# Fast $b$-Matching via Sufficient Selection Belief Propagation

**Bert Huang**
Computer Science Department
Columbia University
New York, NY 10027
bert@cs.columbia.edu

**Tony Jebara**
Computer Science Department
Columbia University
New York, NY 10027
jebara@cs.columbia.edu

## Abstract

This article describes scalability enhancements to a previously established belief propagation algorithm that solves bipartite maximum weight $b$-matching. The previous algorithm required $O(|V| + |E|)$ space and $O(|V||E|)$ time, whereas we apply improvements to reduce the space to $O(|V|)$ and the time to $O(|V|^{2.5})$ in the expected case (though worst case time is still $O(|V||E|)$). The space improvement is most significant in cases where edge weights are determined by a function of node descriptors, such as a distance or kernel function. In practice, we demonstrate maximum weight $b$-matchings to be solvable on graphs with hundreds of millions of edges in only a few hours of compute time on a modern personal computer without parallelization, whereas neither the memory nor the time requirement of previously known algorithms would have allowed graphs of this scale.

## 1  INTRODUCTION

The maximum weight perfect $b$-matching problem is a generalization of maximum weight matching in which the solver is given a weighted graph and a set of target degrees, and must output the maximum weight induced subgraph such that each node has its target number of neighbors. The problem is solvable in $O(|V||E|)$ time with min-cost flow methods (Fremuth-Paeger and Jungnickel, 1999). In problems with dense graphs, the running time for $b$-matching solvers is

$O(N^3)$, where $N = |V|$. Huang and Jebara (2007) introduced a belief propagation algorithm which has the same asymptotic running time guarantee $O(N^3)$ but is lightweight and has much smaller constant factors on running time than other available solvers. In modern applications, however, the more obstructive bottleneck is the $O(N^2)$ space requirement to store messages from each node to each of its candidate neighbors. While it is possible to wait for time-intensive jobs to run, a task that requires too much storage is further burdened by the need for complicated memory swapping strategies.

This article presents an improved algorithm for weighted $b$-matching that significantly reduces the memory cost and the running time for solving $b$-matching. Specifically, in problems where the edge weights are determined by a function of node descriptors, the space requirement is reduced to $O(N)$ and the running time can be reduced to $O(N^{2.5})$ in some cases (but no worse than previous algorithms in adversarial cases). Both improvements are on each iteration of belief propagation, and the resulting algorithm computes the original belief updates *exactly*, so any previous analysis of the number of iterations necessary for convergence remains intact. The memory bottleneck is reduced by unrolling one level of recursion in the belief updates such that the explicit belief need never be stored, and the running time improvement is achieved by a variant of the algorithm by McAuley and Caetano (2010), in which speedups are available by decomposing a maximization procedure into the maximization of two components.

**Related Work.**  This article extends the belief propagation $b$-matching algorithm first introduced by Huang and Jebara (2007), which is proven to converge in $O(|V|)$ iterations with a constant depending on the difference between the maximum weight edge and the minimum weight edge as well as the difference between the maximum weight $b$-matching and the second best $b$-matching. This algorithm was further analyzed by Sanghavi et al. (2007) and Bayati et al. (2007), who

showed independently that the algorithm is guaranteed to converge if and only if the linear programming relaxation of the integer program formulation of $b$-matching is tight. This result confirms the previous theorem that the algorithm converges on bipartite problems and further extends guaranteed convergence to some non-bipartite cases. The 1-matching with *iid*, random weights was further analyzed by Salez and Shah (2009), where the surprising result was proven that the algorithm converges with high probability in $O(1)$ iterations and, thus, costs $O(|V|^2)$ time overall, which is optimal, as it is equivalent to the time needed to read the input edge weights.

In addition to classical optimization tasks, such as discrete resource allocation, weighted $b$-matching has been shown to be a useful tool for various machine learning tasks, including semi-supervised learning, spectral clustering, graph embedding, and manifold learning (Jebara et al., 2009; Jebara and Shchogolev, 2006; Shaw and Jebara, 2007, 2009).

Weighted $b$-matching solvers can also be used as drivers for a *maximum a posteriori* estimation procedure for graph structure given edge likelihoods and soft degree priors (Huang and Jebara, 2009). The general formulation allows for concave penalty functions on the degrees of nodes by constructing an augmented graph with auxiliary edges encoding the degree penalties. The augmented graph has at most double the nodes of the original graph, so the asymptotic running time of the algorithm is equivalent to the running time of the $b$-matching solver.

For graphs restricted to nonnegative integer weights, the bipartite maximum weight 1-matching problem was shown to be solvable in $O(\sqrt{|V|}|E|\log(|V|))$ time by Gabow and Tarjan (1989). An $\tilde{O}(|V|^{2.376})$ randomized algorithm which succeeds with high probability was revealed by Sankowski (2009). A $(1\text{-}\epsilon)$ approximation algorithm for nonbipartite maximum weight matching with real weights was given by Duan and Pettie (2010), which runs in $O(|E|\epsilon^{-2}\log^3|V|)$ time.

**Outline.** The remainder of this paper is organized as follows. Section 2 describes the proposed algorithm in detail and provides analysis. Section 3 describes empirical evaluation of the proposed algorithm on synthetic and real data, including comparisons with a state-of-the-art maximum weight matching solver. Finally, Section 4 concludes with a brief discussion.

# 2 ALGORITHM DESCRIPTION

This section describes the proposed algorithm, which is derived from the previous belief propagation approaches for $b$-matching and incorporates some further improvements to improve scalability. First, we provide a formal definition of the problem; then we describe the algorithm. Finally, we provide some analysis showing the correctness of the enhanced algorithm as well as the speed and space improvements.

## 2.1 Dense Maximum Weight $b$-Matching

The *bipartite dense maximum weight perfect b-matching problem* (abbreviated as $b$-matching) is, given a dense, bipartite graph, in which all pairs of points that cross bipartitions have candidate edges and a target degree for each node, to find the maximum weight induced subgraph such that the nodes in the subgraph have their target degrees. Formally, the solver is given node descriptors $\{x_1, \ldots, x_{m+n}\}$ drawn from space $\Omega$, a weight function $W : (\Omega, \Omega) \mapsto \mathbb{R}$, and a set of target degrees $\{b_1, \ldots, b_{m+n}\}$, where each $b_i \in \mathbb{N}$. The goal is to output a symmetric, binary adjacency matrix $\mathbf{A} \in \mathbb{B}^{(m+n) \times (m+n)}$ whose entries $A_{ij} = 1$ for all matched edges $(x_i, x_j)$ and are otherwise zero. The optimization can also be written as

$$
\begin{aligned}
\underset{\mathbf{A}}{\text{argmax}} \quad & \sum_{i=1}^{m} \sum_{j=m+1}^{m+n} A_{ij} W(x_i, x_j) \\
\text{s.t.} \quad & \sum_{j=1}^{m+n} A_{ij} = b_i, \forall i, \qquad A_{ij} = A_{ji}, \forall (i,j).
\end{aligned}
$$

In particular, we consider the bipartite scenario, where edges may only be matched between nodes $\{x_1, \ldots, x_m\}$ and nodes $\{x_{m+1}, \ldots, x_{m+n}\}$ but not within each set. This can be implemented with abuse of notation by defining the weight function $W$ to output $-\infty$ for any edges within bipartitions. This same problem can be expressed in many other forms, including graph notations using node and edge sets, but when considering the dense bipartite form of the problem, it is convenient to use matrix notation.

## 2.2 Linear Memory $b$-Matching Belief Propagation

In this section, we describe the method to reduce memory usage of $b$-matching via belief propagation to $O(N)$, where the total number of nodes $N = m + n$. First, we review the results from previous work (Bayati et al., 2005; Huang and Jebara, 2007; Sanghavi et al., 2007) defining a simplified update rule for message updates, which allows for the standard $O(N^2)$ space and $O(N^2)$ per-iteration running time. A key component of the simplified belief propagation algorithm is the selection operation. This is the operation that finds the $k$'th largest element of a set for some index $k$. For notational convenience, denote the selection operation

over any set $S$ as

$$\sigma_k(S) = s \in S \text{ where } |\{t \in S | t \geq s\}| = k.$$

Belief propagation maintains a belief value for each edge, which, in the dense case, is conveniently represented as a matrix $\mathbf{B}$, where entry $B_{ij}^t$ is the belief value for the edge between $x_i$ and $x_j$ at iteration $t$. The simplified update rule for each belief is

$$B_{ij}^t = W(x_i, x_j) - \sigma_{b_j}(\{B_{jk}^{t-1} | k \neq i\}). \quad (1)$$

In the above equation and for the remainder of this text, indices range from 1 to $(m+n)$, unless otherwise noted, and are omitted for cleanliness.

The key insight for reducing memory usage is that the full beliefs never need to be stored (not even the compressed messages). Instead, by unrolling one level of recursion, all that need to be stored are the *selected* beliefs, because the selection operation in Equation (1) only weakly depends on index $i$. That is, the selection operation is over all indices excluding $i$, which means the selected value will be either the $b_j$'th or the $b_j + 1$'th greatest element,

$$\sigma_{b_j}(\{B_{jk}^{t-1} | k \neq i\}) \in$$
$$\{\sigma_{b_j}(\{B_{jk}^{t-1} | k\}), \sigma_{b_j+1}(\{B_{jk}^{t-1} | k\})\}.$$

Thus, once each row of the belief matrix $\mathbf{B}$ is updated, these two selected values can be computed and stored, and the rest of the row can be deleted from memory. Any further reference to $\mathbf{B}$ is therefore abstract, as it will never be fully stored. Any entry of the belief matrix can be computed in an online manner from the stored selected value. Let $\alpha_j$ be the negation of the $b_j$'th selection and $\beta_j$ be that of the $b_j + 1$'th selection. Then the update rules for these parameters are

$$\alpha_j^t = -\sigma_{b_j}(\{B_{jk}^{t-1}|k\}), \quad \beta_j^t = -\sigma_{b_j+1}(\{B_{jk}^{t-1}|k\}), \quad (2)$$

and the resulting belief lookup rule is

$$B_{ij}^t = W(x_i, x_j) + \begin{cases} \alpha_j^t & \text{if } A_{ji}^t \neq 1 \\ \beta_j^t & \text{otherwise.} \end{cases} \quad (3)$$

After each iteration, the current estimate of $A$ is

$$A_{ij}^t = \begin{cases} 1 & \text{if } B_{ij}^{t-1} \geq \alpha_i^t \\ 0 & \text{otherwise,} \end{cases}$$

which is computed when the $\alpha$ and $\beta$ values are updated in Equation (2). When this estimate is a valid $b$-matching, i.e., when the columns of $A_{ij}$ sum to their target degrees, the algorithm has converged to the solution. The algorithm can be viewed as simply computing each row of the belief matrix and performing the selections on that row and is summarized in Algorithm 1.

---

**Algorithm 1** Belief Propagation for $b$-Matching. Computes the adjacency matrix of the maximum weight $b$-matching.

1: $\alpha_j^0, \beta_j^0 \leftarrow 0, \forall j$
2: $\mathbf{A}^0 \leftarrow [0]$
3: $t \leftarrow 1$
4: **while** not converged **do**
5:     **for all** $j \in \{1, \dots, m+n\}$ **do**
6:         $A_{jk}^t \leftarrow 0, \forall k$
7:         $\alpha_j^t \leftarrow -\sigma_{b_j}(\{B_{jk}^{t-1}|k\})$ {Algorithm 2}
8:         $\beta_j^t \leftarrow -\sigma_{b_j+1}(\{B_{jk}^{t-1}|k\})$
9:         **for all** $\{k | B_{jk}^{t-1} \geq \alpha_j^t\}$ **do**
10:            $A_{jk}^t \leftarrow 1$
11:         **end for**
12:     **end for**
13:     delete $\mathbf{A}^{t-1}$, $\alpha^{t-1}$ and $\beta^{t-1}$ from memory
14:     $t \leftarrow t + 1$
15: **end while**

---

### 2.3 Sufficient Selection

This section describes the running time enhancement in the proposed algorithm, which is a variation of the faster belief propagation algorithm proposed by McAuley and Caetano (2010). The enhancements aim to reduce the running time of each iteration by exploiting the nature of the quantities being selected. In particular, the key observation is that each belief is a sum of two quantities: a weight and an $\alpha$ or $\beta$ value. These quantities can be sorted in advance, outside of the inner (row-wise) loop of the algorithm, and the selection operation can be performed without searching over the entire row, significantly reducing the amount of work necessary. This is done by testing a stopping criterion that guarantees no further belief lookups are necessary.

Some minor difficulties arise, however, when sorting each component, so the algorithm by McAuley and Caetano (2010) does not directly apply as-is. First, the weights cannot always be fully sorted. In general, storing full order information for each weight between all pairs of nodes requires quadratic space, which is impossible with larger data sets. Thus, the proposed algorithm instead stores a cache of the heaviest weights for each node. In some special cases, such as when the weights are a function of Euclidean distance, data structures such as *kd-trees* can be used to implicitly store the sorted weights. This construction can provide one possible variant to our main algorithm.

Second, the $\alpha$-$\beta$ values require careful sorting, because the true belief updates mostly include $\alpha^t$ terms but a few $\beta^t$ terms. Specifically, the indices that index the greatest $b_j$ elements of the row should use $\beta^t$. One way

to handle this technicality is to first compute the sort-order of the $\alpha^t$ terms and, on each row, correct the ordering using a binary search-like strategy for each index in the selected indices. This method is technically a logarithmic time procedure, but requires some extra indexing logic that creates undesirable constant time penalties. Another approach, which is much simpler to implement and does not require extra indexing logic, is to use the sort-order of the $\beta^t$'s and adjust the stopping criterion to account for the possibility of unseen $\alpha^t$ values.

Since the weights do not change during belief propagation, at initialization, the algorithm computes index cache $\mathbf{I} \in \mathbb{N}^{(m+n) \times c}$ of cache size $c$, which is a parameter set by the user, where entry $I_{ik}$ is the index of the $k$'th largest weight connected to node $x_i$ and, for $u = I_{ik}$,

$$W(x_i, x_u) = \sigma_k(\{W(x_i, x_j)|j\}).$$

At the end of each iteration, the $\beta^t$ values are similarly sorted and stored in index vector $\mathbf{e} \in \mathbb{N}^{m+n}$, where, for $v = e_k$, entry $\beta_v^t = \sigma_k(\beta_j^t|j)$.

The selection operation from (2) is then computed by checking the beliefs corresponding to the sorted weight and $\beta$ indices. At each step, maintain a set $S$ of the greatest $b_j + 1$ beliefs seen so far. These provide tight lower bounds on the true $\alpha - \beta$ values. At each stage of this procedure, the current estimates for $\alpha_j^t$ and $\beta_j^t$ are

$$\tilde{\alpha}_j^t \leftarrow \sigma_{b_j}(S), \text{ and } \tilde{\beta}_j^t \leftarrow \min(S).$$

Incrementally scan the beliefs for both index lists $(\mathbf{I})_j$ and $\mathbf{e}$, computing for incrementing index $k$, $B_{iI_{ik}}$ and $B_{ie_k}$. Each of these computed beliefs is compared to the beliefs in set $S$ and if any member of $S$ is less than the new belief, the new belief replaces the minimum value in $S$.[1]) This maintains $S$ as the set of the greatest $b_j + 1$ elements seen so far.

At each stage, we bound the greatest possible unseen belief as the sum of the least weight seen so far from the sorted weight cache and the least $\beta$ value so far from the $\beta$ cache. Once the estimate $\tilde{\beta}_j^t$ is less than or equal to this sum, the algorithm can exit because further comparisons are unnecessary. Algorithm 2 summarizes the sufficient selection procedure.

---

[1]A small hash table for the indices will indicate whether an index has been previously visited in $O(1)$ time per lookup. For small values of $b_j$ where ($b_j << n + m$), a linear scan through $S$ to find the minimum is sufficiently fast, but a priority queue can be used to achieve sub-linear time insertion and replacement when $b_j$ is large.

---

**Algorithm 2** Sufficient Selection. Given sort-order of $\beta^t$ values and partial sort-order of weights, selects the $b_j$'th and $b_j + 1$'th greatest beliefs of row $j$.

1: $k \leftarrow 1$
2: bound $\leftarrow \infty$
3: $S \leftarrow \emptyset$
4: $\tilde{\alpha}_j{}^t \leftarrow -\infty$
5: $\tilde{\beta}_j{}^t \leftarrow -\infty$
6: **while** $\tilde{\beta}_j{}^t <$ bound **do**
7:     **if** $k \leq c$ **then**
8:         $u \leftarrow I_{jk}$
9:         **if** ($u$ is unvisited and $(B_{ju}^{t-1} > \min(S))$ **then**
10:             $S \leftarrow (S \setminus \min(S)) \cup B_{ju}^{t-1}$
11:         **end if**
12:     **end if**
13:     $v \leftarrow e_k$
14:     **if** ($v$ is unvisited and $(B_{jv}^{t-1} > \min(S))$ **then**
15:         $S \leftarrow (S \setminus \min(S)) \cup B_{jv}^{t-1}$
16:     **end if**
17:     bound $\leftarrow W(x_j, x_u) + \beta_v^{t-1}$
18:     $\tilde{\alpha}_j^t \leftarrow \sigma_{b_j}(S)$
19:     $\tilde{\beta}_j^t \leftarrow \sigma_{b_j+1}(S)$
20:     $k \leftarrow k + 1$
21: **end while**
22: $\alpha_j^t \leftarrow \tilde{\alpha}_j^t$
23: $\beta_j^t \leftarrow \tilde{\beta}_j^t$

## 2.4 Implementation Details

The implementation of Algorithms 1 and 2 used in the experiments of Section 3 is in C. To perform the initial iteration, during which the weight cache is constructed, our program uses the Quick Select algorithm, which features the same pivot-based partitioning strategy as Quick Sort to perform selection in (average case) $O(N)$ time per node (Cormen et al., 2001). For low-dimensional data and distance-based weights, we can run the same selection using a *kd-tree* and provide the index cache as an input to the program. [2]

## 2.5 Analysis

In this section, we analyze the correctness, space and running time requirements of the proposed algorithm. First, we verify that the bound from the sufficient selection procedure holds even though it is computed using only the $\beta_j^t$ values, when many of the beliefs are actually computed using $\alpha_j^t$ values.

**Claim 1.** *At each stage of the scan, where set $S$ contains the $b_j + 1$ greatest beliefs corresponding to the first through $k$'th indices of $(\mathbf{I})_j$ and $\mathbf{e}$, the following*

---

[2]A newer C++ version of the solver is available at http://www.cs.columbia.edu/~bert/code/bmatching/.

*properties are invariant: the current estimates bound the true values from below, $\tilde{\alpha}_j^t \leq \alpha_j^t$, $\tilde{\beta}_j^t \leq \beta_j^t$, and the greatest unexplored belief is no greater than the sum of the least cached weight and the least $\beta_j^{t-1}$ value,*

$$W(x_j, x_u) + \beta_v^{t-1} \geq$$
$$\max\left(\left\{B_{j\ell}^{t-1} | \ell \in \{e_{k+1}, \ldots, e_{m+n}\}\right\}\right), \quad (4)$$

*where $u = I_{jk}$ and $v = e_k$.*

*Proof.* The first two inequalities follow from the fact that the algorithm is selecting from but has not necessarily seen the full row yet. The third inequality (4) is the result of two bounds. First, the beliefs in the right-hand side can be expanded and bounded by ignoring the conditional in the belief update rule and always using $\beta_\ell^{t-1}$:

$$W(x_j, x_\ell) + \beta_\ell^{t-1} \geq B_{j\ell}^{t-1}.$$

By definition $\alpha_\ell^{t-1} \leq \beta_\ell^{t-1}$, since the former is the negation of a larger value than the latter. A sufficient condition to guarantee Inequality (4) is then

$$W(x_j, x_u) + \beta_v^{t-1} \geq \max(\{W(x_j, x_\ell) + \beta_\ell^{t-1} | \ell\}),$$

where $\ell$ is in the remaining unseen indices as in (4). Since each component on the left-hand side has been explored in decreasing order, the maximization on the right can be relaxed into independent maximizations over each component, and neither can exceed the corresponding value on the left. □

Thus, the algorithm will never stop too early. However, the running time of the selection operation depends on how early the stopping criterion is detected. In the worst case, the process examines every entry of the row, with some overhead checking for repeat comparisons. McAuley and Caetano (2009, 2010) showed that for random orderings of each dimension (and no truncated cache size), the expected number of belief comparisons necessary is $O(\sqrt{N})$ to find the maximum, where, in our case $N = m + n = |V|$. We show that selection is computable with $O(\sqrt{bN})$ expected comparisons. However, for problems where the orderings of each dimension are negatively correlated, the running time can be worse. In the case of $b$-matching, the orderings of the beliefs and potentials are in fact negatively correlated, but in a weak manner. We first establish the expected performance of the sufficient selection algorithm under the assumption of randomly ordered $\beta$ values.

**Theorem 1.** *Considering the element-wise sum of two real-valued vectors $\vec{w}$ and $\vec{\beta}$ of length $N$ with independently random sort orders, the expected number of elements that must be compared to compute the selection of the b'th greatest entry $\sigma_b(\{w_i + \beta_i | i\})$ is $\sqrt{bN}$.*

*Proof.* The sufficient selection algorithm can be equivalently viewed as checking element-wise sums in the sort orders of the $\vec{w}$ and $\vec{\beta}$ vectors, and growing a set of $k$ indices that have been examined. The algorithm can stop once it has seen $b$ entries that are in the first $k$ of both sort orders.

We first consider the algorithm once it has examined $k$ indices of each vector, and derive the expected number of entries that will be in both sets of $k$ greatest entries. Since the sort orders of each set are random, the problem can be posed as a simple sampling scenario. Without loss of generality, consider the set of indices that correspond to the greatest $k$ entries in $\vec{w}$. Examining the greatest $k$ elements of $\vec{\beta}$ is then equivalent to randomly sampling $k$ indices from 1 to $N$ without replacement. Thus, the probability of any of the $k$ greatest entries of $\vec{\beta}$ being sampled is $k/N$, and, since there are $k$ of these, the expected number of sampled entries that are in the greatest $k$ entries of both vectors is $k^2/N$.

Finally, to determine the number of entries the algorithm must examine to have, in expectation, $b$ entries in the top $k$, we simply solve the equation $b = k^2/N$ for $k$, which yields that when $k = \sqrt{bN}$, the algorithm will in the expected case observe $b$ entries in the top $k$ of both lists and therefore completes computation. □

Applying the estimated running time to analysis of the full algorithm provides the following corollary.

**Corollary 1.** *Assuming the $\beta$ messages and the weight potentials are always randomly, independently ordered, and for constant b, the total running time for each iteration of belief propagation for b-matching with sufficient selection is $O(N^{1.5})$, and the total running time to solve b-matching is $O(N^{2.5})$.*

It is important to point out the differences between the assumptions in Theorem 1 and why they do not always hold in real data scenarios. When nodes represent actual objects or entities and the weights are determined by a function between nodes, the weight values have dependencies and are therefore not completely randomly ordered. Furthermore, the $\beta$ values change during belief propagation according to rules that depend on the weights, and in some cases can cause the selection time to grow to $O(N)$. Nevertheless, in many sampling settings and real data generating processes, the weights are random enough and the messages behave well enough that the algorithm yields significant speed improvements. Section 3 contains synthetic and real data experiments that demonstrate the significant speed improvement as well as a contrived, synthetic experiment where the speedup is less significant due to a special sampling process.

Finally, the space requirement for this algorithm has been reduced from the $O(N^2)$ beliefs (or messages) of the previous belief propagation algorithm to $O(N)$ storage for the $\alpha$ and $\beta$ values of each row. Naturally, this improvement is most significant in settings where the weights are computable from an efficient function, whereas if the weights are arbitrary, the input itself requires $O(N^2)$ memory, so the memory reduction only allows the additional storage to be linear. In most machine learning applications, however, the weights are computed from functions of node descriptor pairs, such as Euclidean distance between vectors or kernel values. In these applications, the algorithm needs only to store the node descriptors, the $\alpha$ and $\beta$ values and, during the computation of Algorithm 2, $O(N)$ beliefs (which can be immediately deleted before computing the next row). The weight cache adds $O(cN)$ space, where we consider $c$ a user-selected constant.

The space reduction is also significant for the purposes of parallelization. The computation of belief propagation is easy to parallelize, but the communication costs between processors can be prohibitive. With the proposed algorithm, each computer in a cluster stores only a copy of the node descriptors and the current $\alpha$ and $\beta$ values. At each iteration, the cluster must share the $2N$ updated $\alpha$ and $\beta$ values. This is in contrast to previous formulations where $O(N^2)$ messages or beliefs needed to be transmitted between computers at each iteration for full parallelization. Thus, when it is possible to provide each computer with a copy of the node descriptor data, an easy parallelization scheme is to split the row updates between cluster computers at each iteration.

## 3 EXPERIMENTS

This section describes empirical results from synthetic tests, which provide useful insight into the behavior of the algorithm, and a simple test on the MNIST handwritten digits data set, which demonstrates that the performance improvements apply to real data.

### 3.1 Synthetic Gaussian Data

In these experiments, the running time of the proposed algorithm is measured and compared against two baseline methods: the standard belief propagation algorithm, which is equivalent to setting the proposed algorithm's cache size to zero, and the Blossom V code by Kolmogorov (2009), which is considered to be a state-of-the-art maximum weight non-bipartite matching solver.

For both experiments, node descriptors are sampled from zero-mean, spherical Gaussian distributions with
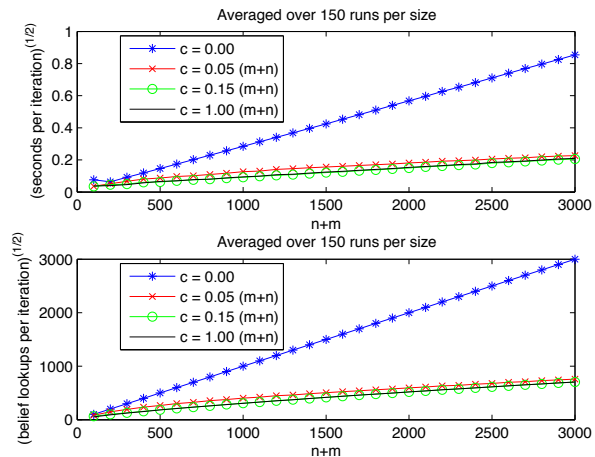


Figure 1: Running Time Measurements on Synthetic Gaussian Data. Top: Square root CPU time per iteration used to solve *b*-matching of varying sizes. The default belief propagation algorithm is equivalent to cache size $c = 0$, where the running time appears to grow quadratically. Nonzero cache sizes are clearly sub-quadratic (sub-linear in the square root plot). Bottom: Count of belief lookups per iteration. The number of belief lookups serves as a surrogate measure of running time which is not affected by other processes running on the computer.

variance 1.0, the weight function returns negative Euclidean distance, and we sample bipartitions of equal size ($m = n = N/2$). In the first experiment, points are sampled from $\mathbb{R}^{20}$. Using different cache sizes, the running time of the algorithm is measured for varying point set sizes from 10 to 500. We set $b_i = 1, \forall i$. We measure the running time using actual CPU time as well as a count of belief lookups. The square roots of per-iteration running times are drawn in Figure 1. It is clear that for a cache size of zero, where the algorithm is default belief propagation, the running time per iteration scales quadratically and that for non-zero cache sizes, the running time scales sub-quadratically. This implies that, at least for random, *iid*, Gaussian data and Euclidean weights, the weights and $\beta$ values are uncorrelated enough to achieve the random permutation case speedup.

For the second experiment, node descriptors are drawn from $\mathbb{R}^5$, and we compare 1-matching performance between sufficient selection belief propagation, full belief propagation and Kolmogorov's Blossom V code. For sufficient selection, we set the cache size to $c = 2\sqrt{m+n}$. In this case, there is no equivalent notion of per-iteration time for Blossom V, so we compare the full solution time. Full belief propagation and Blossom V seem to scale similarly, but sufficient se-
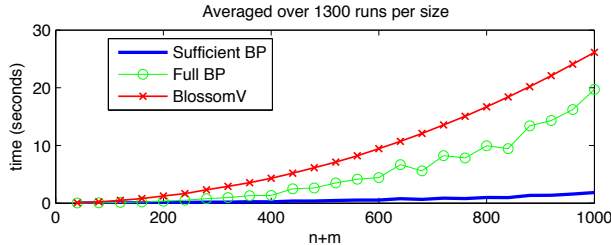
Figure 2: Comparison against Blossom V. Running times for solving varying sized bipartite 1-matching problems using Kolmogorov's Blossom V code, full belief propagation and sufficient selection belief propagation. Node descriptors are sampled from a spherical Gaussian in $\mathbb{R}^5$ and weights are negative Euclidean distances. Full belief propagation tends to run faster than Blossom V, but not always. Belief propagation with sufficient selection is significantly faster for these random problems.

lection improves the running time significantly. For this comparison, it is important to note some differences between the problem classes that the compared code solve: the algorithm behind Blossom V solves non-bipartite 1-matchings, whereas the proposed algorithm is specialized for bipartite $b$-matchings. Nevertheless, in this comparison, all algorithms are given bipartite 1-matchings. These tests were run on a personal computer with an 8-core 3 GHz Intel Xeon processor (though each run was single-threaded).

## 3.2 Synthetic Adversarial Example

In this section, we present an experiment that is an adversarial example for the sufficient selection algorithm. We construct an *iid* sampling scheme that generates data where the cached nearest neighbors of certain points will not be the $b$-matched neighbors until we cache $\Omega(N)$ neighbors. The data is generated by randomly sampling points uniformly from the surfaces of two hyperspheres in high dimensional space $\mathbb{R}^{500}$, one with radius 1.0 and the other with radius 0.1. The result is that, due to concentration, the points on the outer hypersphere are closer to all points on the inner sphere than any other points on the outer sphere, with high probability. Yet, the minimum distance $b$-matching will connect points according to which sphere they were sampled from. The distance between outer points to inner points will be in the range $[0.9, 1.1]$, and the distance between outer points to other outer points will concentrate around $\sqrt{2}$ when dimensionality is much larger than $N$ (because each vector is orthogonal with high probability). All outer points will rank the inner points as their nearest neighbors before any other outer points, but due to $b$-matching

constraints, not enough edges are available from the inner points. This is an example where, for belief propagation to find the best $b$-matching, the $\alpha$ and $\beta$ values must be negatively correlated with the weights.

Using cache sizes from 0 to $m + n$, where $c = m + n$ allows the full sufficient selection, running times are compared for different sized input. From the arguments above, the sufficient selection should fail to improve upon the asymptotic time of full selection for all nodes on the outer hypersphere. Nevertheless, a constant time speedup is still achieved by exploiting order information. This may simply be because, sufficient selection speeds up performance for the points on the inner hypersphere but not for the adversarially arranged points on the outer hypersphere.
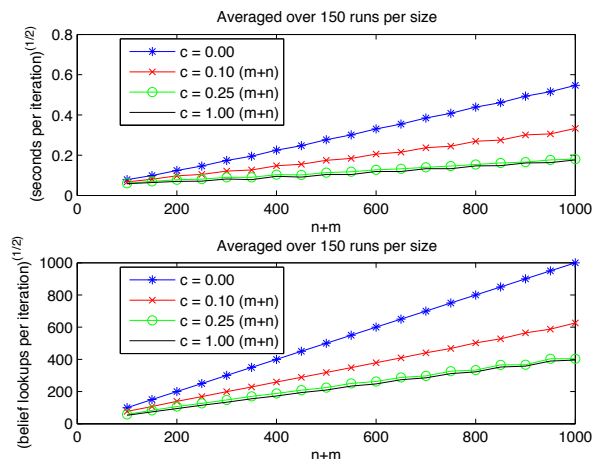


Figure 3: High Dimensional Two Hypersphere Running Times. Even for a full cache size, the running time seems to still scale quadratically, albeit with a smaller constant factor.

## 3.3 Handwritten Digits

We perform timing tests on the MNIST digits data set (LeCun et al., 2001), which contains 60k training and 10k testing handwritten digit images. The images are centered, and represented as $28 \times 28$ pixel grayscale images. We use principle components analysis (PCA) to reduce the 784 pixel dimensions of each image to the top 100 principle eigenvector projections. We use negative Euclidean distance between PCA-projected digits as edge weights, and time sufficient selection belief propagation on a subsampled data set with varying cache sizes. In particular, for this test, we sample 10% of both the training and testing sets, resulting in 6000 training and 1000 testing digits. We generate feasible $b$-matching constraints by setting the target degree $b_{tr} \in \{1, \ldots, 5\}$ for the training points and the

target degree $b_{\text{te}}$ for testing points to $b_{\text{te}} = 6b_{\text{tr}}$ (since there are six times as many training points).

Since there are 600 million candidate edges between training and testing examples, any algorithm that stores and updates beliefs or states for each edge, such as the original belief propagation algorithm described by Huang and Jebara (2007) or the Blossom V algorithm by Kolmogorov (2009) cannot be run on most computers without the use of expensive virtual memory swapping. Thus, we only compare the running times of linear memory $b$-matching belief propagation as described in Section 2.2 using different cache sizes.

These timing tests were run on a Mac Pro with an 8-core 3 GHz Intel Xeon processor, each $b$-matching job running on only a single core. The results show that for a cache size of 200, the solution time is reduced from around an hour to fewer than ten minutes. Interestingly, the running time for larger $b$ values is less, which is because belief propagation seems to converge in fewer iterations. For larger cache sizes, we achieve minimal further improvement in running time; it seems that once the cache size is large enough, the algorithm finishes selection before running out of cached weights.

Finally, using a cache size of 3500, finding the minimum distance matching for the full MNIST data set, which contains six hundred million candidate edges between training and testing examples, took approximately five hours for $b_{\text{tr}} = 1$ and $b_{\text{tr}} = 4$. The statistics from each run are summarized in Table 1. As in the synthetic examples, we count the number of belief lookups during the entire run and can compare against the total number that would have been necessary had a standard selection algorithm been used (which is $(m + n)^2$ per iteration). The running time is approximately 100 times faster than the estimated time for belief propagation with naive selection.
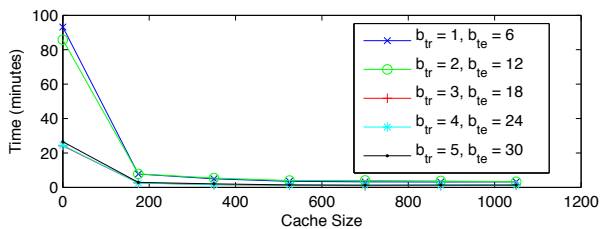


Figure 4: Minimum Euclidean Distance $b$-Matching Subsampled MNIST Digit Running Times. Weighted $b$-matching is solved on a subset of the MNIST data set. Running times are measured for various target degrees $b_{\text{tr}}$ and $b_{\text{te}}$, as well as weight cache sizes. See Table 1 for running time measurements on the full MNIST data set.

Table 1: Running Time Statistics on Full MNIST Data Set. Matching the full MNIST training set to the testing set considers 7000 nodes and 600 million edges. The table columns are, from left to right, the target degrees $b_{\text{tr}}$ and $b_{\text{te}}$ for training and testing nodes, raw running time for $b$-matching in minutes, the total number of belief lookups during the entire run, and the percentage of the belief lookups that would have been necessary using naive belief propagation (% Full).

| $b_{\text{tr}}$ | $b_{\text{te}}$ | Time (min.) | Belief Lookups | % Full |
|---|---|---|---|---|
| 1 | 6 | 285.77 | $4.5992 \times 10^{10}$ | 0.94% |
| 4 | 24 | 306.76 | $5.2208 \times 10^{10}$ | 1.11% |

## 4 DISCUSSION

This article presented an enhanced belief propagation algorithm that solves maximum weight $b$-matching. The enhancements yield significant improvements in space requirement and running time. The space requirement is reduced from quadratic to linear, and the running time is reduced from $O(N^3)$ to $O(N^{2.5})$ under certain assumptions. Empirical performance is consistent with the theoretical analysis, yet the theoretical analysis needs restrictive assumptions, so relaxing these to more realistic scenarios remains future work.

Further speed and space improvements may be possible by conceding exactness in favor of an approximation scheme. For example, node descriptors can be stored using hashing schemes that preserve the reconstruction of node distances (Karatzoglou et al., 2010). Additionally, the initial iteration requires essentially a $k$-nearest neighbor computation, for which there are various approximate methods with speed tradeoffs. Extra analysis is necessary, however, to provide the error bound for the resulting $b$-matching, as well as to ensure that belief propagation converges. Parallel versions of the proposed algorithm are yet to be implemented, and, while they seem theoretically straightforward, exactly implementing the parallelization as efficiently as possible remains future work. Finally, because of this algorithm, the class of $b$-matching problems efficiently solvable is now much larger, so application of $b$-matching (and the algorithms that build on $b$-matching) to larger scale data is a significant direction of future research.

### Acknowledgements

# References

M. Bayati, D. Shah, and M. Sharma. Maximum weight matching via max-product belief propagation. In *Proc. of the IEEE International Symposium on Information Theory*, 2005.

M. Bayati, C. Borgs, J. T. Chayes, and R. Zecchina. Belief-propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. *CoRR*, abs/0709.1190, 2007.

T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms*. McGraw-Hill Book Company, Cambridge, London, 2 edition, 2001.

A. Danyluk, L. Bottou, and M. Littman, editors. *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, 2009. ACM. ISBN 978-1-60558-516-1.

R. Duan and S. Pettie. Approximating maximum weight matching in near-linear time. In *Proceedings 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

C. Fremuth-Paeger and D. Jungnickel. Balanced network flows. i. a unifying framework for design and analysis of matching algorithms. *Networks*, 33(1), 1999.

H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

B. Huang and T. Jebara. Loopy belief propagation for bipartite maximum weight b-matching. In M. Meila and X. Shen, editors, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, volume 2 of JMLR: W&CP, March 2007.

B. Huang and T. Jebara. Exact graph structure estimation with degree priors. In M. Wani, M. Kantardzic, V. Palade, L. Kurgan, and Y. Qi, editors, *ICMLA*, pages 111–118. IEEE Computer Society, 2009. ISBN 978-0-7695-3926-3.

T. Jebara and V. Shchogolev. B-matching for spectral clustering. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 679–686. Springer, 2006. ISBN 3-540-45375-X.

T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. In Danyluk et al. (2009), page 56. ISBN 978-1-60558-516-1.

A. Karatzoglou, A. Smola, and M. Weimer. Collaborative filtering on a budget. In Y. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, pages 389–396, 2010.

V. Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009. ISSN 1867-2949. URL http://dx.doi.org/10.1007/s12532-009-0002-8. 10.1007/s12532-009-0002-8.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

J. McAuley and T. Caetano. Faster algorithms for max-product message-passing. *CoRR*, abs/0910.3301, 2009.

J. McAuley and T. Caetano. Exploiting data-independence for fast belief-propagation. In J. Fürnkranz and T. Joachims, editors, *ICML*, pages 767–774. Omnipress, 2010.

J. Salez and D. Shah. Optimality of belief propagation for random assignment problem. In C. Mathieu, editor, *SODA*, pages 187–196. SIAM, 2009.

S. Sanghavi, D. Malioutov, and A. Willsky. Linear programming analysis of loopy belief propagation for weighted matching. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1273–1280, Cambridge, MA, 2007. MIT Press.

P. Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theor. Comput. Sci.*, 410(44): 4480–4488, 2009.

B. Shaw and T. Jebara. Minimum volume embedding. In M. Meila and X. Shen, editors, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, volume 2 of JMLR: W&CP, March 2007.

B. Shaw and T. Jebara. Structure preserving embedding. In Danyluk et al. (2009), page 118. ISBN 978-1-60558-516-1.