not rely on its property of keeping the connectivity, but on its main ideas of bounding the treewidth and providing each node a chance of having a neighborhood of size k in sampled subgraphs: every node keeps essential relationships with the others.

5.3 Optimization of BTW

Figure 5 shows the running time of BTW on graphs with different numbers of edges, with and without the optimization in Lemma 3.1. We have sampled nine principle submatrices from the adjacency matrix of Wikipedia for creating graphs of different sizes. BTW shows near-linear scalability with the number of edges as discussed in Lemma 3.2. It also shows the effect of optimization: introducing it makes BTW up to 7.7 times faster than before. Thus, BTW can be used efficiently for running inference in large graphs.

6 CONCLUSION

In this work, we propose the bounded treewidth (BTW) sampling, a novel graph sampling algorithm that generates subgraphs with guaranteed bounded treewidth. BTW samples as many edges as possible while bounding the treewidth, selecting cliques as minimal units of sampling and thus preserving the comprehensive neighborhood of each node. The subgraphs generated from BTW make the junction tree algorithm tractable by the bounded treewidth, which shows up to 13.7% higher micro F1 in node classification than loopy belief propagation (LBP) does on the original graph. Moreover, BTW speeds up LBP up to 23.8 times with accuracy comparable to that

on the original graph, and consistently higher than those from the other sampling algorithms. We expect that subgraphs generated from BTW are suited to other applications which can exploit the low treewidth. Future works include extending BTW to heterogeneous networks by introducing new score functions that consider various types of nodes, edges, and additional attributes.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) funded by MSIT (NRF-2015K1A3A1A14021055), and partially supported by the Swiss NSF grant IZKSZ2 162188.

REFERENCES

- [1] Lada A. Adamic and Natalie S. Glance. 2005. The political blogosphere and the 2004 U.S. election: divided they blog. In *LinkKDD*. 36–43.
- [2] Nesreen K. Ahmed, Jennifer Neville, and Ramana Rao Kompella. 2013. Network Sampling: From Static to Streaming Graphs. *TKDD* 8, 2 (2013), 7:1–7:56.
- [3] Leman Akoglu. 2014. Quantifying Political Polarity Based on Bipartite Opinion Networks. In *ICWSM*.
- [4] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. 1987. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic Discrete Methods* 8, 2 (1987), 277–284.
- [5] Erman Ayday and Faramarz Fekri. 2010. A belief propagation based recommender system for online services. In *RecSys*. 217–220.
- [6] Hans L. Bodlaender and Arie M. C. A. Koster. 2010. Treewidth computations I. Upper bounds. *Inf. Comput.* 208, 3 (2010), 259–275.
- [7] Andrés Cano and Serafin Moral. 1994. Heuristic Algorithms for the Triangulation of Graphs. In *IPMU*. 98–107.
- [8] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-Scale Graph Mining for Malware Detection. In *SDM*. 131–142.
- [9] Wolfgang Gatterbauer. 2017. The Linearization of Belief Propagation on Pairwise Markov Random Fields. In *AAAI*. 3747–3753.
- [10] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. 2015. Linearized and Single-Pass Belief Propagation. *PVLDB* 8, 5 (2015), 581–592.
- [11] Alexander T. Ihler, John W. Fisher III, and Alan S. Willsky. 2005. Loopy Belief Propagation: Convergence and Effects of Message Errors. *J. Mach. Learn. Res.* 6 (2005), 905–936.
- [12] Min-Hee Jang, Christos Faloutsos, Sang-Wook Kim, U. Kang, and Jiwoon Ha. 2016. PIN-TRUST: Fast Trust Propagation Exploiting Positive, Implicit, and Negative Information. In *CIKM*. 629–638.
- [13] Saehan Jo, Jaemin Yoo, and U. Kang. 2018. Fast and Scalable Distributed Loopy Belief Propagation on Real-World Graphs. In *WSDM*.
- [14] Minsoo Jung, Yongsu Lim, Sunmin Lee, and U. Kang. 2019. FURL: Fixed-memory and uncertainty reducing local triangle counting for multigraph streams. *Data Min. Knowl. Discov.* 33, 5 (2019), 1225–1253.
- [15] U. Kang, Duen Horng Chau, and Christos Faloutsos. 2011. Mining large graphs: Algorithms, inference, and discoveries. In *ICDE*. 243–254.
- [16] David Karger, David Karger, and Nathan Srebro. 2001. Learning Markov networks: Maximum bounded tree-width graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 392–401.
- [17] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.
- [18] Vaishnavi Krishnamurthy, Michalis Faloutsos, Marek Chrobak, Li Lao, Jun-Hong Cui, and Allon G. Percus. 2005. Reducing Large Internet Topologies for Faster Simulations. In *NETWORKING*. 328–341.
- [19] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *KDD*.
- [20] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*.
- [21] Rong-Hua Li, Jeffrey Xu Yu, Lu Qin, Rui Mao, and Tan Jin. 2015. On random walk based graph sampling. In *ICDE*. 927–938.
- [22] Yongsu Lim, Minsoo Jung, and U. Kang. 2018. Memory-Efficient and Accurate Sampling for Counting Local Triangles in Graph Streams: From Simple to Multigraphs. *ACM Trans. Knowl. Discov. Data* 12, 1, Article 4 (Jan. 2018), 28 pages.
- [23] Yongsu Lim and U. Kang. 2015. MASCOT: Memory-efficient and Accurate Sampling for Counting Local Triangles in Graph Streams. In *KDD*. 685–694.
- [24] Mary McGlohon, Stephen Bay, Markus G. Anderle, David M. Steier, and Christos Faloutsos. 2009. SNARE: a link analytic system for graph labeling and risk detection. In *KDD*. 1265–1274.
- [25] Joris M. Mooij and Hilbert J. Kappen. 2012. Sufficient conditions for convergence of Loopy Belief Propagation. *CoRR* abs/1207.1405 (2012). arXiv:1207.1405

- [26] Sharad Nandanwar and M. Narasimha Murty. 2016. Structural Neighborhood Based Classification of Nodes in a Network. In *KDD*. 1085–1094.
- [27] Siqi Nie, Cassio Polpo de Campos, and Qiang Ji. 2016. Learning Bayesian Networks with Bounded Tree-width via Guided Search. In *AAAI*. 3294–3300.
- [28] Siqi Nie, Denis Deratani Mauá, Cassio Polpo de Campos, and Qiang Ji. 2014. Advances in Learning Bayesian Networks of Bounded Treewidth. In *NIPS*.
- [29] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*. 201–210.
- [30] HP Patil. 1986. On the structure of k-trees. *Journal of Combinatorics, Information and System Sciences* 11, 2-4 (1986), 57–64.
- [31] Bruno F. Ribeiro and Donald F. Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *IMC*. 390–403.
- [32] Mauro Scanagatta, Giorgio Corani, Cassio Polpo de Campos, and Marco Zaffalon. 2016. Learning Treewidth-Bounded Bayesian Networks with Thousands of Variables. In *NIPS*. 1462–1470.
- [33] Mauro Scanagatta, Giorgio Corani, Marco Zaffalon, Jaemin Yoo, and U Kang. 2018. Efficient learning of bounded-treewidth Bayesian networks from complete and incomplete data sets. *Int. J. Approx. Reasoning* 95 (2018), 152–166.
- [34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (2008), 93–106.
- [35] Nathan Srebro. 2003. Maximum likelihood bounded tree-width Markov networks. *Artificial intelligence* 143, 1 (2003), 123–138.
- [36] Acar Tamersoy, Kevin A. Roundy, and Duen Horng Chau. 2014. Guilt by association: large scale malware detection by mining file-relation graphs. In *KDD*.
- [37] Elli Voudigari, Nikos Salamanos, Theodore Papageorgiou, and Emmanuel J. Yanakoudakis. 2016. Rank degree: An efficient algorithm for graph sampling. In *ASONAM*. 120–129.
- [38] Tianyi Wang, Yang Chen, Zengbin Zhang, Tianyi Xu, Long Jin, Pan Hui, Beixing Deng, and Xing Li. 2011. Understanding Graph Sampling Algorithms for Social Network Analysis. In *ICDCS*. 123–128.
- [39] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. 2003. Understanding Belief Propagation and Its Generalizations. In *Exploring Artificial Intelligence in the New Millennium*. 239–269.
- [40] Jaemin Yoo, Saehan Jo, and U. Kang. 2017. Supervised Belief Propagation: Scalable Supervised Inference on Attributed Networks. In *ICDM*. 595–604.

A POTENTIAL MATRICES

The following matrices ψ_B and ψ_M show the edge potentials used for PolBlogs and PubMed, respectively, that are generated from the process of modeling MRFs in Section 4.3:

$$\psi_B = \begin{bmatrix} 0.437 & 0.047 \\ 0.047 & 0.469 \end{bmatrix}, \quad \psi_M = \begin{bmatrix} 0.118 & 0.042 & 0.019 \\ 0.042 & 0.356 & 0.038 \\ 0.019 & 0.038 & 0.329 \end{bmatrix}.$$

Figure 6 shows the potentials of CoRA and Wikipedia by representing the values as colors (the larger, the darker). The diagonal elements in all these matrices are significantly larger than the others, implying that the nodes are positively correlated and follow the assumption of homophily in social networks.

B CONVERGENCE OF LBP

An exact condition that LBP converges has not been presented in the literature. Thus, we empirically show that LBP converges more often when the potential matrix is less skewed. We introduce a new parameter d which controls the skewness of a potential: a potential matrix ψ is replaced by a weighted average $\frac{1}{d}(\psi + (d-1)M)$ with a constant matrix M , whose elements are uniform and sum to one. Thus, the potentials become less skewed as d increases.

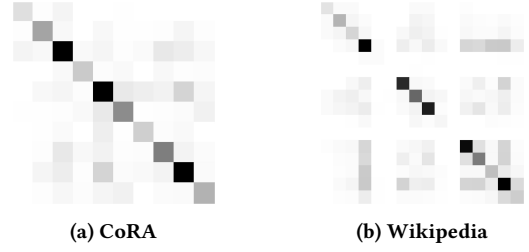


Figure 6: The potential matrices of CoRA and Wikipedia. The nodes in those graphs have the homophily as the diagonal elements are significantly larger than the others.

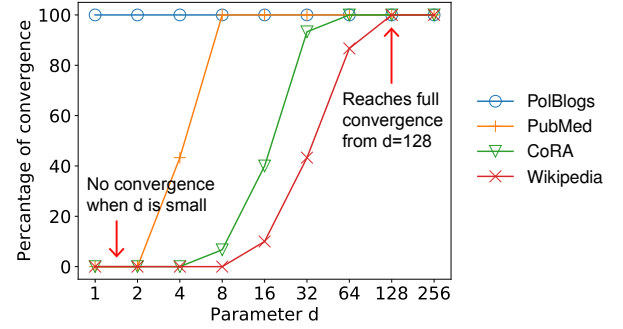


Figure 7: Percentages of runs that LBP converges by different values of d . LBP hardly converges with small d on three of the four datasets, and the convergence ratios increase as d increases: all ratios become 100% if $d \geq 128$.

For each dataset, we run LBP ten times with different sets of observed nodes and show in Figure 7 the percentage of runs that converge. We conclude it as a convergence when the maximum difference between the messages in two consecutive iterations is within 10^{-4} before 1000 iterations. As a result, it is shown that LBP hardly converges in three of the four datasets when d is a small value such as 2 or 4. The convergence seems especially difficult on CoRA and Wikipedia, which contain larger numbers of labels compared with the other datasets. The convergence ratios become 100% for all datasets when d is at least 128.

Based on this result, we set d to 256 when LBP is used, to guarantee a full convergence. On the contrary, it is not necessary to use d when the junction tree algorithm is used because the convergence is guaranteed: it stops after one iteration of fully propagating the observed information. This is an important advantage of running exact inference on subgraphs from BTW because it is not necessary to modify the relationships arbitrarily, which leads to different marginals from the ones that can be inferred from a dataset.