Fast and Scalable Distributed Loopy Belief Propagation on Real-World Graphs

Saehan Jo
Seoul National University
naheas@snu.ac.kr

Jaemin Yoo
Seoul National University
jaeminyoo@snu.ac.kr

U Kang
Seoul National University
ukang@snu.ac.kr

ABSTRACT

Given graphs with millions or billions of vertices and edges, how can we efficiently make inferences based on partial knowledge? Loopy Belief Propagation (LBP) is a graph inference algorithm widely used in various applications including social network analysis, malware detection, recommendation, and image restoration. The algorithm calculates approximate marginal probabilities of vertices in a graph within a linear running time proportional to the number of edges. However, when it comes to real-world graphs with millions or billions of vertices and edges, this cost overwhelms the computing power of a single machine. Moreover, this kind of large-scale graphs does not fit into the memory of a single machine. Although several distributed LBP methods have been proposed, previous works do not consider the properties of real-world graphs, especially the effect of power-law degree distribution on LBP. Therefore, our work focuses on developing a fast and scalable LBP for such large real-world graphs on distributed environment.

In this paper, we propose DLBP, a Distributed Loopy Belief Propagation algorithm which efficiently computes LBP in a distributed manner across multiple machines. By setting the correct convergence criterion and carefully scheduling the computations, DLBP provides up to 10.7× speed up compared to standard distributed LBP. We show that DLBP demonstrates near-linear scalability with respect to the number of machines as well as the number of edges.

ACM Reference Format:
Saehan Jo, Jaemin Yoo, and U Kang. 2018. Fast and Scalable Distributed Loopy Belief Propagation on Real-World Graphs. In *WSDM 2018: WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining , February 5–9, 2018, Marina Del Rey, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3159652.3159722

1 INTRODUCTION

Given large graphs that do not fit in a single machine, how can we make inference on unobserved vertices? Loopy Belief Propagation (LBP) is an inference algorithm which calculates approximate marginal probabilities of vertices based on partial information. It has been widely used in various fields including social network analysis [15, 16, 27, 31, 35], malware detection [5, 33], fraud detection [4, 28], recommendation [14, 38], and image restoration [8].

WSDM 2018, February 5–9, 2018, Marina Del Rey, CA, USA

ACM ISBN 978-1-4503-5581-0/18/02…$15.00
https://doi.org/10.1145/3159652.3159722

A nice property of LBP is that its computation complexity scales linearly with the number of edges. Since the computing of exact marginal probabilities in a Markov Random Field (MRF) is known to be computationally exorbitant, LBP has been one of the best alternatives. However, when it comes to real-world graphs whose sizes are very large, the application of LBP on these graphs is problematic in several aspects. First, the power-law degree distribution of real-world graphs disrupts a steady convergence of LBP algorithm. Second, LBP suffers from extensive computations per iteration due to the large size of graph data. Finally, the high communication overhead impedes the speed of distributed LBP algorithm since a large graph is distributed across multiple machines.

In this paper, we propose DLBP, a Distributed Loopy Belief Propagation, which solves the above challenges by 1) utilizing a convergence criterion appropriate for real-world graphs, and 2) dividing the vertices according to their degrees and carefully scheduling the iterations to minimize data communications. As shown in Figure 1, DLBP minimizes the communication cost and achieves up to 10.7× speed up on real-world graphs compared to standard LBP. Our contributions are as follows.

- **Algorithm.** We propose DLBP, a novel distributed algorithm for LBP which solves the challenges associated with the power-law degree distribution of real-world graphs.
- **Analysis.** We provide a theoretical analysis of two different convergence criteria of LBP. Moreover, we analyze DLBP in terms of the time complexity, space complexity, and the amount of shuffled data.
- **Experiment.** We provide evidence of the theoretical analysis by conducting rigorous experiments on DLBP to present empirical results on real-world graphs. DLBP shows up to 10.7× speed up on real-world graphs compared to baseline methods, as depicted in Figure 1.

The code of DLBP and the datasets are publicly available at http://datalab.snu.ac.kr/dlbp. The rest of the paper is organized as follows. In Section 2, we introduce the preliminaries and related works on LBP and distributed graph processing frameworks. Section 3 presents our proposed method DLBP in detail. After showing experimental results of DLBP on real-world graphs in Section 4, we conclude in Section 5.

2 PRELIMINARIES AND RELATED WORKS

In this section, we provide preliminaries and related works on belief propagation and graph processing. Table 1 describes the symbols used throughout the paper.

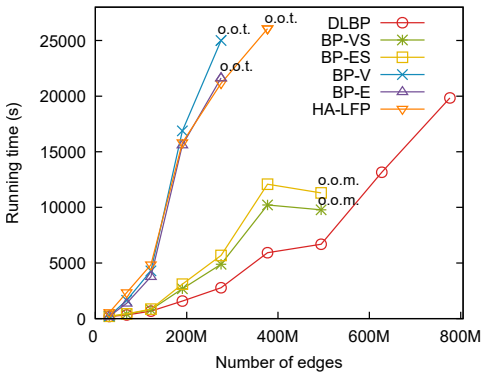


Figure 1: Running time of DLBP and competing methods on the subgraphs of YahooWeb (HA-LFP on Hadoop, rest on Spark). DLBP is up to 10.0× faster than HA-LFP, 10.7× faster than BP-E and BP-V, and 2.0× faster than BP-ES and BP-VS. o.o.t.: out of time (exceeds 8 hours). o.o.m.: out of memory (a worker node fails).

2.1 Loopy Belief Propagation

Loopy Belief Propagation (LBP) [20, 26] is an inference algorithm which approximately calculates the marginal distribution of unobserved variables in a probabilistic graphical model. We focus on LBP in a pairwise Markov Random Field (MRF) among other probabilistic graphical models to simplify the explanation. A pairwise MRF is an undirected graphical model consisting of vertices which represent random variables with a discrete number of states, and edges which represent the relationships between two variables.

To illustrate the equations of LBP, two concepts need to be introduced. A node potential $\phi_i(x_i)$ of a vertex i is the prior knowledge about the probability distribution of vertex i in the state x_i. An edge potential $\psi_{ij}(x_i, x_j)$ is the probability that vertex i has the state x_i and vertex j has the state x_j. LBP relies on the notion of iterative message passing between variables. At each iteration, all vertices send messages to their neighboring vertices. The algorithm assumes that a message m_{ij} successfully captures the influence of a vertex i on its neighboring vertex j's marginal probability. The message $m_{ij}(x_j)$ from vertex i to vertex j is updated as follows:

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j)\frac{\prod_{l\in N(i)} m_{li}(x_i)}{m_{ji}(x_i)} \tag{1}$$

where $N(i)$ is the neighbors of vertex i. The belief b_i of vertex i is computed based on the converged messages as follows:

$$b_i(x_i) = c_i\phi_i(x_i) \prod_{l\in N(i)} m_{li}(x_i) \tag{2}$$

where c_i is a normalization constant to make the beliefs of vertex i with different states x_i sum up to 1. Note that we can utilize the belief b_i of the current iteration to compute the outgoing messages m_{ij} from vertex i in the next iteration [17, 18]. Since Equation (1) uses the multiplication of the node potential $\phi_i(x_i)$ and the incoming messages $m_{li}(x_i)$ to vertex i in order to compute the next iteration messages, we can replace these terms with Equation (2) which results in:

$$m_{ij}(x_j) \leftarrow \frac{1}{c_i}\sum_{x_i} \psi_{ij}(x_i, x_j)\frac{b_i(x_i)}{m_{ji}(x_i)} \tag{3}$$

Table 1: Table of symbols.

Symbol	Definition
m_{ij}	message from vertex i to vertex j
b_i	belief of vertex i
c_i	normalization constant of vertex i
$N(i)$	set of neighboring vertices of vertex i
V	sets of vertices
E	sets of edges
V_h, V_s	sets of hub and spoke vertices
E_{hh}, E_{ss}, E_{hs}, E_{sh}	sets of edges from $\{V_h, V_s\}$ to $\{V_h, V_s\}$
B_h, B_s	sets of beliefs of V_h, V_s
M_{hh}, M_{ss}, M_{hs}, M_{sh}	sets of messages of E_{hh}, E_{ss}, E_{hs}, and E_{sh}
θ	convergence threshold
k	hub ratio
τ	maximum sub-iterations
T_h	total number of hub-to-hub iterations
T_s	total number of spoke-to-spoke iterations
T_o	total number of super-steps
T	number of iterations until standard LBP converges
n	number of machines
s	number of possible states

2.2 Distributed Graph Processing

In an attempt to solve problems in distributed graph processing, a variety of graph processing frameworks [6, 12, 19, 21, 24, 29, 30] have emerged. In those frameworks, graph data are distributed along multiple machines to promote parallel computing. However, the distribution of graph data is also a major bottleneck because of the communication cost between machines. Unfortunately, real-world graphs are known to contain little well-defined clusters [22].

Partitioning Strategies. Most of the distributed graph processing frameworks adapt either of the two graph partitioning strategies, edge-cut or vertex-cut. Edge-cut and vertex-cut mainly differ on which of the two main components of a graph, edges or vertices, are divided evenly among multiple machines. Edge-cut uniformly assigns vertices to multiple machines; hence, the edges are cut across different machines. In contrast, vertex-cut uniformly distributes edges to machines; thus, the vertices span over different machines. It is known that vertex-cut efficiently handles high-degree vertices in a real-world graph while edge-cut suffers imbalanced workloads among partitions [6, 12].

PowerLyra [6] utilizes both edge-cut and vertex-cut in parallel graph processing where one of the two partitioning methods is selected for a vertex depending on its degree. For high-degree vertices, vertex-cut is used to promote workload balancing between partitions. The low-degree vertices are partitioned using edge-cut which prevents the low-degree vertices from spanning over a large number of machines.

Parallel Belief Propagation. Different factors of parallel LBP are analyzed in [25] which still does not consider the property of real-world graphs. A parallel asynchronous LBP [10, 11, 34] uses a priority scheme in order to dynamically schedule the message updates. HA-LFP [18] formulates a MapReduce [7] version of LBP that runs in parallel on top of Apache Hadoop.

3 PROPOSED METHOD

In this section, we propose Distributed Loopy Belief Propagation (DLBP), a distributed LBP algorithm on real-world graphs. First, we give an overview of DLBP in Section 3.1 by describing the problems associated with LBP on real-world graphs. Next, we elaborate on

each of the main ideas of DLBP through Section 3.2 to Section 3.4. In Section 3.5, we analyze DLBP in terms of the amount of shuffled data, time complexity, and memory usage. Section 3.6 presents the implementation details of DLBP on the Spark platform.

3.1 DLBP: Overview

DLBP is a distributed LBP algorithm that efficiently computes the beliefs of vertices in real-world graphs. When considering an efficient parallel computation of LBP in a distributed environment, there are several challenges that need to be addressed.

(1) **Setting convergence criterion.** LBP iteratively updates messages until all messages converge. However, is this "message" convergence criterion appropriate for a real-world graph with high-degree vertices?
(2) **Minimizing numerical computations.** How can we minimize the overall computations of messages and the total number of iterations until convergence?
(3) **Minimizing network communication cost.** Real-world graphs are known to have highly-skewed degree distribution. With this property in mind, how can we partition a real-world graph and arrange the message computations to minimize the data communication between machines?

We tackle the problems mentioned above with the following main ideas, which are described in detail in later subsections.

(1) **Using "belief" as convergence criterion (Section 3.2).** We show that the message-based stopping criterion is insufficient to guarantee the convergence of further iterations, especially for real-world graphs with high-degree vertices.
(2) **Skipping of converged vertices (Section 3.3).** DLBP skips the computations of outgoing messages from converged vertices due to its belief convergence criterion. As a result, we minimize redundant message computations and promote faster convergence.
(3) **Reducing full shuffling of graph data (Section 3.4).** DLBP focuses on the high-degree vertices which require more number of iterations to converge than low-degree vertices. This idea reduces the amount of data shuffling.

3.2 DLBP: Belief Convergence Criterion

DLBP checks the convergence of beliefs instead of messages as the stopping criterion to overcome the shortcoming of message-based approach on real-world graphs. It is important to note that the beliefs, not messages, are the desired output of LBP; therefore, the reliability of resulting beliefs is the most crucial factor. In this context, we show that the convergence of messages in two consecutive iterations neither guarantees 1) the convergence of beliefs nor 2) the convergence of messages in the next iteration. Then, we present the belief convergence criterion and demonstrate its superiority over its counterpart.

When the "message" convergence criterion is used, LBP iteratively computes the messages until the following inequality holds for all messages m_{ij}^{t-1} of previous iteration $t-1$ and messages m_{ij}^{t} of current iteration t:

$$\|m_{ij}^{t} - m_{ij}^{t-1}\|_\infty \leq \theta \tag{4}$$

where $\theta \geq 0$ is a small constant, and $\|x\|_\infty$ denotes the largest absolute value of a vector x. However, high-degree vertices (which we refer to as hubs) in real-world graphs impede the stability of the message convergence criterion as we show in this subsection.

We use the approximate equations of *Linearized BP* proposed in [9] to give the condition where the message convergence criterion is problematic. Using the following definitions, Linearized BP utilizes Maclaurin series expansions to replace multiplications in Equations (1), (2), and (3) with summations.

Definition 1 (Centering [9]). *We call a vector x is centered around c if the average of all elements of x is exactly c, and each element deviates from c only by a small value.*

Definition 2 (Residual Value [9]). *When there is a vector x which is centered around c, the residual value $\hat{x}_i$ of the ith element x_i is defined as the deviation of x_i from c; that is, $\hat{x}_i = x_i - c$.*

The modified equations of Linearized BP are expressed using the residual values, which are Equations (5), (6) and (7). We denote the residual values of beliefs, messages, node potentials, and edge potentials by $\hat{b}_i(x_i)$, $\hat{m}_{ij}(x_j)$, $\hat{\phi}_i(x_i)$, and $\hat{\psi}_{ij}(x_i, x_j)$, respectively.

$$\hat{m}_{ij}(x_j) \leftarrow s \sum_{x_i} \hat{\psi}_{ij}(x_i, x_j)\hat{\phi}_i(x_i) + \sum_{x_i} \sum_{l \in N(i)\setminus j} \hat{\psi}_{ij}(x_i, x_j)\hat{m}_{li}(x_i) \tag{5}$$

$$\hat{b}_i(x_i) = \hat{\phi}_i(x_i) + \frac{1}{s} \sum_{l \in N(i)} \hat{m}_{li}(x_i) \tag{6}$$

$$\hat{m}_{ij}(x_j) \leftarrow s \sum_{x_i} \hat{\psi}_{ij}(x_i, x_j)\hat{b}_i(x_i) - \sum_{x_i} \hat{\psi}_{ij}(x_i, x_j)\hat{m}_{ji}(x_i) \tag{7}$$

where s is the number of possible states x_i, and $N(i) \setminus j$ is the set of vertices neighboring vertex i except vertex j.

Using these equations, we give an upper bound to the difference of belief $b_i(x_i)$ between two consecutive iterations under the condition that all incoming messages m_{li} to vertex i have converged. The result is in Lemma 1.

Lemma 1. *The convergence of incoming messages to a high-degree vertex i **does not** guarantee the convergence of its belief.*

Proof. We compute the difference of belief $b_i(x_i)$ between two consecutive iterations using Equation (6). Since all incoming messages $m_{li}(x_i)$ have converged, we use the inequality $|\triangle \hat{m}_{li}(x_i)| \leq \theta$ to give an upper bound of the value.

$$|\triangle b_i(x_i)| = \left| \frac{1}{s} \sum_{l \in N(i)} \triangle \hat{m}_{li}(x_i) \right| \leq \frac{1}{s} \sum_{l \in N(i)} \theta = \frac{1}{s}|N(i)|\theta$$

where $|N(i)|$ is the degree of vertex i. Since $|\triangle b_i(x_i)| \leq \frac{1}{s}|N(i)|\theta$, there is no guarantee $|\triangle b_i(x_i)|$ is within a small value when $|N(i)|$ is sufficiently large. □

Next, we solve the inequality in Equation (4) for an outgoing message $m_{ij}(x_j)$ from vertex i under the same condition as above to get Lemma 2.

Lemma 2. *The convergence of incoming messages **does not** guarantee the convergence of outgoing messages when a high-degree vertex i fails to satisfy the following inequality:*

$$|N(i)| \leq \frac{1}{s \cdot \max(\hat{\psi}_{ij}(x_i, x_j))} + 1$$

Proof. We compute the difference of the outgoing message $m_{ij}(x_j)$ between previous and current iterations by substituting the message terms with Equation (5). Similar to the proof in Lemma 1, we use the inequality $|\triangle \hat{m}_{li}(x_i)| \leq \theta$ to give an upper bound of the value. We use $\max(\hat{\psi}_{ij}(x_i, x_j))$ to denote the largest absolute value among the residuals of edge potentials.

$$\begin{aligned}|\triangle m_{ij}(x_j)| &= \left| \sum_{x_i} \sum_{l \in N(i) \setminus j} \hat{\psi}_{ij}(x_i, x_j) \cdot \triangle \hat{m}_{li}(x_i) \right| \\ &\leq \sum_{x_i} \sum_{l \in N(i) \setminus j} \max(\hat{\psi}_{ij}(x_i, x_j)) \cdot \theta \\ &= s(|N(i)| - 1) \cdot \max(\hat{\psi}_{ij}(x_i, x_j)) \cdot \theta\end{aligned}$$

Since $|\triangle m_{ij}(x_j)|$ is within the convergence threshold θ, we get the convergence criterion as:

$$|\triangle m_{ij}(x_j)| \leq s(|N(i)| - 1) \cdot \max(\hat{\psi}_{ij}(x_i, x_j)) \cdot \theta \leq \theta$$

By canceling out θ and rearranging the terms, we get the inequality in Lemma 2. □

Note that s is a constant value and $\max(\hat{\psi}_{ij}(x_i, x_j))$ depends on the edge potential values. Therefore, Lemma 2 indicates that if a vertex i has a degree $|N(i)|$ higher than some constant value, there is no guarantee that its outgoing messages will converge even after its incoming messages have converged.

A reasonable solution to this problem would be to introduce an alternative convergence criterion that mitigates the problem, which is to use beliefs instead of messages. When using "belief" convergence criterion, LBP stops when the following inequality holds for all beliefs b_i of two consecutive iterations:

$$\|b_i^t - b_i^{t-1}\|_\infty \leq \theta \tag{8}$$

This condition directly leads to the following lemma.

Lemma 3. *The belief convergence criterion guarantees the convergence of all beliefs.*

Next, we analyze the belief convergence criterion on the convergence of outgoing messages. By once more utilizing the Linearized BP equations, we get Lemma 4.

Lemma 4. *The convergence of the beliefs of adjacent vertices i and j restricts the difference of message between them in consecutive iterations within a small value:* $|\triangle m_{ij}(x_j)| \leq \frac{s^2 \cdot \max(\hat{\psi}_{ij}(x_i, x_j))}{1 - s \cdot \max(\hat{\psi}_{ij}(x_i, x_j))} \theta$

Proof. We compute the difference of the outgoing message $m_{ij}(x_j)$ by utilizing Equation (7). We give an upper bound by the inequality $|\triangle \hat{b}_i(x_i)| \leq \theta$.

$$\begin{aligned}|\triangle m_{ij}(x_j)| &= \left| s \sum_{x_i} \hat{\psi}_{ij}(x_i, x_j) \triangle \hat{b}_i(x_i) - \sum_{x_i} \hat{\psi}_{ij}(x_i, x_j) \triangle m_{ji}(x_i) \right| \\ &\leq s \sum_{x_i} \max(\hat{\psi}_{ij}(x_i, x_j)) \cdot \theta + \sum_{x_i} \max(\hat{\psi}_{ij}(x_i, x_j)) |\triangle m_{ji}(x_i)| \\ &= s \cdot \max(\hat{\psi}_{ij}(x_i, x_j)) \cdot (s\theta + |\triangle m_{ji}(x_i)|)\end{aligned}$$

Since the same conditions apply to $|\triangle m_{ji}(x_i)|$, we get a symmetric inequality $|\triangle m_{ji}(x_i)| \leq s \cdot \max(\hat{\psi}_{ij}(x_i, x_j)) \cdot (s\theta + |\triangle m_{ij}(x_j)|)$. By solving these two inequalities and rearranging the terms, we get the inequality in Lemma 4. □

We compare these two convergence criteria by introducing an example. Let's suppose there is a pairwise MRF whose vertices have two possible states, and we set edge potentials to 0.51 for the case where adjacent vertices have the same state and 0.49 for the opposite case. First, we examine the message convergence criterion by plugging the values $s = 2$ and $\max(\hat{\psi}_{ij}(x_i, x_j)) = 0.01$ to the inequality in Lemma 2:

$$|N(i)| \leq \frac{1}{2 \cdot 0.01} + 1 = 51$$

For high-degree vertices with degree $|N(i)| > 51$, the convergence of their incoming messages does not guarantee the convergence of their outgoing messages. On the other hand, when we use the belief convergence criterion, the following guarantee holds on the messages between converged vertices i and j regardless of their degrees: $|\triangle m_{ij}(x_j)| \leq \frac{2^2 \cdot 0.01}{1 - 2 \cdot 0.01} \theta \approx 0.0408 \cdot \theta$.

3.3 DLBP: Skipping of Converged Vertices

DLBP reuses outgoing messages m_{ij} of the previous iteration if the belief of their source vertex i satisfies the inequality in Equation (8). LBP-based algorithms using the message convergence criterion cannot safely skip the computation of outgoing messages of converged vertices because of Lemma 2. DLBP utilizes the belief convergence criterion which provides a more reliable standard for checking the convergence of a vertex. Thus, DLBP skips the computation of outgoing messages from vertices that have converged at the previous iteration with more reliability.

The obvious advantage of reusing the outgoing messages of converged vertices is that it omits the redundant computations. More importantly, skipping the computations also reduces the number of iterations required until all beliefs converge because it allows the algorithm to ignore very small errors of outgoing messages. The latter advantage contributes significantly in reducing the total running time of LBP algorithm. We easily see this by looking at the time complexity of LBP on graphs with cycles, which is given by Lemma 5.

Lemma 5 (Time Complexity of LBP [2]). *The time complexity of LBP on a graph $G = (V, E)$ with cycles is $O(|E|T)$, where V is the set of vertices, E is the set of edges, and T is the number of iterations required until all beliefs converge.*

3.4 DLBP: Hub-oriented Scheduling

We propose a hub-oriented scheduling, which DLBP uses to minimize the shuffled data when applying LBP in a distributed environment. The main idea is to focus on achieving the convergence of hubs and then propagate the messages to low-degree vertices (which we refer to as spokes). First, DLBP performs pre-processing to partition data based on hubs and spokes (Algorithm 1). Then the algorithm iterates carefully designed super-steps until all beliefs converge (Algorithm 2). Each super-step consists of four different sub-steps: hub-to-hub iteration, hub-to-spoke propagation, spoke-to-spoke iteration, and spoke-to-hub propagation. The objective of hub-oriented LBP is to minimize the full data shuffling by reducing the number of super-steps required until convergence.

3.4.1 Preprocessing Stage (Algorithm 1). In preprocessing stage, we introduce the hub ratio parameter $0 \leq k \leq 1$ to divide the

Algorithm 1: DLBP Preprocessing Stage

Input: vertex set V, edge set E, node potential set $\boldsymbol{\phi}$, and hub ratio k
Output: message sets M_{hh}, M_{sh}, M_{hs}, M_{ss},
belief sets B_h, B_s and node potential sets $\boldsymbol{\phi}_h$, $\boldsymbol{\phi}_s$

1: compute the degrees of all vertices in V
2: divide V into V_h, V_s according to vertices' degrees and k
3: divide $\boldsymbol{\phi}$ into $\boldsymbol{\phi}_h$, $\boldsymbol{\phi}_s$ with respect to V_h, V_s
4: initialize all beliefs B to $\frac{1}{s}$ where s is the number of states
5: divide B into B_h, B_s with respect to V_h, V_s
6: divide E into E_{hh}, E_{ss}, E_{hs}, E_{sh} according to their sources and destinations
7: initialize all messages M to 1
8: divide message set M into M_{hh}, M_{ss}, M_{hs}, M_{sh} with respect to E_{hh}, E_{ss}, E_{hs}, E_{sh}
9: **return** $\boldsymbol{\phi}_h$, $\boldsymbol{\phi}_s$, B_h, B_s, and M_{hh}, M_{sh}, M_{hs}, M_{ss}

vertices into hubs and spokes (lines 1 to 2). For example, if $k = 0.3$, then 30% of the vertices with the highest degrees are selected as hubs while the rest 70% become spokes. V_h and V_s represent the sets of hubs and spokes, respectively. Since each node potential ϕ_i is associated with a vertex i, the node potentials are also divided into two separate sets, $\boldsymbol{\phi}_h$ and $\boldsymbol{\phi}_s$, where boldfaced $\boldsymbol{\phi}$ denotes the set of all node potentials (line 3). In addition, we divide the beliefs into two sets, B_h and B_s (lines 4 to 5). Likewise, we divide the edges into four different sets according to their sources and destinations (line 6). E_{hh}, E_{ss}, E_{hs}, and E_{sh} denote the sets of edges whose source and destination are both hubs, both spokes, hub to spoke, and spoke to hub, respectively. M_{hh}, M_{ss}, M_{hs}, and M_{sh} are their affiliated message sets (lines 7 to 8). In the end, we have two subgraphs $G_h = (V_h, E_{hh})$ and $G_s = (V_s, E_{ss})$, and two intermediate sets of edges, E_{hs} and E_{sh}, that span over the two subgraphs (line 9).

We adapt the hybrid partitioning method from [6] and use differentiated partitioning strategies for the two subgraphs. The subgraph G_h of hubs is partitioned using vertex-cut to achieve balanced workload among machines while edge-cut is used for the subgraph G_s of spokes. For the vertex-cut, we use a 2d hash partitioner [3, 13] to limit the maximum number of machines over which a vertex spans. The intermediate edges, E_{hs} and E_{sh}, use edge-cut to group them by their destination vertices.

3.4.2 Main Stage (Algorithm 2). In main stage, DLBP carefully schedules the iterative message update steps in LBP according to the subgraphs, G_h and G_s, and intermediate edges, E_{hs} and E_{sh}. Since the subgraph of hubs is known to form a more dense graph than spokes [23], DLBP focuses on the subgraph G_h to promote the faster convergence of hubs and reduce the total number of super-steps until convergence. Each super-step of DLBP consists of two sub-iteration steps and two intermediate propagation steps: hub-to-hub iteration, hub-to-spoke propagation, spoke-to-spoke iteration, and spoke-to-hub propagation. DLBP iterates the super-steps until all beliefs satisfy the belief convergence criterion.

Hub-to-Hub Iteration (line 2 & Algorithm 3). In hub-to-hub iteration, we iteratively update M_{hh}, instead of all messages, until the belief convergence criterion holds for all beliefs in B_h. However, the naive approach to fully iterate the message updates until the convergence of B_H suffers from wasteful computations. Even though the hubs converged at current super-step, the information received

Algorithm 2: DLBP Main Stage

Input: vertex sets V_h, V_s, node potential sets $\boldsymbol{\phi}_h$, $\boldsymbol{\phi}_s$,
belief sets B_h, B_s, message sets M_{hh}, M_{sh}, M_{hs}, M_{ss},
edge potentials ψ_{ij}, and maximum inner iterations τ
// Note that c_i is a normalization constant
Output: beliefs B of all vertices

1: **while** $B_h \cup B_s$ not converged **do**
2: $B_h \leftarrow$ Sub-iteration(V_h, ψ_{ij}, τ, B_h, M_{hh}, $\boldsymbol{\phi}_h$, M_{sh})
3: $M_{hs} \leftarrow \frac{1}{c_i} \sum_{x_i} \psi_{ij}(x_i, x_j) \frac{b_i(x_i)}{m_{ji}(x_i)}$ using M_{sh} and B_h
4: $B_s \leftarrow$ Sub-iteration(V_s, ψ_{ij}, τ, B_s, M_{ss}, $\boldsymbol{\phi}_s$, M_{hs})
5: $M_{sh} \leftarrow \frac{1}{c_i} \sum_{x_i} \psi_{ij}(x_i, x_j) \frac{b_i(x_i)}{m_{ji}(x_i)}$ using M_{hs} and B_s
6: **end while**
7: **return** B

Algorithm 3: Sub-iteration($\cdot$) for Main Stage

Input: vertices V, edge potentials ψ_{ij}, maximum inner iterations τ,
$\forall v \in V$: beliefs B_v, messages M_{vv}, node potentials $\boldsymbol{\phi}_v$,
$\forall x \notin V$ and $\forall v \in V$: messages M_{xv}
Output: beliefs B_v

1: $p_i(x_i) \leftarrow \phi_i(x_i) \prod_{l \in N(i) \setminus V} m_{li}(x_i)$ for $\forall i \in V$ using $\boldsymbol{\phi}_v$
2: **while** (B_v not converged) & (less than τ iterations) **do**
3: $B_v \leftarrow c_i p_i(x_i) \prod_{l \in N(i) \cap V} m_{li}(x_i)$
4: $M_{vv} \leftarrow \frac{1}{c_i} \sum_{x_i} \psi_{ij}(x_i, x_j) \frac{b_i(x_i)}{m_{ji}(x_i)}$
5: **end while**
6: **return** B_v

from spokes breaks the convergence of hubs in the next super-step. Then we need to re-compute M_{hh} until they re-converge, where some of these computations might be inefficient in terms of approaching toward a global stationary point. To alleviate the problem, we introduce a parameter τ that restricts the maximum number of sub-iterations in hub-to-hub iteration. τ controls the trade-off between the information gained by the sub-iteration and the deviation of the converged values from the global optimum.

Since $\boldsymbol{\phi}_h$ and M_{sh} have constant values at each hub-to-hub iteration, we compute the multiplication of these values in advance to avoid redundant computations. More precisely, we compute the multiplication of constant node potential and incoming messages of vertex i as $p_i(x_i) = \phi_i(x_i) \prod_{l \in N(i) \setminus V_h} m_{li}(x_i)$. $N(i) \setminus V_h$ denotes the set of neighboring vertices of vertex i, which are not hubs. Then, the belief $b_i(x_i)$ of a hub i is computed as follows:

$$b_i(x_i) = c_i \phi_i(x_i) \prod_{l \in N(i)} m_{li}(x_i) = c_i p_i(x_i) \prod_{l \in N(i) \cap V_h} m_{li}(x_i)$$

This equation resembles Equation (2), except that the term $\phi_i(x_i)$ is replaced by $p_i(x_i)$. Using this equation, only M_{hh} and the set of $p_i(x_i)$ for $i \in V_h$ are required to compute B_h.

Hub-to-Spoke Propagation (line 3). After B_h converges, we compute M_{hs} which transfer the accumulated information from hubs to spokes. Note that only one step of full computations of M_{hs} is sufficient to fully propagate the accumulated information from hubs to spokes because B_h of the last hub-to-hub iteration is used to update M_{hs}.

Spoke-to-Spoke Iteration (line 4 & Algorithm 3). In spoke-to-spoke iteration, the only two differences from the hub-to-hub iteration are that 1) M_{ss} is updated instead of M_{hh}, and 2) B_s is used to check the convergence of beliefs. Similar to that of hub-to-hub iteration, we compute the set of $p_i(x_i) = \phi_i(x_i) \prod_{l \in N(i) \setminus V_s} m_{li}(x_i)$

for $i \in V_s$ before the iteration. It is noteworthy to mention that each spoke-to-spoke iteration converges within 1-3 steps in real-world graphs, which accords with our intuition that hubs are accountable for most of the iterations until an LBP algorithm converges.

Spoke-to-Hub Propagation (line 5). We compute M_{sh} to propagate the updated information of spokes to hubs.

3.5 Analysis of DLBP

We analyze DLBP with respect to the amount of shuffled data, time complexity, and memory usage. We use the following symbols to represent each factor: n (number of machines), T_o (total number of super-step iterations), T_h (total number of hub-to-hub iterations), T_s (total number of spoke-to-spoke iterations), T (number of iterations until standard LBP converges). Note that the equation $|E_{hs}| = |E_{sh}|$ always holds since the outgoing and incoming edges are symmetric.

3.5.1 Amount of Shuffled Data. We present the main result Lemma 9 which is supported by Lemmas 6, 7, and 8.

Lemma 6. *The amount of shuffled data in each loop of hub-to-hub iteration (lines 2 to 5 in Algorithm 3 called by line 2 in Algorithm 2) is $O(\sqrt{n}|V_h|)$.*

Proof. DLBP partitions the subgraph $G_h = (V_h, E_{hh})$ using vertex-cut; thus, a vertex spans across at most $2\sqrt{n}$ number of machines using a 2d hash partitioner [13]. First step is to aggregate the messages in M_{hh} by their destination vertices in order to compute B_h. Since the messages are initially aggregated within each machine before shuffling, the amount of shuffled data is $O(\sqrt{n}|V_h|)$. Next step is to update M_{hh}, which requires B_h to be distributed across at most $2\sqrt{n}$ machines. This also results in $O(\sqrt{n}|V_h|)$ shuffled data. Consequently, the amount of shuffled data for one hub-to-hub iteration is $O(\sqrt{n}|V_h|)$. □

Lemma 7. *The amount of shuffled data in each loop in spoke-to-spoke iteration (lines 2 to 5 in Algorithm 3 called by line 4 in Algorithm 2) is $O(|E_{ss}|)$.*

Proof. DLBP partitions the subgraph $G_s = (V_s, E_{ss})$ using edge-cut which assigns edges to machines according to their destinations. Thus, no shuffling is required when computing B_s and M_{ss}. However, the directions of newly updated messages in M_{ss} are inverted (source and destination are flipped) according to Equation (3). It is necessary to re-group the edges according to their new destinations, which requires the shuffling of $O(|E_{ss}|)$ data. □

Lemma 8. *The amount of shuffled data in each super-step except the loops in sub-iterations (line 1 in Algorithm 3, and lines 3 and 5 in Algorithm 2) is $O(|V| + |E_{hs}|)$*

Proof. When we consider hub-to-hub iterations, we compute the multiplication of $\boldsymbol{\phi}_h$ and M_{sh} in line 1 (Algorithm 3). Since the edges in E_{sh} are partitioned by edge-cut, no shuffling happens when multiplying messages in M_{sh}. However, we still need to multiply the multiplications of messages with $\boldsymbol{\phi}_h$, where the shuffled data is $O(|V_h|)$. Note that Algorithm 3 is reused for both hubs and spokes. In line 3 (Algorithm 2), there are two parts that we need to consider. First, we distribute the beliefs in B_h according to the destinations of M_{sh} to compute M_{hs}, which require $O(|V_h|)$ data to be shuffled. Then, we need to re-partition the updated M_{hs} according to their destination vertices which requires $O(|E_{hs}|)$ of shuffled data. The computation in line 3 (Algorithm 2) is symmetric with respect to hubs and spokes; line 5 is its counterpart. Therefore, the total amount of shuffled data in each super-step except the loops in sub-iterations is $O(|V| + |E_{hs}|)$. □

Lemma 9. *The total amount of shuffled data of DLBP until all beliefs B converge is as follows:*

$$O((|V| + |E_{hs}|)T_o + \sqrt{n}|V_h|T_h + |E_{ss}|T_s)$$

Proof. The equation is derived by a straightforward application of Lemmas 6, 7, and 8. □

Since $T_o \leq T_s < T_h \approx T$ holds in most cases (see Table 5), DLBP effectively reduces the amount of shuffled data compared to that of LBP. We easily show this by dividing the equation in Lemma 9 by the amount of shuffled data of standard LBP using vertex-cut with n machines, which is $O(\sqrt{n}|V|T)$ [13]. Under the condition that $T_o \leq T_s \ll T_h \approx T$, the shuffled data of DLBP is approximately k times smaller than that of LBP as follows, where k is the hub ratio:

$$\frac{(|V| + |E_{hs}|)T_o}{\sqrt{n}|V|T} + \frac{\sqrt{n}|V_h|T_h}{\sqrt{n}|V|T} + \frac{|E_{ss}|T_s}{\sqrt{n}|V|T} \approx \frac{\sqrt{n}|V_h|T_h}{\sqrt{n}|V|T} \approx \frac{|V_h|}{|V|} = k$$

The above simplification is based on the assumption that $|V| + |E_{hs}|$ and $|E_{ss}|$ are less than or equal to $\sqrt{n}|V|$, which is reasonable when n is sufficiently large.

3.5.2 Time Complexity.

Lemma 10. *The time complexity of DLBP is as follows:*

$$O\left(\frac{T_h|E_{hh}| + T_s|E_{ss}| + T_o|E_{hs}|}{n}\right)$$

Proof. Since hub-to-hub iterations are identical to applying LBP on a subgraph $G_h = (V_h, E_{hh})$ where $p_i(x_i)$ resembles the node potential $\phi_i(x_i)$, the time complexity is $O(T_h|E_{hh}|)$ on a single machine according to Lemma 5. When we use n machines to synchronously compute the messages, the time complexity becomes $O(\frac{T_h|E_{hh}|}{n})$. A similar justification applies for spoke-to-spoke iterations where the time complexity is $O(\frac{T_s|E_{ss}|}{n})$. The computations of messages in M_{hs} and M_{sh} simulate LBP algorithm on a bipartite graph $G_h = (V_h \cup V_s, E_{hs} \cup E_{sh})$; thus, the time complexity is $O(\frac{T_o|E_{hs}|}{n})$. By summing up these terms, we prove the lemma. □

When compared to the time complexity of standard LBP, $O(\frac{T|E|}{n})$, DLBP reduces the complexity under the condition that $T_o \leq T_s \ll T_h \approx T$. It is because $|E| = |E_{hh}| + |E_{ss}| + 2|E_{hs}|$; thus, $T_h|E_{hh}| + T_s|E_{ss}| + T_o|E_{hs}| < T|E|$ always holds true.

3.5.3 Memory Usage.

Lemma 11. *DLBP requires at most the memory space of:*

$$O(\max(|E_{hh}| + |V_h|, |E_{hs}| + |V_h|, |E_{sh}| + |V_s|, |E_{ss}| + |V_s|))$$

Proof. At each sub-step, we only need the memory space for the message set and the belief set associated with the sub-step. Thus, DLBP occupies up to the maximum memory space requirement among the sub-steps because unused data at current sub-step are stored in a secondary storage. □

The memory requirement of DLBP is always less than that of standard LBP since $|V| = |V_h| + |V_s|$ and $|E| = |E_{hh}| + |E_{ss}| + 2|E_{hs}|$.

Table 2: Summary of the datasets.

Dataset	Labeled	Nodes	Edges
YahooWeb-250M	13,463,596	174,577,088	776,375,840
YahooWeb-225M	11,649,829	151,082,221	627,242,209
YahooWeb-200M	9,806,416	128,663,268	494,471,965
YahooWeb-175M	8,146,152	106,656,291	376,985,487
YahooWeb-150M	6,456,257	85,309,472	275,133,522
YahooWeb-125M	4,906,727	65,829,290	190,485,192
YahooWeb-100M	3,517,351	47,141,314	121,566,975
YahooWeb-75M	2,319,233	30,731,733	68,492,600
YahooWeb-50M	1,264,080	16,402,838	30,403,395
Campaigns	12,059	23,191	877,729
PubMed	19,717	19,717	88,651
PolBlogs	991	1,224	16,716

3.6 Implementation

In this section, we present the implementation details of DLBP on Apache Spark [36, 37]. Note that DLBP can be implemented on any other distributed system which supports synchronous data processing. First, we store messages and node potentials as two separate *Resilient Distributed Datasets* (RDDs). Then, we divide RDDs according to the hub ratio k using *filter* operations. Each of the divided RDDs is partitioned by the *partitionBy* function with a customized partitioner which corresponds to either vertex-cut or edge-cut. Next, the *aggregateByKey* operation computes beliefs, and the *join* operation is used to check their convergence. Last, hub-to-hub message RDD partitioned by vertex-cut uses the *zipPartitions* function, which preserves the previous partitioning, to compute new messages while other message RDDs partitioned by edge-cut use the *join* function.

4 EXPERIMENTS

In this section, we evaluate the experimental results of DLBP. We answer the following questions:

- **Q1: Accuracy (Section 4.2).** Do the skipping technique and the hub-oriented scheduling of DLBP affect the accuracy of computed beliefs?
- **Q2: Speed (Section 4.3).** How quickly does DLBP compute the beliefs compared to baseline LBP algorithms?
- **Q3: Effect of parameters k and τ (Section 4.4).** How do the hub ratio k and the maximum sub-iterations τ affect the performance of DLBP in terms of speed?
- **Q4: Machine Scalability (Section 4.5).** How does DLBP scale up with respect to the number of machines?

4.1 Experimental Settings

Datasets. We use 4 datasets summarized in Table 2. Campaigns[1] is a graph of political committees and candidates with two possible states, where the edges represent donations. PolBlogs[2] [1] is a graph of hyperlinks where each vertex represents a web blog on US politics with two possible states. PubMed[3] is a citation graph where each vertex is a research paper with three possible categories. YahooWeb[4] is a web page hyperlink graph where labeled web pages

[1]http://www.cs.cmu.edu/~mmcgloho/data.html
[2]http://www-personal.umich.edu/~mejn/netdata/
[3]http://linqs.cs.umd.edu/projects/projects/lbc/index.html
[4]http://webscope.sandbox.yahoo.com/

Table 3: Edge potential values for homophilic graphs.

(a) Two states.

State	1	2
1	**0.501**	0.499
2	0.499	**0.501**

(b) Three states.

State	1	2	3
1	**0.334**	0.333	0.333
2	0.333	**0.334**	0.333
3	0.333	0.333	**0.334**

Table 4: Classification accuracy of the algorithms using 5-fold cross validation. There is no meaningful difference between the algorithms; however, DLBP is much faster than competitors as described in Section 4.3.

Dataset	BP-E	BP-V	BP-ES	BP-VS	DLBP
Campaigns	89.36%	89.36%	89.31%	89.31%	89.31%
PolBlogs	95.62%	95.62%	95.62%	95.62%	95.62%
PubMed	82.65%	82.65%	82.79%	82.79%	82.65%

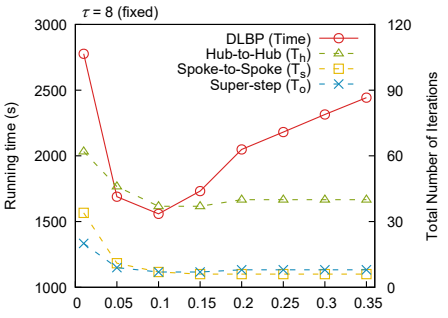
are either educational or adult pages. To make various real-world graphs with different sizes, we sample principal submatrices from the adjacency matrix of YahooWeb graph. For instance, YahooWeb-50M is the subgraph of YahooWeb where the dimensions of principal submatrix is 50M × 50M.

Note that these graphs are appropriate for LBP since they meet the homophily assumption. In particular, political groups are strongly connected to those with the same political opinion; research papers typically cite other papers within the same area; and educational web pages have links to other educational web pages, not adult web pages. The edge potential values are given by Table 3. The node potentials of labeled vertices are set to 0.9 for the correct label while other incorrect labels get a value that equally splits the remaining 0.1. Node potentials of unlabeled vertices have a uniform value $\frac{1}{s}$ for each state, where s is the number of possible states.

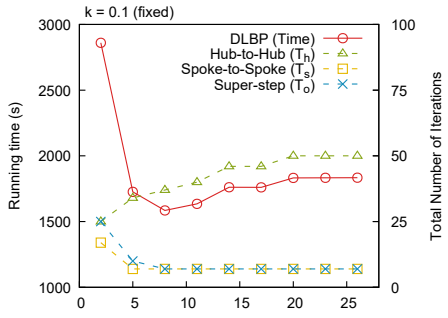
Methods. We evaluate our proposed DLBP compared to two baseline LBP algorithms based on edge-cut and vertex-cut (BP-E and BP-V). Moreover, we implement two additional LBP algorithms that improve BP-E and BP-V by utilizing the skipping technique introduced in Section 3.3 (BP-ES and BP-VS). The purpose of implementing BP-ES and BP-VS is to analyze the effects of the skipping technique on accuracy and speed. Lastly, DLBP uses both the skipping technique and the hub-oriented scheduling. All five algorithms are implemented on the Spark platform for a fair comparison, and use the belief convergence criterion with threshold $\theta = 10^{-4}$. The algorithms stop when the belief convergence criterion is met or total number of iterations exceeds 200.

We also evaluate DLBP compared to HA-LFP [18], a parallel LBP algorithm implemented on Apache Hadoop [7, 32]. Since HA-LFP requires users to manually choose the number of iterations, we give the number of iterations that BP-E and BP-V took to converge.

Environment Setting. We use Spark, a general data processing platform on distributed environment, to implement DLBP and four baseline algorithms mentioned above. We use the original code of HA-LFP provided by the authors of [18] and test on Apache Hadoop. The experiments were conducted on a cluster with 17 machines where each machine has a 32GB RAM and an Intel Xeon E3-1240 3.5GHz CPU with 4 cores. We configure Spark to run its application on the cluster using one machine as the driver node and the rest as the worker nodes where each node uses 24GB free memory.



(a) Hub Ratio k.



(b) Maximum Sub-iterations τ

Figure 2: Effects of the hub ratio k and the maximum sub-iterations τ on the performance of DLBP. We use YahooWeb-125M dataset to evaluate the different processing time according to the parameters. Note that the running time of DLBP is minimized when $k = 0.1$ and $\tau = 8$.

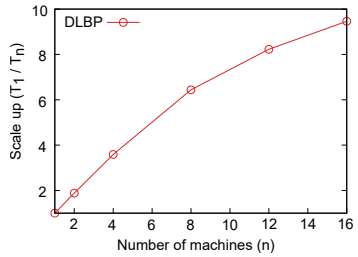


Figure 3: Running times of DLBP on YahooWeb-50M using different number of machines. DLBP demonstrates near-linear scaling with respect to the number of machines.

4.2 Accuracy (Q1)

We use vertex classification problem to measure the accuracy of the algorithms. The goal is to classify each vertex by assigning the label with the highest belief. We use 5-fold cross validation to ensure that we correctly estimate the accuracy of each algorithm. In addition, we balance the number of labeled vertices per label according to the label with the minimum number of labeled vertices.

We compare the classification accuracies of BP-ES, BP-VS, and DLBP to the baseline algorithms, BP-E and BP-V. Table 4 shows the classification results on each dataset. BP-ES, BP-VS, and DLBP show almost identical accuracies with the baseline algorithms on all the datasets. Note that the difference between the accuracies of two different methods is always less than 0.14%. Nevertheless, DLBP is much faster than competitors as we present in Section 4.3.

4.3 Speed (Q2)

We compare the running time of BP-ES, BP-VS, and DLBP with those of the baseline algorithms and HA-LFP. Since other datasets are too small to be a meaningful comparison of the speed of the algorithms, we use the subgraphs of YahooWeb to measure the running time. We measure the wall-clock time which includes the running time of the preprocessing stage of DLBP. Figure 1 presents the running time of each algorithm with regard to the number of edges. DLBP is up to 10.0× faster than HA-LFP, 10.7× faster than BP-E and BP-V, and even 2.0× faster than BP-ES and BP-VS. Table 5 demonstrates the number of sub-iterations and super-steps of DLBP until all beliefs converge. It shows that the number of super-steps of DLBP is significantly less than those of BP-ES and BP-VS.

4.4 Effects of Parameters in DLBP (Q3)

We analyze the effects of the hub ratio k and the maximum sub-iterations τ in terms of running time. Figures 2a and 2b present the running time of DLBP with varying parameters. While we evaluate the effect of one parameter, we fix the other parameters to their optimal values. Thus, $\tau = 8$ in Figure 2a and $k = 0.1$ in Figure 2b.

As shown in Figure 2a, DLBP takes more super-steps T_o until convergence when k is too small. It also causes the number of sub-iterations, T_h and T_s, to increase as well. On the other hand, large k does not affect the total number of iterations but rather increases the size of subgraph $G_h = (V_h, E_{hh})$, resulting in longer execution time. In Figure 2b, the number of super-steps until convergence increases when τ is very small, which results in longer running time. When $\tau = 1$, DLBP simulates the standard LBP where each iteration endures full shuffling of graph data. On the contrary, the number of hub-to-hub iterations increases as τ grows past the optimum value. As a result, the redundant computations in hub-to-hub iteration increase accordingly, an effect we mentioned in Section 3.4.

Table 5: Number of total sub-iterations and super-steps in DLBP on the subgraphs of YahooWeb. T denotes the number of iterations until BP-ES and BP-VS converge. The number of super-steps T_o in DLBP is much less than T on all datasets. Note that T_h has a value close to T, and T_s has a value close to T_o. k and τ are the values of hub ratio and maximum sub-iterations used in DLBP, respectively.

Dataset	k	τ	T_h	T_s	T_o	T
YahooWeb-50M	0.05	3	14	4	**4**	**12**
YahooWeb-75M	0.05	3	9	5	**5**	**13**
YahooWeb-100M	0.1	4	16	9	**7**	**16**
YahooWeb-125M	0.1	8	37	7	**7**	**39**
YahooWeb-150M	0.1	10	51	12	**8**	**50**
YahooWeb-175M	0.15	10	72	10	**10**	**77**
YahooWeb-200M	0.15	10	59	9	**9**	**54**
YahooWeb-225M	0.2	10	85	13	**13**	
YahooWeb-250M	0.25	10	86	12	**12**	

4.5 Machine Scalability (Q4)

We measure the machine scalability by measuring the running time of DLBP on YahooWeb-50M while increasing the number of machines from 1 to 16. Figure 3 illustrates the machine scalability in terms of speed up T_1/T_n where T_n is the running time using n machines. DLBP gains 9.46× speed up as the number of machines increases from 1 to 16.

5 CONCLUSION

We propose DLBP, a fast and scalable distributed loopy belief propagation method. DLBP enhances the standard LBP algorithm by overcoming the challenges associated with the skewed degree distribution of large real-world graphs. DLBP exploits a convergence criterion better suited for real-world graphs, and carefully schedules computations for better performance. Experimental results demonstrate that DLBP minimizes the communication cost of LBP and provides up to 10.7× speed up on real-world graphs without sacrificing the classification accuracy.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning(NRF-2015K1A3A1A14021055). U Kang is the corresponding author.

REFERENCES

[1] Lada A. Adamic and Natalie Glance. 2005. The Political Blogosphere and the 2004 U.S. Election: Divided They Blog. In *Proceedings of the 3rd International Workshop on Link Discovery (LinkKDD '05)*. 36–43.

[2] Ron Bekkerman, Mikhail Bilenko, and John Langford. 2011. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press.

[3] Ümit V. Çatalyürek, Cevdet Aykanat, and Bora Uçar. 2010. On Two-Dimensional Sparse Matrix Partitioning: Models, Methods, and a Recipe. *SIAM J. Scientific Computing* 32, 2 (2010), 656–683.

[4] Duen Horng Chau, Shashank Pandit, and Christos Faloutsos. 2006. Detecting Fraudulent Personalities in Networks of Online Auctioneers. In *Knowledge Discovery in Databases: PKDD 2006*. Springer Berlin Heidelberg, Berlin, Heidelberg, 103–114.

[5] Duen Horng Polo Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2013. Polonium: Tera-Scale Graph Mining and Inference for Malware Detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 131–142.

[6] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. 2015. PowerLyra: differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*. 1:1–1:15.

[7] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[8] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. 2006. Efficient Belief Propagation for Early Vision. *International Journal of Computer Vision* 70, 1 (2006), 41–54.

[9] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. 2015. Linearized and Single-Pass Belief Propagation. *PVLDB* 8, 5 (2015), 581–592.

[10] Joseph Gonzalez, Yucheng Low, and Carlos Guestrin. 2009. Residual Splash for Optimally Parallelizing Belief Propagation. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*. 177–184.

[11] Joseph Gonzalez, Yucheng Low, Carlos Guestrin, and David R. O'Hallaron. 2009. Distributed Parallel Inference on Large Factor Graphs. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*. 203–212.

[12] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*. 17–30.

[13] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. 599–613.

[14] Jiwoon Ha, Soon-Hyoung Kwon, Sang-Wook Kim, Christos Faloutsos, and Sunju Park. 2012. Top-N recommendation through belief propagation. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*. 2343–2346.

[15] 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 *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 629–638.

[16] Kyomin Jung, Wooram Heo, and Wei Chen. 2012. IRIE: Scalable and Robust Influence Maximization in Social Networks. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*. 918–923.

[17] U Kang, D.H. Chau, and C. Faloutsos. 2010. Inference of Beliefs on Billion-Scale Graphs. *The 2nd Workshop on Large-scale Data Mining: Theory and Applications* (2010).

[18] U. Kang, Duen Horng Chau, and Christos Faloutsos. 2011. Mining large graphs: Algorithms, inference, and discoveries. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*. 243–254.

[19] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. 2009. PEGASUS: A Peta-Scale Graph Mining System. In *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*. 229–238.

[20] Danai Koutra, Tai-You Ke, U. Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. 2011. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *ECML/PKDD (2)*. 245–260.

[21] Ho Lee, Bin Shao, and U Kang. 2015. Fast graph mining with HBase. *Information Sciences* 315, 0 (2015), 56 – 66.

[22] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2009. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[23] Yongsub Lim, U. Kang, and Christos Faloutsos. 2014. SlashBurn: Graph Compression and Mining beyond Caveman Communities. *IEEE Trans. Knowl. Data Eng.* 26, 12 (2014), 3077–3089.

[24] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*. 135–146.

[25] Alexander Mendiburu, Roberto Santana, José Antonio Lozano, and Endika Bengoetxea. 2007. A parallel framework for loopy belief propagation. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007, Companion Material*. 2843–2850.

[26] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. 1999. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999*. 467–475.

[27] Huy Nguyen and Rong Zheng. 2012. Influence Spread in Large-Scale Social Networks - A Belief Propagation Approach. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part II*. 515–530.

[28] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. *Netprobe: a fast and scalable system for fraud detection in online auction networks*. ACM, New York, New York, USA.

[29] Chiwan Park, Ha-Myung Park, Minji Yoon, and U. Kang. 2017. PMV: Pre-partitioned Generalized Matrix-Vector Multiplication for Scalable Graph Mining. *CoRR* abs/1709.09099 (2017).

[30] Ha-Myung Park, Chiwan Park, and U Kang. 2018. PegasusN: A Scalable and Versatile Graph Mining System. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, February 2-7, 2018, New Orleans, Lousiana, USA*.

[31] Adam Sadilek, Henry A. Kautz, and Jeffrey P. Bigham. 2012. Finding your friends and following them to where you are. In *Proceedings of the Fifth International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, WA, USA, February 8-12, 2012*. 723–732.

[32] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*. 1–10.

[33] Acar Tamersoy, Kevin A. Roundy, and Duen Horng Chau. 2014. Guilt by association: large scale malware detection by mining file-relation graphs. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. 1524–1533.

[34] Jiangtao Yin and Lixin Gao. 2014. Scalable Distributed Belief Propagation with Prioritized Block Updates. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*. 1209–1218.

[35] Jaemin Yoo, Saehan Jo, and U. Kang. 2017. Supervised Belief Propagation: Scalable Supervised Inference on Attributed Networks. In *IEEE 17th International Conference on Data Mining, ICDM 2017, November 18-21, 2017, New Orleans, USA*.

[36] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*. 15–28.

[37] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*.

[38] Jun Zou and Faramarz Fekri. 2013. A belief propagation approach for detecting shilling attacks in collaborative filtering. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*. 1837–1840.