Practical Message-passing Framework for Large-scale Combinatorial Optimization

Inho Cho* Soya Park* Sejun Park Dongsu Han Jinwoo Shin

KAIST

***Abstract*—Graphical Model (GM) has provided a popular framework for big data analytics because it often lends itself to distributed and parallel processing by utilizing graph-based 'local' structures. It models correlated random variables where in particular, the max-product Belief Propagation (BP) is the most popular heuristic to compute the most-likely assignment in GMs. In the past years, it has been proven that BP can solve a few classes of combinatorial optimization problems under certain conditions.**

Motivated by this, we explore the prospect of using BP to solve generic combinatorial optimization problems. The challenge is that, in practice, BP may converge very slowly and even if it does converge, the BP decision often violates the constraints of the original problem. This paper proposes a generic framework that enables us to apply BP-based algorithms to compute an approximate feasible solution for an arbitrary combinatorial optimization task. The main novel ingredients include (a) careful initialization of BP messages, (b) hybrid damping on BP updates, and (c) post-processing using BP beliefs. Utilizing the framework, we develop parallel algorithms for several large-scale combinatorial optimization problems including maximum weight matching, vertex cover and independent set. We demonstrate that our framework delivers high approximation ratio, speeds up the process by parallelization, and allows large-scale processing involving billions of variables.

***Keywords*-Combinatorial optimization, Belief propagation, Parallel algorithm, Maximum weighted matching**

I. Introduction

Graphical Models (GMs) provide a useful framework for modeling and processing real-world, large-scale data applications. From traditional big data analytics, such as page rank [1] and graph mining [2], to more recent deep learning [3], graphical models have been commonly applied for processing large-scale data-sets. GM is a particularly good fit for big data applications because it lends itself to fast parallel implementations by utilizing graph-based 'local' structures. Several modern programming models, such as GraphLab [4], GraphChi [5] and GraphX [6], enable distributed, parallel computation on GMs.

One of the most common computational tasks found in GM's applications is to compute the most-likely assignment to random variables, so-called a MAP (Maximum-A-Posteriori) estimate. This can be viewed as solving a large-scale optimization problem, which is becoming increasingly important for big data analytics since it presents a major computational bottleneck. Motivated by this, we explore the prospect of using GMs to solve optimization problems at scale. In particular, we propose a message-passing based algorithm to solve combinatorial optimization problems based on Belief Propagation (BP). The (max-product) BP algorithm is a well-studied heuristic that has been popularly used to approximately solve MAP optimization tasks. It is an iterative, message-passing algorithm proven to produce exact solutions for tree structured GMs. However, understanding on the performance of BP for loopy GMs has been quite limited.

Our goal is to design a highly accurate approximation algorithm based on BP that solves generic large-scale combinatorial optimization problems. The benefit of such an algorithm is that it inherently lends itself to parallel implementations, enabling fast performance and ensuring scalability. However, the challenge is that the BP algorithm is not guaranteed to be correct or even converge in general. Even if it converges to the correct solution, its convergence speed is too low for solving large-scale instances. Especially, when there are multiple optima (multiple solutions), BP is known to oscillate in many cases [7, 8, 9]. One can stop BP iterations without waiting for convergence, but then BP algorithms often produce infeasible solutions, i.e., violate the constraints of the targeted combinatorial optimization.

Contribution. We resolve the issues by designing a generic BP-based framework that computes highly accurate and feasible approximation solutions. The basic idea is to use a truncated BP algorithm in conjunction with existing heuristics to enforce a feasible solution in a way that ensures high approximation ratio. At a high level, the algorithm takes the following steps. First, given an optimization problem, we represent the problem in the MAP framework of GM. Second, we run the corresponding BP's message-passing and 'partially' solve the optimization problem. However, because BP often does not converge quickly, we run only a fixed number of BP iterations; we do not wait for convergence. Instead, to boost up the quality of BP decisions, we (a) carefully initialize the BP messages, (b) add a small noise to weights, and (c) apply a hybrid damping strategy on BP message updates (Section III-B). Finally, we apply existing heuristics as post-processing procedures to enforce the feasibility of the BP decision. In particular, we run known heuristics for a given combinatorial optimization problem by replacing the parameters of the original problems (e.g., edge weights) with BP beliefs. This ensures that the framework is applicable to any combinatorial optimization problems,

*The first two authors contributed equally to this work.

978-1-4799-9926-2/15/$31.00 ©2015 IEEE

while achieving a higher approximation ratio than existing heuristics.

In summary, this paper makes three key contributions:

1) **Practical BP-based algorithm design:** To the best of our knowledge, this paper is the first to propose a generic concept for designing BP-based algorithms that solve large-scale combinatorial optimization problems.
2) **Parallel implementation:** We also demonstrate that the algorithm is easily parallelizable. For the maximum weighted matching problem, this translates to 71x speed up while sacrificing only 0.1% accuracy compared to the state-of-art exact algorithm [10].
3) **Extensive empirical evaluation:** We evaluate our algorithms on three different combinatorial optimization problems on diverse synthetic and real-world data-sets. Our evaluation shows that the framework shows higher accuracy compared to other known heuristics.

Related Work. In the past years, the convergence and correctness of BP has been studied analytically for several classical combinatorial optimization problems, including matchings [7, 11, 12], perfect matchings [13], shortest paths [8], independent sets [14], network flows [9] and vertex covers [15]. The important common feature of these models is that BP converges to a correct assignment when the linear programming (LP) relaxation of the combinatorial optimization is tight, i.e., when it shows no integrality gap. However, LP tightness is an inevitable condition to guarantee the convergence of BP to the optimal solution, which is the main limitation of these theoretical studies towards wider applicability. Moreover, even if BP converges to the optimal solution, its convergence speed is often too slow for solving large-scale instances. There have been also empirical studies of BP-based algorithms for specific combinatorial optimization instances, including traveling salesman [16], graph partitioning [16], Steiner tree [17] and network alignment [18]. However, their focuses are not on large-scale instances and the running times of the proposed algorithms typically grow super-linearly with respect to the input size. In contrast, we provide a generic framework on designing BP-based scalable, parallel algorithms that are widely applicable to arbitrary large-scale combinatorial optimization problems.

Organization. We provide backgrounds on BP and combinatorial optimization problems in Section II. Section III describes our BP-based algorithm design, and Section IV provides details on its parallel implementation. We evaluate our algorithm on several combinatorial optimization problems in Section V. Finally, we conclude in Section VI.

II. Preliminaries

A. Graphical Model and Belief Propagation

A joint distribution of n (binary) random variables $Z = [Z_i] \in \{0,1\}^n$ is called a Graphical Model (GM) if it factorizes as follows: for $z = [z_i] \in \{0,1\}^n$,

$$\Pr[Z = z] \propto \prod_{i\in\{1,\ldots,n\}} \psi_i(z_i) \prod_{\alpha\in F} \psi_\alpha(z_\alpha),$$

where $\{\psi_i, \psi_\alpha\}$ are non-negative functions, so-called factors; F is a collection of subsets

$$F = \{\alpha_1, \alpha_2, ..., \alpha_k\} \subset 2^{\{1,2,\ldots,n\}}$$

(each α_j is a subset of $\{1, 2, \ldots, n\}$ with $|\alpha_j| \geq 2$); z_α is the projection of z onto dimensions included in α.[1] In particular, ψ_i is called a variable factor. Assignment z^* is called a maximum-a-posteriori (MAP) assignment if $z^* = \arg\max_{z\in\{0,1\}^n} \Pr[z]$. This means that computing a MAP assignment requires us to compare $\Pr[z]$ for all possible z, which is typically computationally intractable (i.e., NP-hard) unless the induced bipartite graph of factors F and variables z, the so-called factor graph, has a bounded tree width [19].

The max-product belief propagation (BP) is a popular heuristic for approximating the MAP assignment in GM. BP is implemented iteratively; at each iteration t, BP maintains four messages $\{m^t_{\alpha\to i}(c), m^t_{i\to\alpha}(c) : c \in \{0,1\}\}$ between every variable z_i and every associated $\alpha \in F_i$, where $F_i := \{\alpha \in F : i \in \alpha\}$. The messages are updated as follows:

$$m^{t+1}_{\alpha\to i}(c) = \max_{z_\alpha : z_i = c} \psi_\alpha(z_\alpha) \prod_{j\in\alpha\setminus i} m^t_{j\to\alpha}(z_j) \quad (1)$$

$$m^{t+1}_{i\to\alpha}(c) = \psi_i(c) \prod_{\alpha'\in F_i\setminus\alpha} m^t_{\alpha'\to i}(c). \quad (2)$$

One can reduce the complexity of messages by combining (1) and (2) as:

$$m^{t+1}_{i\to\alpha}(c) = \psi_i(c) \prod_{\alpha'\in F_i\setminus\alpha} \max_{z_{\alpha'} : z_i = c} \psi_{\alpha'}(z_{\alpha'}) \prod_{j\in\alpha'\setminus i} m^t_{j\to\alpha'}(z_j).$$

Given a set of messages $\{m_{i\to\alpha}(c), m_{\alpha\to i}(c) : c \in \{0,1\}\}$, the so-called BP marginal beliefs are computed as follows:

$$b_i[z_i] = \psi_i(z_i) \textstyle\prod_{\alpha\in F_i} m_{\alpha\to i}(z_i). \quad (3)$$

This BP algorithm outputs $z^{BP} = [z^{BP}_i]$ where

$$z^{BP}_i = \begin{cases} 1 & \text{if } b_i[1] > b_i[0] \\ ? & \text{if } b_i[1] = b_i[0] \\ 0 & \text{if } b_i[1] < b_i[0] \end{cases}.$$

It is known that z^{BP} converges to a MAP assignment after a sufficient number of iterations, if the factor graph is a tree and the MAP assignment is unique. However, if the graph contains cycles or MAP is not unique, the BP algorithm is not guaranteed to converge to a MAP assignment in general.

B. Belief Propagation for Combinatorial Optimization

The max-product BP can be applied to compute an approximate solution for any 'discrete' optimizations. This

[1] For example, if $z = [0, 1, 0]$ and $\alpha = \{1, 3\}$, then $z_\alpha = [0, 0]$.

section describes the maximum weight matching problem as an example. Given a graph $G = (V, E)$ and edge weights $w = [w_e] \in \mathbb{R}^{|E|}$, it finds a set of edges such that each vertex is connected to at most one edge in the set and the sum of edge weights in the set is maximized. The problem is formulated as the following IP (Integer Programming):

$$\begin{aligned} &\text{maximize} \quad w \cdot x \\ &\text{s.t.} \quad \sum_{e \in \delta(v)} x_e \leq 1, \ \forall v \in V, \quad x = [x_e] \in \{0,1\}^{|E|}, \end{aligned} \tag{4}$$

where $\delta(v)$ is the set of edges incident to vertex $v \in V$.

Now consider the following GM: for $x = [x_e] \in \{0,1\}^{|E|}$,

$$\Pr[X = x] \ \propto \ \prod_{e \in E} e^{w_e x_e} \prod_{v \in V} \psi_v(x_{\delta(v)}), \tag{5}$$

where $$\psi_v(x_{\delta(v)}) = \begin{cases} 1 & \text{if } \sum_{e \in \delta(v)} x_e \leq 1 \\ 0 & \text{otherwise} \end{cases}.$$

One can easily observe that the MAP assignments for GM (5) correspond to the (optimal) solution of IP (4). Therefore, one can use the max-product BP for solving the maximum weight matching problem, where the algorithm can be simplified using $a_{i \to j} = \log \frac{m_{i \to (i,j)}(0)}{m_{i \to (i,j)}(1)}$ as described in Algorithm 1.

Algorithm 1 BP for Maximum Weight Matching

(ITERATION) Calculate new messages as follows:

$$a_{i \to j}^{t+1} \leftarrow \max_{k \in \delta(i) \setminus \{j\}} \left\{ \max \left\{ w_{ik} - a_{k \to i}^{t}, 0 \right\} \right\}$$

(DECISION) For each edge $e = (i, j) \in E$, decide

$$x_e \ = \ \begin{cases} 1 & \text{if} \quad (a_{i \to j} + a_{j \to i}) < w_e \\ ? & \text{if} \quad (a_{i \to j} + a_{j \to i}) = w_e \\ 0 & \text{if} \quad (a_{i \to j} + a_{j \to i}) > w_e \end{cases}.$$

Note that one can design similar BP algorithms for arbitrary combinatorial optimization problems (e.g., minimum weight vertex cover and maximum weight independent set) as we demonstrate in Section V.

III. Algorithm Design

The main goal of this paper is to design BP-based parallel algorithms for solving combinatorial optimizations. To this end, one can design a BP algorithm as described in Section II-B. However (a) it might diverge or converge very slowly, (b) even if it converges quickly, the BP decision might be not correct, and (c) even worse, BP might produce an infeasible solution, i.e., it does not satisfy the constraints of the problem.

To address these issues, we propose a generic BP-based framework that provides highly accurate approximate solutions for combinatorial optimization problems. The framework has two steps, as shown in Figure 1. In the first phase, it runs a BP algorithm for a fixed number of iterations without waiting for convergence. Then, the second phase runs a known heuristic using BP beliefs instead of the original weights to output a feasible solution. Namely, the first and second phases are respectively designed for 'BP weight transforming' and 'post-processing'. In the following sections, we describe the two phases in more detail in reverse order. For illustration purposes, we focus on the maximum weight matching problem while the results are derived using Erdős–Rényi random graphs with 1000 vertices, average degree 100, and edge weights drawn from the uniform distribution over the interval $[0, 1]$. Later in Section V, we demonstrate the framework is applicable to any other combinatorial optimization problems.

A. Post-Processing Phase.

The decision on BP beliefs might give an infeasible solution. For example, in the maximum weight matching problem, BP decides $x_e = 0, 1$ based on the sign of $w_{ij} - (a_{i \to j} + a_{j \to i})$ (see Algorithm 1), but the edges of $x_e = 1$ might not form a matching even if BP converges, i.e., there exists a vertex v such that $\sum_{e \in \delta(v)} x_e > 1$.

To resolve the issue, we use post-processing by utilizing existing heuristics to the given problem that find a feasible solution. Applying post-processing ensures that the solution is at least feasible. In addition, our key idea is to replace the original weights by the logarithm of BP beliefs, i.e., the new weight on edge e becomes: $w_{ij} - (a_{i \to j} + a_{j \to i})$. After this, we apply known heuristics using the logarithm of BP beliefs to achieve higher accuracy.

For example, the following 'local' greedy algorithm can be used as a post-processing mechanism:

1. Initially, all vertices are 'unmatched', i.e., $x_e = 0$ for all $e \in E$.
2. Choose an arbitrary unmatched vertex i and match it to an unmatched vertex $j \neq i$ having the highest value in $w_{ij} - (a_{i \to j} + a_{j \to i})$, i.e., set $x_{ij} = 1$.
3. Keep iterating step 2 until no more vertex can be matched.

To confirm the effectiveness of the proposed post-processing mechanism, we compare it with the following two alternative post-processing schemes that remove edges to enforce matching after BP processing in a naive manner:

- Random: If there exists a vertex v such that $\sum_{e \in \delta(v)} x_e > 1$ on the BP decision, randomly select one edge and remove other edges.
- Weight: If there exists a vertex v such that $\sum_{e \in \delta(v)} x_e > 1$ on the BP decision, remove edges of smaller weight.

Figure 3(a) compares the approximation ratio obtained using BP-belief-based post-processing versus the naive post-processing heuristics (random and weight). It shows that the proposed BP-belief-based post-processing outperforms the rest. Note, the results in Figure 3 were obtained by first applying BP message passing for weight transformation. Next, we explain how this is done in our framework.

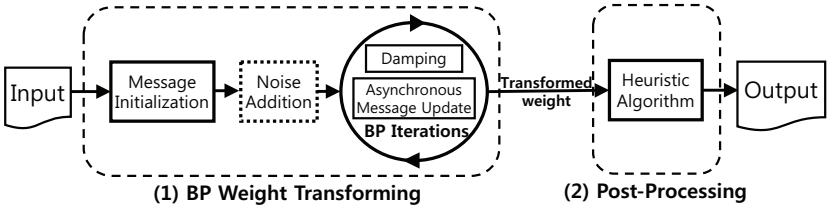

Figure 1: Overview of our generic BP-based framework

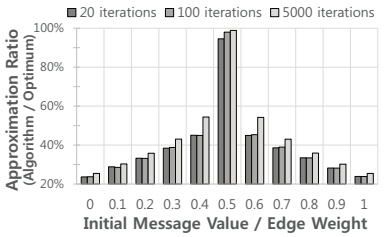

Figure 2: Effects of initial messages on the number of BP iterations. We set $a^0_{i\to j} = a^0_{j\to i} = c \cdot w_{ij}$ for a value c of x-axis.

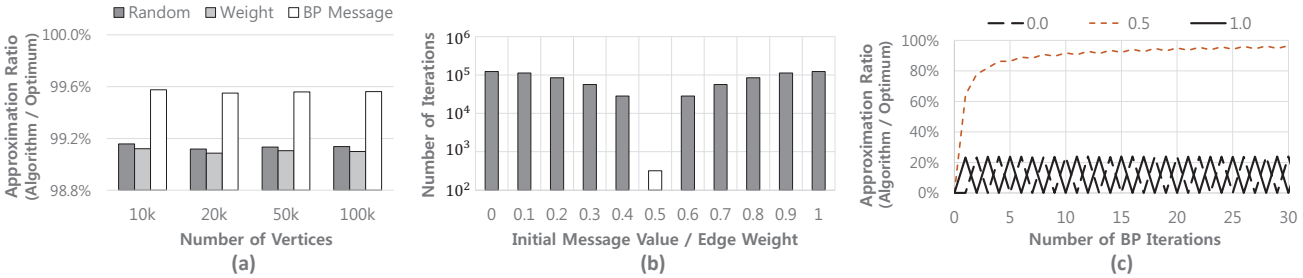

Figure 3: (a) Average approximation ratio for different post-processing schemes. We use a local greedy algorithm as a post-processing based on original weights and BP messages (i.e., beliefs). The 'Random selection' post-processing is also compared. (b) Effects of initial messages on the convergence of BP. We set $a^0_{i\to j} = a^0_{j\to i} = c \cdot w_{ij}$ for the value c of x-axis (c) Approximation ratio for different initial messages $a^0_{i\to j} = a^0_{j\to i} = 0, w_{ij}/2, w_{ij}$

B. BP Weight Transforming Phase

To improve the approximation quality and solve the convergence issues, we use three modifications to the standard BP algorithm: (1) careful initialization on messages, (2) noise addition and (3) hybrid damping on message updates.

Message Initialization. The standard message initialization is $m^0_{\alpha\to i} = m^0_{i\to\alpha} = 1$, i.e., $a^0_{i\to j} = 0$ for the maximum weight matching problem (see Algorithm 1). The convergence rate of BP depends on the initialized messages. As reported in Figure 3(b), we try different initializations, $a^0_{i\to j} = c \cdot w_{ij}$ for $0 \le c \le 1$, where the case $c = 0.5$ shows the fastest convergence. The main intuition we found for explaining such a phenomenon is as follows. If $a^0_{i\to j} = w_{ij}/2$, the BP decision at the initial step is neutral, i.e., x_e =?, since $a^0_{i\to j} + a^0_{j\to i} = w_{ij}$. On the other hand, if $a^0_{i\to j} = 0$, BP chooses $x_e = 1$ initially for all edges and most likely does $x_e = 0$ for most edges in the next step, i.e., it keeps oscillating between $x_e = 1$ and $x_e = 0$ for a while. The choice $a^0_{i\to j} = w_{ij}/2$ alleviates the fluctuation behavior of BP and boosts up its convergence speed.

We remind that, under our framework, BP runs only for a fixed number of iterations since it might converge too slowly, even with the initialization $a^0_{i\to j} = w_{ij}/2$, for practical purposes. With fixed number of iterations, careful initialization becomes even more critical as experimental results in Figure 3(c) and Figure 2 suggest. For example, if one runs 5000 iterations of BP, they show that the standard initialization ($a^0_{i\to j} = 0$) achieves at most 30% approximation ratio, while the proposed method ($a^0_{i\to j} = w_{ij}/2$) achieves 99%. Moreover, one can also observe that the advantage of more BP updates diminishes as the number of iterations becomes large. The observation holds for much larger graphs. Thus, we only run 100 BP iterations in our algorithm and do not wait for BP's convergence.

# vertices (# edges)	Approximation Ratio		Difference
	Multiple optima	Unique optimum	
1k (50k)	99.88 %	99.90 %	-0.02 %
5k (250k)	99.86 %	99.85 %	+0.01 %
10k (500k)	99.85 %	99.84 %	+0.01 %
20k (1M)	99.84 %	99.83 %	+0.01 %

Table I: Approximation ratio of BP for MWM with multiple optima and a unique optimum. We introduce a small noise to the edge weights and set the initial message by $a^0_{i\to j} = a^0_{j\to i} = w_{ij}/2$.

Noise Addition. The BP algorithm often oscillates when the MAP solution is not unique. To address this issue, we transform the original problem to one that has a unique solution with high probability by adding small noises to the weights, i.e., $w_e \leftarrow w_e + r_e$, where $r_e \in [-r, r]$ is a random number chosen independently across edges. We apply this to all cases. Here, one has to be careful in deciding the range r of noises. If r_e is too large, the quality of BP solution deteriorates because the optimal solution might have changed from the original problem. On the other hand, if r_e is too small compared to w_e, BP converges very slowly. To achieve a balance, we choose the range r of noise r_e as 10% of the minimum distance among weights. We find that this results in over 99.8% approximation ratio even when the solution is not unique, which has little difference with that of unique solution as shown in Table I.

Hybrid Damping. To boost up the convergence speed of BP updates further, we use a specific damping strategy to alleviate message oscillation. We update messages to be the

# vertices (# edges)	Approximation Ratio			
	no-damp(100)	damp(100)	no-damp(50) +damp(50)	damp(50) +no-damp(50)
10k (500k)	99.58 %	99.69 %	**99.83 %**	99.56 %
20k (1M)	99.55 %	99.68 %	**99.82 %**	99.56 %
50k (2.5M)	99.56 %	99.69 %	**99.83 %**	99.57 %
100k (5M)	99.56 %	99.69 %	**99.83 %**	99.57 %

Table II: Approximation ratio of BP without damping, BP with damping, BP with damping only for first 50 iterations, and BP with damping for last 50 iterations. We introduce a small noise to the edge weights and set the initial message by $a^0_{i\to j} = a^0_{j\to i} = w_{ij}/2$.

average of old and new messages.

$$a^{t+1}_{i\to j} \leftarrow \max_{k\in\delta(i)\setminus\{j\}} \left\{\max\left\{w_{ik} - a^t_{k\to i}, 0\right\}\right\}$$
$$a^{t+1}_{i\to j} \leftarrow (a^t_{i\to j} + a^{t+1}_{i\to j})/2.$$

We note that the damping strategy provides a similar effect as our proposed initialization $a^0_{i\to j} = w_{ij}/2$. Hence, if one uses both, the effect of one might be degraded due to the other. Due to this, we first run the half of BP iterations without damping (this is for keeping the effect of the proposed initialization) and perform the last half of BP iterations with damping. As reported in Table II, this hybrid approach outperforms other alternatives, including (a) no use of damping, (b) using damping in every iteration, and (c) damping in the first half of BP iterations and no-damping in the last half.

IV. Parallel Design and Implementation

This section addresses issues in parallelization of our algorithm. First, we introduce asynchronous message update that enables efficient parallelization of BP message passing. Second, we illustrate the issues in parallelizing post-processing. Finally, we describe the parallel implementations of our algorithm and their benefits.

A. Asynchronous Message Update

So far, we have assumed that there is only one thread, and BP messages are updated synchronously among vertices after calculating new message values. Thus, each iteration consists of two phases: message calculation phase and message update phase.

For parallelization, we first divide the graph by partitioning the vertices, and assign each partition to a single thread (see Section IV-C for details). However, if we naively parallelize the process using multiple threads, frequent synchronization may incur large overhead. Thus, we apply asynchronous message update where each vertex updates the message value right after new message value is calculated and eliminate synchronization point between iterations. This makes the process faster because of the reduced synchronization points. Figure 4 shows that performance improvement (speed up in running time) of asynchronous update over synchronous is up to 237% in our example graph for the maximum weight matching problem with 16 threads[2]; we leave detailed evaluation in Section V.

We now discuss its impact on approximation quality. Two factors marginally affect the approximation quality in opposite directions. First, updating the message value of each vertex right after its message calculation marginally improves the approximation ratio, as shown in Figure 5. Updating a message right after its calculation on a individual vertex basis implicitly has a similar effect to applying an additional iteration, which improves the quality. Second, having multiple threads run without synchronizing across iterations marginally degrade the approximation quality. The reason is that some threads run faster than others, and messages from the slower threads are not updated as frequently. We quantify this effect in Figure 5 and find that the impact is marginal; we still achieve 99.9% accuracy with multiple threads.

In summary, using asynchronous message updates, we speed up the run-time of the algorithm by up to 240%, while achieving 99.9% approximation ratio. In Section V, we show that the results also extend to other combinatorial optimization problems.

B. Local Post-Processing

The second phase of our algorithm runs existing heuristics for post-processing to enforce the feasibility of BP decisions. While the framework works with *any* heuristics-based post-processing methods, for the entire process to be parallel, it is important that the post-processing step is also parallel. An important criterion for efficient parallelization is locality of computation; if the post-processing heuristics can compute the result locally without requiring global knowledge, they can be easily parallelized. Moreover, if they do not require synchronization, the running time can be further reduced.

Fortunately, for most combinatorial optimization problems heuristics that match the two criteria exist. A local greedy algorithm, for example, enables local post-processing (i.e., it does not require global information). and require little synchronization. As reported in Section V, we also evaluate our algorithm in conjunction with other post-processing heuristics to demonstrate its flexibility.

C. Parallel Implementation

The BP algorithm is easy to parallelize because of its message passing nature. To demonstrate this, we parallelize our BP-based framework using three platforms:

- *GraphChi* enables large-scale graph computation on a personal computer by utilizing disk drive [5]. Using its API, we specify the BP message update rules to enable parallel message updates and scale the size of the problem up to billions of variables.
- *OpenMP* is a task-based parallelization library developed by Intel. Simply putting OpenMP pragma directive enables the compiler to support parallel computation.
- For our *pthread*-based implementation, we divide a single BP iteration into the smaller execution blocks

[2]We use a machine with two Intel Xeon(R) CPU E5-2690 @ 2.90GHz each with 8 cores.

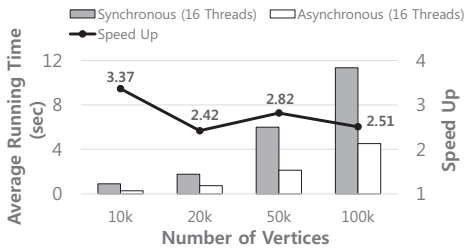

Figure 4: Average running time of our BP-based algorithm with synchronous message update and asynchronous message update. We apply all three modifications to BP of Section III-B.

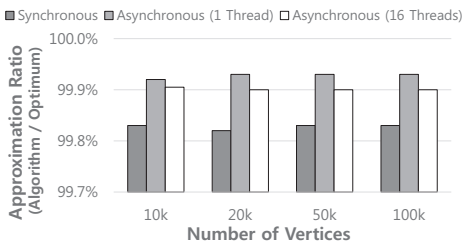

Figure 5: Approximation ratio of our BP-based algorithm with synchronous and asynchronous message update with 16 threads. We apply all three modifications to BP of Section III-B.

	# Vertices	# Edges
apache1	80k	230k
apache2	715k	2M
ecology2	1M	2M
G3_circuit	1.6M	3M
bone010	1M	23.4M

Table III: Summary of MWM data-sets

	# Vertices	Optimal Cost	
		MVC	MIS
frb-30-15	450	420	30
frb-45-21	945	900	45
frb-53-24	1,272	1,219	53
frb-59-26	1,534	1,475	59

Table IV: Summary of MWVC and MWIS data-sets

called tasks. Each iteration is divided into per-thread tasks. Because we have fixed multiple number of BP iterations, it concerns many more tasks. We put these tasks in a task queue. Initially all threads are assigned a task and the thread finished its task will be assigned the next task in the task queue. This minimizes the overlap between different iterations and synchronization points, which reduces the run time while obtaining high approximation ratio.

Algorithm 2 BP for Minimum Weight Vertex Cover

(ITERATION) Calculate new messages as follows:

$$a_{i\to j}^{t+1} \leftarrow \min\left\{ w_v + \sum_{k\in\delta(i)\setminus\{j\}} a_{k\to i}^{t}, 0 \right\}$$

(DECISION) For each vertex $i \in V$, decide

$$x_i \;=\; \begin{cases} 1 & \text{if} \quad \sum_{j\in\delta(i)} a_{j\to i} < -w_i \\ ? & \text{if} \quad \sum_{j\in\delta(i)} a_{j\to i} = -w_i \\ 0 & \text{if} \quad \sum_{j\in\delta(i)} a_{j\to i} > -w_i \end{cases}$$

Algorithm 3 BP for Maximum Weight Independent Set

(ITERATION) Calculate new messages as follows:

$$a_{i\to j}^{t+1} \leftarrow \max\left\{ w_v - \sum_{k\in\delta(i)\setminus\{j\}} a_{k\to i}^{t}, 0 \right\}$$

(DECISION) For each vertex $i \in V$, decide

$$x_i \;=\; \begin{cases} 1 & \text{if} \quad \sum_{j\in\delta(i)} a_{j\to i} < w_i \\ ? & \text{if} \quad \sum_{j\in\delta(i)} a_{j\to i} = w_i \\ 0 & \text{if} \quad \sum_{j\in\delta(i)} a_{j\to i} > w_i \end{cases}$$

V. Evaluation

We evaluate our BP framework using three popular combinatorial optimization problems: maximum weight matching, minimum weight vertex cover and maximum weight independent set problem. We perform extensive empirical evaluation to demonstrate the benefit of our algorithm for the following three questions:

1) *Does the BP-based algorithm provide high approximation ratio?*
2) *Can the algorithm achieve speed-up due to parallel implementations?*
3) *Can it solve large-scale problems involving billions of variables?*

We already introduced the IP formulation of the maximum weight matching (MWM) in (4), where those of the minimum weight vertex cover (MWVC) and maximum weight independent set problem (MWIS) are as follows:

$$\begin{aligned} \textbf{MWVC:}\quad & \text{minimize} \quad w \cdot x \\ & \text{subject to} \quad x_u + x_v \geq 1,\ \forall e = (u,v) \in E \\ & \qquad x = [x_v] \in \{0,1\}^{|V|} \end{aligned} \tag{6}$$

$$\begin{aligned} \textbf{MWIS:}\quad & \text{maximize} \quad w \cdot x \\ & \text{subject to} \quad x_u + x_v \leq 1,\ \forall e = (u,v) \in E \\ & \qquad x = [x_e] \in \{0,1\}^{|V|}. \end{aligned} \tag{7}$$

We provide descriptions of BP algorithms in Algorithm 2–3 for MWVC and MWIS. As we propose in Section III-B, we choose initial BP messages for neutral decisions: $a_{j\to i}^{0} = -w_{ij}/|\delta(i)|$ for vertex cover and $a_{j\to i}^{0} = w_{ij}/|\delta(i)|$ for independent set.

A. Experiment Setup

In our experiments, both real-world and synthetic data-sets are used for evaluation. For MWM, we used data-sets from the university of Florida sparse matrix collection [20] summarized in Table III. For larger scale synthetic evaluation, we generate Erdős-Rényi random graphs (up to 50 million vertices with 2.5 billion edges) with average vertex degree of 100 with edge weights drawn independently from the uniform random distribution over the interval $[0, 1]$. For MWVC and MWIS, we use the frb-series from BHOSLIB [21] summarized in Table IV, where it also contains the optimal solutions. We note that we perform no experiment using synthetic data-sets for MWVC and MWIS since they are NP-hard problems, i.e., impossible to compute the optimal solutions. On the other hand, for MWM the Edmonds' Blossom algorithm [10] can compute the optimal solution in polynomial time. All experiments in this section are conducted on a machine with Intel Xeon(R) CPU E5-2690 @ 2.90GHz with 8 cores and 8 hyperthreads with 128GB of memory, unless otherwise noted.

		Approximation Ratio	
		Serial BP (1 thread)	Parallel BP (16 threads)
Synthetic # vertices (# edges)	500k (25M)	99.93 %	99.90 %
	1M (50M)	99.93 %	99.90 %
	2M (100M)	99.94 %	99.91 %
	5M (250M)	99.93 %	99.90 %
Real-world Data-set (Florida)	apache1	100.0 %	100.0 %
	apache2	100.0 %	100.0 %
	ecology2	100.0 %	100.0 %
	G3_circuit	99.95 %	99.95 %
	bone010	99.11 %	99.12 %

Table V: MWM: Approximation ratio of our BP-based algorithm on synthetic and sparse matrix collection data-sets [20].

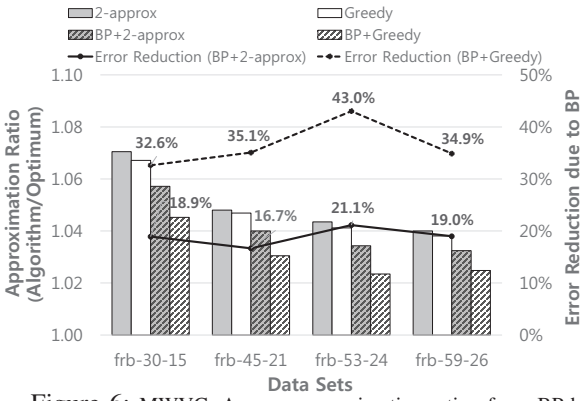

Figure 6: MWVC: Average approximation ratio of our BP-based algorithm, the 2-approximation algorithm and the greedy algorithm on frb-series data-sets.

B. Approximation Ratio

We now demonstrate our BP-based approximation algorithm produces highly accurate results. In particular, we show that our BP-based algorithms outperform well-known heuristics for MWVC, MWIS and closely approximate exact solutions for MWM for all cases we evaluate.

Maximum Weight Matching. For MWM, we compare the approximation qualities of serial, synchronous BP (i.e., one thread) in Section III and parallel, asynchronous implementation (i.e., using multiple threads) in Section IV on both synthetic and real-world data-sets, where we compute the optimal solution using the Blossom algorithm [10] to measure the approximation ratios. Table V summarize our experimental results for MWM for the synthetic data-sets and the Florida data. Our BP-based algorithm achieves 99% to 99.9% approximation ratios.

Minimum Weight Vertex Cover. For MWVC, we use two post-processing procedures: greedy and 2-approximation algorithm [22]. For the local greedy algorithm, we choose a random edge and add one of its adjacent vertices with a smaller weight until all edges are covered. We compare the approximation qualities of our BP-based algorithm compared to the cases when one uses only the greedy algorithm and the 2-approximation algorithm. Figure 6 shows the experimental results for the two post-processing heuristics. The results show that our BP-based weight transformation enhances the approximation quality of known approximation heuristics by up to 43%.

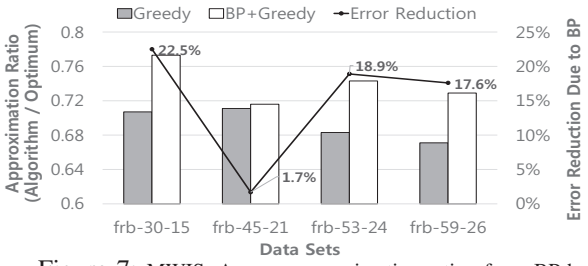

Figure 7: MWIS: Average approximation ratio of our BP-based algorithm and the greedy algorithm on frb-series data-sets.

Maximum Weight Independent Set. For MWIS, the experiment was performed on frb-series data-sets. We use a greedy algorithm as the post-processing procedure, which selects vertices in the order of higher weights until no vertex can be selected without violating the independent set constraint. We compare the approximation qualities of our BP-based algorithm and the standard greedy algorithm. Figure 7 shows that our BP-based framework enhances the approximation ratio of the solution by 2% to 23%.

C. Parallelization Speed-up

One of the important advantages of our BP-based algorithm is that it is fast, while delivering high approximation guarantees. In this section, we focus on the speed-up due to parallelization. Figure 8 compares the running time of the Blossom algorithm and our BP-based algorithm with 1 single core and 16 cores. With five million vertices, our asynchronous parallel implementation is eight times faster than the synchronous serial implementation, while still retaining 99.9% approximation ratio as reported in Table V. To demonstrate the overall benefit in context, we compare its running time with that of the current fastest implementation of the Blossom algorithm due to Kolmogorov [10]. Here, we note that the Blossom algorithm is inherently not easy to parallelize. For parallel implementation, we report results for our pthread implementation , but the OpenMP implementation also show comparable performance. For 20 million vertices (one billion edges), it shows that the running time of our algorithm can be accelerated by up to 71 times than the Blossom algorithm, while sacrificing 0.1% of accuracy. The running time gap is expected be more significant for larger graphs since the running times of our algorithm and the Blossom algorithm are linear and cubic with respect to the number of vertices, respectively. We also experiments our algorithms for other problems to demonstrate the parallelization speedup. Due to the space limitation, we only report that for the minimum weight vertex cover problem in Figure 9.

D. Large-scale Optimization

Our algorithm can also handle large-scale instances because it is based on GMs that inherently lend itself to parallel and distributed implementations. To demonstrate this, we create a large-scale instance containing up to 50 million vertices and 2.5 billion edges. We experiment our algorithm

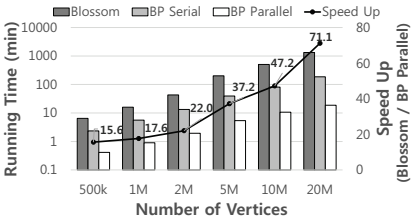

Figure 8: MWM: Running time of Blossom algorithm and our BP-based algorithms.

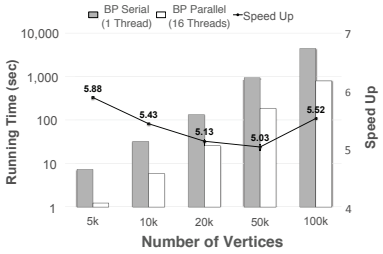

Figure 9: MWVC: Running time of our parallel BP-based algorithm (pthread implementation) on large-scale graphs.

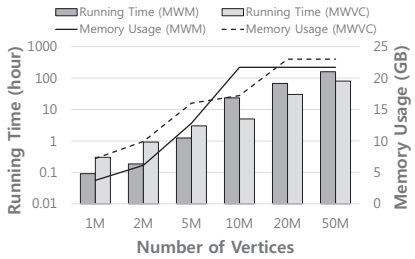

Figure 10: MWM and MWVC: Running time and memory usage of GraphChi-based implementation on large-scale graphs.

using GraphChi on a single consumer level machine with i7 CPU and 24GB of memory. Figure 10 shows the running time and memory usage of our algorithm for MWM and MWVC on large data-sets. For large graphs, GraphChi partitions them to load parts of them in memory, while storing the rest on disk by leveraging graph-based 'local' structures. Thus, we were able to solve problems with 2.5 billion edges on a single machine. In contrast, Kolmogorov's implementation [10] of the Blossom algorithm cannot handle such large graphs because distributed processing is difficult (e.g., it cannot handle more than 6M vertices on the same machine). Similarly, our algorithm can be run on multiple machines to scale to even larger problems. However, we leave this as future work.

VI. Conclusion

This paper explores the possibility of applying the BP algorithm to solve generic combinatorial optimizations at scale. We propose BP-based algorithm that achieves high approximation ratio and allows parallel implementation. We evaluate the algorithm's effectiveness and performance by applying our framework on three popular combinatorial optimization problems. Our evaluation shows that the algorithm outperforms existing approximation algorithms across many instances and is able to solve large-scale problems with billions of variables. We believe our BP-based framework is of broader interest for a wider class of large-scale optimization tasks.

Acknowledgment

This work was supported in part by Institute for Information & Communications Technology Promotion (IITP) granted by the Korea government (MSIP) (No.B0126-15-1078, Creation of PEP based on automatic protocol behavior analysis and Resource management for hyper connected for IoT Services); the Center for Integrated Smart Sensors funded by MSIP as Global Frontier Project (CISS-2011-0031863); and Asian Office of Aerospace Research and Development (AOARD) Project (FA2386-14-1-4058).

References

[1] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.

[2] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 2, 2006.

[3] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *International Conference on Artificial Intelligence and Statistics*, 2009, pp. 448–455.

[4] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012. [Online]. Available: http://dx.doi.org/10.14778/2212351.2212354

[5] A. Kyrola, G. E. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc." in *OSDI*, vol. 12, 2012, pp. 31–46.

[6] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *Proceedings of OSDI*, 2014, pp. 599–613.

[7] S. Sanghavi, D. Malioutov, and A. Willsky, "Belief propagation and lp relaxation for weighted matching in general graphs," *Information Theory, IEEE Transactions on*, vol. 57, no. 4, pp. 2203–2212, 2011.

[8] N. Ruozzi and S. Tatikonda, "st paths using the min-sum algorithm," in *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*. IEEE, 2008, pp. 918–921.

[9] D. Gamarnik, D. Shah, and Y. Wei, "Belief propagation for min-cost network flow: Convergence and correctness," *Operations Research*, vol. 60, no. 2, pp. 410–428, 2012.

[10] V. Kolmogorov, "Blossom v: a new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 43–67, 2009.

[11] M. Bayati, D. Shah, and M. Sharma, "Maximum weight matching via max-product belief propagation," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*. IEEE, 2005, pp. 1763–1767.

[12] B. C. Huang and T. Jebara, "Loopy belief propagation for bipartite maximum weight b-matching," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 195–202.

[13] M. Bayati, C. Borgs, J. Chayes, and R. Zecchina, "Belief propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions," *SIAM Journal on Discrete Mathematics*, vol. 25, no. 2, pp. 989–1011, 2011.

[14] S. Sanghavi, D. Shah, and A. S. Willsky, "Message passing for maximum weight independent set," *Information Theory, IEEE Transactions on*, vol. 55, no. 11, pp. 4822–4834, 2009.

[15] S. Park and J. Shin, "Max-product belief propagation for linear programming: Convergence and correctness," *arXiv preprint arXiv:1412.4972*, 2014.

[16] S. Ravanbakhsh, R. Rabbany, and R. Greiner, "Augmentative message passing for traveling salesman problem and graph partitioning," in *Advances in Neural Information Processing Systems*, 2014, pp. 289–297.

[17] M. Bayati, C. Borgs, A. Braunstein, J. Chayes, A. Ramezanpour, and R. Zecchina, "Statistical mechanics of steiner trees," *Physical review letters*, vol. 101, no. 3, p. 037208, 2008.

[18] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang, "Algorithms for large, sparse network alignment problems," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 2009, pp. 705–710.

[19] V. Chandrasekaran, N. Srebro, and P. Harsha, "Complexity of inference in graphical models," in *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*. AUAI Press, 2008, pp. 70–78.

[20] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[21] "bhoslib benchmark set." [Online]. Available: http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark

[22] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.