Loopy Belief Propagation for Bipartite Maximum Weight b-Matching

Bert Huang
Computer Science Dept.
Columbia University
New York, NY 10027

Tony Jebara
Computer Science Dept.
Columbia University
New York, NY 10027

Abstract

We formulate the weighted b-matching objective function as a probability distribution function and prove that belief propagation (BP) on its graphical model converges to the optimum. Standard BP on our graphical model cannot be computed in polynomial time, but we introduce an algebraic method to circumvent the combinatorial message updates. Empirically, the resulting algorithm is on average faster than popular combinatorial implementations, while still scaling at the same asymptotic rate of $O(bn^3)$. Furthermore, the algorithm shows promising performance in machine learning applications.

1 INTRODUCTION

Belief propagation (BP) is a method for efficiently performing probabilistic inference on most graphical models. However, its convergence is often unreliable if these graphical models contain loops. In a few interesting cases, convergence proofs have been made for certain graphical models. This paper identifies another situation where BP on a loopy graphical model can provably and efficiently solve a combinatorial optimization known as b-matching.

Classical applications of b-matching include resource allocation problems. Variations on b-matching are also useful for VLSI applications [6]. In particular, bipartite b-matching applies when one has n suppliers and n customers, and needs to ship b supplies between customers and suppliers. However, there are varying costs for assigning different suppliers to customers. Other b-matching applications include providing referrals on social networks or matching bidders to sellers in auction situations.

In addition to its classical applications, recent work has demonstrated weighted b-matching's utility in machine learning [4]. The b-matching algorithm can be used in place of k-nearest neighbors where it permits a given datum to select k neighboring points but also ensures that exactly k points will select the given datum as their neighbor. Furthermore, [4] shows how b-matching can be used as a preprocessing step for improving spectral clustering by pruning the weighted affinity graph and removing noisy edges.

We cast maximum weighted b-matching as a graphical model because we are interested in the usefulness of BP for solving combinatorial problems in general. From a practical sense, BP inherently allows for parallel computation and we experienced a significant constant speedup when comparing the 1-matching algorithm in [1] to classical 1-matching algorithms. Belief propagation also can be stopped early for an approximate solution in a real-time environment.

This paper is organized as follows. Section 2 discusses BP and prior efforts to characterize convergence on some graphical models. Section 3 sets up the b-matching optimization as BP on a loopy graphical model. Section 4 proves that loopy BP will converge and gives a bound on the number of iterations. Section 5 presents experiments with the loopy BP method that show improved computational efficiency over a popular combinatorial algorithm and improved accuracy over k-nearest neighbors in a classification setting. We conclude and outline future directions in Section 6.

2 BELIEF PROPAGATION

This work generalizes and extends previous work on BP to solve maximum weight matching (which is equivalent to b-matching when $b = 1$) [1]. It uses a similar approach to that of [3], where we describe a probability distribution for a combinatorial optimization on which standard BP is impractical, and we compute the belief propagation in closed form, producing an efficient algorithm.

Graphical models are powerful probabilistic constructions that describe the dependencies between different dimensions in a probability distribution. In an acyclic graph, a graphical model can be correctly maximized or marginalized by collecting messages from all leaf nodes at some root, then distributing messages back toward the leaf nodes [7].

However, when loops are present in the graph, belief propagation methods often reach oscillation states or converge to either local maxima or incorrect marginals. Cases with a single loop have been analyzed in [10], which gives an analytical expression relating the steady-state beliefs to the true marginals.

Previous work related convergence of BP to types of free energy in [8], and [5] describes general sufficient conditions for convergence towards marginals.

Belief propagation has been generalized into a larger set of algorithms called tree-based reparameterization (TRP) in [9]. These algorithms iteratively reparameterize the distribution without changing it based on various trees in the original graph. Pairwise BP can be interpreted as doing this on the two-node trees of each edge. The set of graphs on which TRP converges subsumes that of BP. However, we use standard belief propagation here because it converges on our graph, it is simpler to implement and has additional benefits such as parallel computation.

In this work, we provide a proof of convergence based on our specific graphical model and use the topology of our graph to find our convergence time.

3 THE b-MATCHING GRAPHICAL MODEL

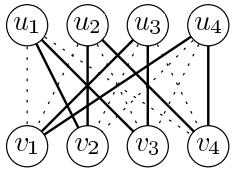


Figure 1: Example b-matching M_G on a bipartite graph G. Dashed lines represent possible edges, solid lines represent b-matched edges. In this case $b = 2$.

Consider a bipartite graph $G = (U, V, E)$ such that $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, and $E = (u_i, v_j), \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, n\}$. Let A be the weight matrix of G such that the weight of edge (u_i, v_j) is A_{ij}. Let a b-matching be characterized by a function $M(u_i)$ or $M(v_j)$ that returns the set of neighbor vertices of the input vertex in the b-matching. The b-matching objective function can then be written as

$$\max_M \ \mathcal{W}(M) = \max_M \sum_{i=1}^{n} \sum_{v_k \in M(u_i)} A_{ik} + \sum_{j=1}^{n} \sum_{u_\ell \in M(v_j)} A_{\ell j}$$
$$s.t. \quad \begin{aligned} |M(u_i)| &= b, \forall i \in \{1, \ldots, n\} \\ |M(v_j)| &= b, \forall j \in \{1, \ldots, n\} \end{aligned} \quad . \tag{1}$$

If we define variables $x_i \in X$ and $y_j \in Y$ for each vertex such that $x_i = M(u_i)$, and $y_j = M(v_j)$, we can define the following functions:

$$\phi(x_i) = \exp(\sum_{v_j \in x_i} A_{ij}), \qquad \phi(y_j) = \exp(\sum_{u_i \in y_j} A_{ij})$$
$$\psi(x_i, y_j) = \neg(v_j \in x_i \oplus u_i \in y_j). \tag{2}$$

Note that both X and Y have values over $\binom{n}{b}$ configurations. For example, for $n = 4$ and $b = 2$, x_i could be any entry from $\{\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\}$. Similarly, if we were to crudely view the potential functions $\phi(x_i)$, $\phi(y_j)$ and $\psi(x_i, y_j)$ as tables, these tables would be of size $\binom{n}{b}$, $\binom{n}{b}$ and $\binom{n}{b}^2$ entries respectively. The potential function $\psi(x_i, y_j)$ is a binary function which goes to zero if y_j contains a configuration that chooses node u_i when x_i does not contain a configuration that includes node v_j and vice-versa. These are zero configurations since they create an invalid b-matching. Otherwise $\psi(x_i, y_j) = 1$ if x_i and y_j are in configurations that could agree with a feasible b-matching. In Section 3.2 we will show how to avoid ever directly manipulating these cumbersome tables explicitly.

Using the potentials and pairwise clique functions, we can write out the weighted b-matching objective as a probability distribution $p(X, Y) \propto \exp(\mathcal{W}(M))$ [1].

$$p(X, Y) = \frac{1}{Z} \prod_{i=1}^{n} \prod_{j=1}^{n} \psi(x_i, y_j) \prod_{k=1}^{n} \phi(x_k)\phi(y_k) \tag{3}$$

3.1 THE MAX-PRODUCT ALGORITHM

We maximize this probability function using the max-product algorithm. The max-product algorithm iteratively passes messages, which are vectors over settings of the variables, between dependent variables and stores beliefs, which are estimates of max-marginals. The following are the update equations for messages from x_i to y_j. To avoid clutter, we omit the formulas for messages from y_j to x_i because these update equations for the reverse messages are the same except we swap the x and the y terms. In general, the default range of subscript indices is 1 through n; we only indicate the exceptions.

$$m_{x_i}(y_j) = \frac{1}{Z}\max_{x_i}\left[\phi(x_i)\psi(x_i,y_j)\prod_{k\neq j} m_{y_k}(x_i)\right]$$
$$b(x_i) = \frac{1}{Z}\phi(x_i)\prod_{k} m_{y_k}(x_i)$$

Loopy belief propagation on this graph converges to the optimum. However, since there are $\binom{n}{b}$ possible settings for each variable, direct belief propagation is not feasible with larger graphs.

3.2 EFFICIENT BELIEF PROPAGATION OF $\binom{n}{b}$-LENGTH MESSAGE VECTORS

We exploit three peculiarities of the above formulation to fully represent the $\binom{n}{b}$ length messages as scalars.

First, the ψ functions are well structured, and their structure causes the maximization term in the message updates to always be one of two values.

$$m_{x_i}(y_j) \propto \max_{v_j\in x_i}\phi(x_i)\prod_{k\neq j} m_{y_k}(x_i), \text{ if } u_i \in y_j$$
$$m_{x_i}(y_j) \propto \max_{v_j\notin x_i}\phi(x_i)\prod_{k\neq j} m_{y_k}(x_i), \text{ if } u_i \notin y_j \quad (4)$$

This is because the ψ function changes based only on whether the setting of y_j indicates that v_j shares an edge with u_i. Furthermore, if we redefine the above message values as two scalars, we can write the messages more specifically as

$$\mu_{x_iy_j} \propto \max_{v_j\in x_i}\phi(x_i)\prod_{u_k\in x_i\setminus v_j}\mu_{ki}\prod_{u_k\notin x_i\setminus v_j}\nu_{ki}$$
$$\nu_{x_iy_j} \propto \max_{v_j\notin x_i}\phi(x_i)\prod_{u_k\in x_i\setminus v_j}\mu_{ki}\prod_{u_k\notin x_i\setminus v_j}\nu_{ki}. \quad (5)$$

Second, since the messages are unnormalized probabilities, we can divide any constant from the vectors without changing the result. We divide all entries in the message vector by $\nu_{x_iy_j}$ to get

$$\hat{\mu}_{x_iy_j} = \frac{\mu_{x_iy_j}}{\nu_{x_iy_j}} \quad \text{and} \quad \hat{\nu}_{x_iy_j} = 1 \ .$$

This lossless compression scheme simplifies the storage of message vectors from length $\binom{n}{b}$ to 1.

We rewrite the ϕ functions as a product of the exponentiated A_{ij} weights and eliminate the need to exhaustively maximize over all possible sets of size b. Inserting Equation (2) into the definition of $\hat{\mu}_{x_iy_j}$ gives

$$\begin{aligned}\hat{\mu}_{x_iy_j} &= \frac{\max_{j\in x_i}\phi(x_i)\prod_{k\in x_i\setminus j}\hat{\mu}_{ki}}{\max_{j\notin x_i}\phi(x_i)\prod_{k\in x_i\setminus j}\hat{\mu}_{ki}}\\ &= \frac{\max_{j\in x_i}\prod_{k\in x_i}\exp(A_{ik})\prod_{k\in x_i\setminus j}\hat{\mu}_{ki}}{\max_{j\notin x_i}\prod_{k\in x_i}\exp(A_{ik})\prod_{k\in x_i\setminus j}\hat{\mu}_{ki}}\\ &= \frac{\exp(A_{ij})\max_{j\in x_i}\prod_{k\in x_i\setminus j}\exp(A_{ik})\hat{\mu}_{ki}}{\max_{j\notin x_i}\prod_{k\in x_i}\exp(A_{ij})\hat{\mu}_{ki}} \quad (6)\end{aligned}$$

We cancel out common terms and are left with the simple message update rule,

$$\hat{\mu}_{x_iy_j} = \frac{\exp(A_{ij})}{\exp(A_{i\ell})\hat{\mu}_{y_\ell x_i}}.$$

Here, the index ℓ refers to the the bth greatest setting of k for the term $\exp(A_{ik})m_{yk}(x_i)$, where $k\neq j$. This compressed version of a message update can be computed in $O(bn)$ time.

We cannot efficiently reconstruct the entire belief vector but we can efficiently find its maximum.

$$\begin{aligned}\max_{x_i} b(x_i) &\propto \max_{x_i}\phi(x_i)\prod_{k\in x_i}\hat{\mu}_{y_kx_i}\\ &\propto \max_{x_i}\prod_{k\in x_i}\exp(A_{ik})\hat{\mu}_{y_kx_i} \quad (7)\end{aligned}$$

Finally, to maximize over x_i we enumerate k and greedily select the b largest values of $\exp(A_{ik})\hat{\mu}_{y_kx_i}$.

The above procedure avoids enumerating all $\binom{n}{b}$ entries in the belief vector, and instead reshapes the distribution into a b dimensional hypercube. The maximum of the hypercube is found efficiently by searching each dimension independently. Note that each dimension represents one of the b edges for node u_i.

4 PROOF OF CONVERGENCE

We begin with the assumption that M_G, the maximum weight b-matching on G, is unique. Moreover, $\mathcal{W}(M_G)$, the weight of M_G, is greater than that of any other matching by a constant ϵ.

$$\epsilon = \mathcal{W}(M_G) - \max_{M\neq M_G}\mathcal{W}(M)$$

Let T be the unwrapped graph of G from node u_i. An unwrapped graph is a tree that contains representations of all paths of length d in G originating from a single root node without any backtracking. Each node in T maps to a corresponding node in G, and each node in G maps to multiple nodes in T. Nodes and edges in T have the same local connectivity and potential functions as their corresponding nodes in G. Let r be the root of T that corresponds to u_i in the original graph

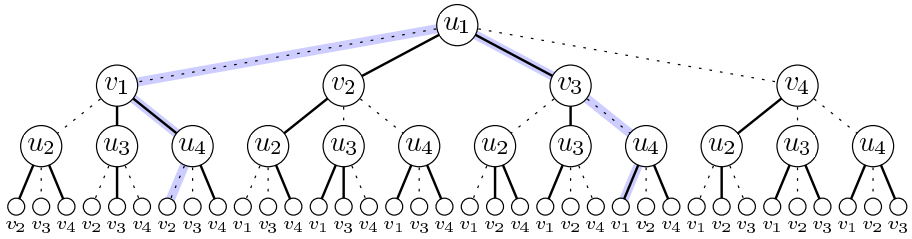


Figure 2: Example unwrapped graph T of G at 3 iterations. The matching $\tilde{M}_T$ is highlighted based on M_G from Figure 1. Note that leaf nodes cannot have perfect b-matchings, but all inner nodes and the root do. One possible path P_T is highlighted, which is discussed in Lemma 1.3.

G. The belief at iteration d of node u_i during BP on G is equivalent to the true max-marginal of r on T of depth d [10, 8]. In this case, the true max-marginal corresponds to the true maximum weight b-matching.

All maximum weight b-matchings on T are imperfect since leaf nodes are unable to complete matchings. We relax the definition of b-matching on trees to be that all non-leaf nodes are b-matched. Let M_T be the maximum weight b-matching on T. See Figure 2 for an example of a b-matching on an unwrapped graph.

Theorem 1. *Belief propagation on G converges for any node in $\Omega(n)$ iterations. In other words, maximizing the belief at the root node r of an unwrapped graph T of depth $\Omega(n)$ is equivalent to maximizing the belief of r's corresponding node u_i in the bipartite graph G. Formally, when $d = \Omega(n)$, $M_G(u_i) = M_T(r)$.*

Our proof will be structured as follows: Start with the contradiction to Theorem 1: the maximum setting of r is different from that of u_i. By modifying M_T, the optimal b-matching on T to include edges from M_G, we get a b-matching on T with higher weight. We modify M_T by finding a path in T that alternates between edges exclusively in M_T and a mapped representation of M_G. Taking the complement of this path provides the new b-matching. We bound the weight of the path by decomposing it into cycles and show the contradiction is impossible once d is large enough, which gives a bound on iterations to convergence. We next give the proof in full detail.

Proof. First we introduce a lemma on the structure of b-matchings on trees.

Lemma 1.1. *For any non-leaf vertex v in T, we refer to the subtree rooted by v as T_v. The intersection of T_v and M_T is either case* (1) *or* (2)*:*

(1) *the maximum weight b-matching of the subtree rooted at v*

(2) *the maximum weight b-matching such that v only has degree $b-1$.*

Proof. For case (1), if the edge connecting v to its parent is not in M_T, the subtree rooted at v is disjoint from the rest of the tree. Therefore, any suboptimal b-matching on v's subtree could be improved by replacing it with its optimal b-matching.

For case (2), if the edge connecting v to its parent is in M_T, v already has one incident edge outside of its subtree. Thus, when considering only its subtree, v only has $b-1$ matches available for its children.

Since the edge connecting v to its parent is either in M_T or not, these two cases cover all possibilities. □

Consider the following contradiction to Theorem 1.

Contradiction 1. *For all d, $M_G(u_i) \neq M_T(r)$*

Let $\tilde{M}_T$ be the mapped representation of M_G on T. Contradiction 1 can be rewritten as $\tilde{M}_T(r) \neq M_T(r)$.

Lemma 1.2. *For any two distinct b-matchings on T, M_1 and M_2 such that $M_1(r) \neq M_2(r)$, there is an alternating path P_T on T from a leaf to the root r then to another leaf. Here, alternating path means that for any two adjacent edges $e_i, e_j \in P_T$, if $e_i \in M_1$ then $e_j \in M_2$ and vice-versa.*

Proof. We construct P_T in two halves. For both constructions, we start at the midpoint, r, and go to two different leaf nodes. Each path proceeds down the tree by choosing alternating edges from M_1 and M_2.

The first half of P_T starts with an edge in M_1. Since $M_1(r) \neq M_2(r)$, we know there is a least one edge in $M_1(r)$ that is not in $M_2(r)$. Select one of these edges and add it to P_T. We continue from the endpoint of the added edge to the next vertex v. Add an edge to one of v's children that is in $M_2(v) \setminus M_1(v)$. Such an edge must exist because the parent edge of v was in M_1, but not in M_2, and according to Lemma 1.1, v in M_1 can only have $b-1$ matched children, and v in M_2 must have b matched children.

Therefore, v must have at least one child edge in M_2 that is not in M_1. We add one of these edges then move on to its endpoint to a new vertex. Now we add an edge that is in M_1 but not in M_2. Such an edge must exist due to the converse argument of the previous paragraph.

The second half of P_T starts with an edge in M_2. As above, we defined M_1 and M_2 so they are different at the root, so such an edge must exist. We can then apply the same arguments as above to show how to select edges down to the final leaf node.

We concatenate the two halves of P_T obtaining a path from one leaf to another. P_T alternates between edges in b-matchings M_1 and M_2. □

We apply Lemma 1.2 to b-matchings M_T and $\tilde{M}_T$ to form a path P_T that alternates between edges in $\tilde{M}_T$ and M_T. Next we will analyze the weights of edges in P_T by decomposing P_T into cycles.

Lemma 1.3. *Let P_G be the path on G that visits the corresponding nodes that P_T traverses on tree T (See Figure 4). P_G can be decomposed into a set of cycles, $C = \{c_1, \ldots, c_\tau\}$, such that $\tau \geq \frac{d-1}{n}$, and a remainder path ρ of length at most $2n$.*

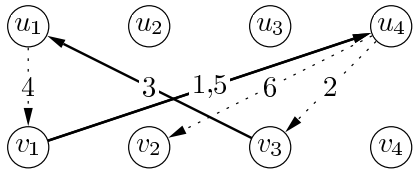


Figure 3: The cyclic alternating path P_G starting at v_1 on G that corresponds to the nodes visited by P_T. Edges are numbered to help follow the loopy path.

Proof. Start with $C = \{\}$. As we traverse P_G on G from one end to the other, once the length of P_G is greater than $2n$, the Pigeonhole Principle tells us at least one node will be visited more than once. The first time this occurs, we have a cycle.

Each time we discover a cycle c_k, we remove the cycle from P_G, add it to C, and continue on P_G. Note that removing cycles does not break the continuity of P_G because cycles begin and end on the same node. See Figure 4 for an example of this decomposition.

Since the longest possible cycle is of length $2n$ (one that visits every node in G) and P_G is of length $2d-1$, we know that τ, the number of cycles in P_G is:

$$\tau \geq \frac{2d-1}{2n} > \frac{d-1}{n}.$$

After fully pruning P_G, the remainder path, ρ must be continuous on G. Since there is no cycle in ρ, it must be of length less than $2n$. □

Consider one of these cycles, $c_k \in C$. Edges in c_k are either members of M_G or mapped edges from M_T. We are interested in the difference of the weights between these two subsets of c_k.

For each c_k, consider a modification of M_G along cycle c_k, which will produce b-matching M_{c_k}. Starting with M_G, toggle the matching of edges in c_k. Consequently, edges that were matched in M_G are free in M_{c_k} and edges that were free are matched. More importantly, the edges that were free were members of M_T. Forming M_{c_k} in this manner preserves a valid b-matching; each node in c_k loses a matched edge and also gains another thereby preserving its total degree.

Because M_{c_k} is identical to M_G except on c_k, we can bound the weight difference of the two b-matchings in c_k by comparing the weight of M_G and M_{c_k}. The difference between the weight of M_G and any other b-matching on G is at least ϵ. Multiplying this minimum difference of weight for all τ possible cycles, we have:

$$\mathcal{W}(C \cap M_G) - \mathcal{W}(C \setminus M_G) \;\geq\; \tau\epsilon \;\geq\; \frac{(d-1)\epsilon}{n} \quad (8)$$

To account for ρ, the remainder of P_G, we create another cycle c_ρ that is a modified version of ρ. Path ρ is of even length path. Without loss of generality, ρ begins with an edge from M_G and ends with an edge from M_T. To create cycle c_ρ, remove the last edge, which is from M_T, and replace it with an edge back to the first node.

This creates cycle c_ρ, and we use the same method as with the cycles in C to bound the weights.

$$\mathcal{W}(c_\rho \cap M_G) - \mathcal{W}(c_\rho \setminus M_G) \geq \epsilon$$

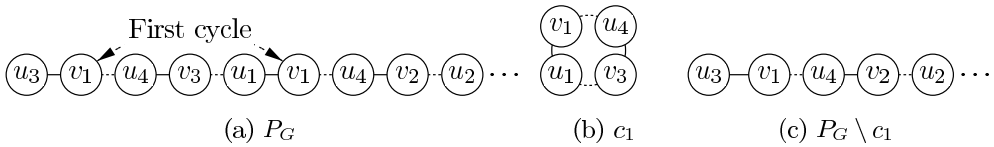
Since c_ρ replaced an edge from ρ, we add terms to the bound by pessimistically assuming the edge with least weight completed c_ρ, and the uncounted edge from ρ had the most weight of M_T.

$$\mathcal{W}(\rho \cap M_G) - \mathcal{W}(\rho \cap M_T) \geq \epsilon + \min_{e \in E} \mathcal{W}(e) - \max_{e \in M_T} \mathcal{W}(e). \quad (9)$$

We reconstruct $P_G = C \cup \rho$ and by summing Equations (8) and (9) we have the following bound:

$$\mathcal{W}(P_G \cap M_G) - \mathcal{W}(P_T \cap M_T) \geq \frac{d\epsilon}{n} + \min_{e \in E} \mathcal{W}(e) - \max_{e \in M_T} \mathcal{W}(e).$$

If the difference between these two weights is positive, there exists some b-matching on T, specifically M_T



(a) P_G (b) c_1 (c) $P_G \setminus c_1$

Figure 4: (a) One possible extension of P_G from Figure 4. This path comes from a deeper T so P_G is longer. The first cycle c_1 detected is highlighted. (b) Cycle c_1 from P_G. (c) The remainder of P_G when we remove c_1. Note the alternation of the matched edges remains consistent even when we cut out cycles in the interior of the path.

with edges in P_T toggled, that has a higher weight than M_T. This should be impossible because M_T is defined as the optimal b-matching. This occurs when

$$d \geq \frac{n}{\epsilon}\left(\max_{e_1,e_2 \in E} \mathcal{W}(e_1) - \mathcal{W}(e_2)\right) .$$

We can consider ϵ and the maximum difference between weights as constants. If $d = \Omega(n)$, Contradiction 1 always leads to this impossible circumstance, and therefore Theorem 1 is true. □

The bound in this proof is for the worst-case. In our experiments, the maximum and minimum weights had little effect and the ϵ term only changed convergence time noticeably if its value was near zero. Due to finite numerical precision, this may appear as a tie between different settings, which often causes BP to fail [11].

5 EXPERIMENTS

We elaborate the speed advantages of our method and an application in machine learning where b-matching can improve classification accuracy.

5.1 RUNNING TIME ANALYSIS

We compare the performance of our implementation of belief propagation maximum weighted b-matching against the free graph optimization package, GOBLIN. [1]

Classical b-matching algorithms such as the balanced network flow method used by the GOBLIN library run in $O(bn^3)$ time [2]. The belief propagation method takes $O(bn)$ time to compute one iteration of message updates for each of the $2n$ nodes and converges in $O(n)$ iterations. So, its overall running time is also $O(bn^3)$.

We ran both algorithms on randomly generated bipartite graphs of $10 \leq n \leq 100$ and $1 \leq b \leq n/2$. We generated the weight matrix with the `rand` function in MATLAB, which picks each weight independently from a uniform distribution between 0 and 1.

The GOBLIN library is C++ code and our implementation[2] of belief propagation b-matching is in C. Both were run on a 3.00 Ghz. Pentium 4 processor.

In general, the belief propagation runs hundreds of times faster than GOBLIN. Figure 5 shows various comparisons of their running times. The surface plots show how the algorithms scale with respect to n and b. The line plots show cross sections of these surface plots, with appropriate transformations on the running time to show the scaling (without these transformations, the belief propagation line would appear to be always zero due to the scale of the plot following the GOBLIN line). Since both algorithms have running time $O(bn^3)$, when we fix $b = 5$, we get a cubic curve. When we fix $b = n/2$, we get a quartic curve because b becomes a function of n.

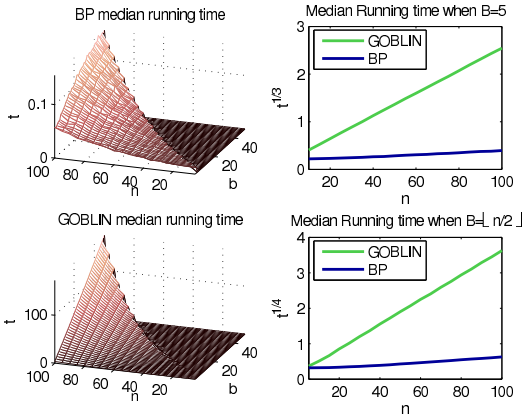


Figure 5: Median running time in seconds on randomly weighted graphs for the GOBLIN library and BP. Top Left: BP median running time with respect to n, b. Bottom Left: GOBLIN b-matching solver median running time over n, b. Top Right: Cube root of running time ($\sqrt[3]{t}$) of both algorithms when $b = 3$; note that both scale linearly in this plot, implying a n^3 term in running time for both. Bottom Right: Root-4 ($\sqrt[4]{t}$) of running time for various n, when $b = \lfloor n/2 \rfloor$. Both algorithms have quartic running time.

[1] Available at: `http://www.math.uni-augsburg.de/opt/goblin.html`

[2] Available at `http://www.cs.columbia.edu/bert/bmatching`

5.2 b-MATCHING FOR CLASSIFICATION

One natural application of b-matching is as an improvement over k-nearest neighbor (KNN) for classification. Using KNN for classification is a quick way of computing a reasonable prediction of class, but it is inherently greedy. The algorithm starts by computing an affinity matrix between all data points. For each test data point, KNN greedily selects its k neighbors regardless of the connectivity or graph this may generate. This invariably leads to some nodes serving as hub nodes and labeling too many unknown examples while other training points are never used as neighbors.

Instead, using b-matching enforces uniform connectivity across the training and testing points. For example, assume the number of testing points and training points is the same and we perform b-matching. Then, each testing point will be labeled by b training points and each training point contributes to the labeling of b testing points. This means b-matching will do a better job of recreating the distribution of classes seen in the training data when it classifies test data. This is useful when test data is transformed in some way that preserves the shape of the distribution, but it is translated or scaled to confuse KNN. This can occur in situations where training data comes from a different source than the testing data.

In both algorithms, we compute a bipartite affinity graph between a set of training points and a set of testing points. We fix the cardinalities of these sets to be equal (though if they are unequal we can downsample and run the algorithm in multiple folds to obtain classifications). We use either KNN or weighted b-matching to prune the graph, then we classify each test point based on the majority of the classes of the neighbors. In our experiments, we use negative Euclidean distance as our affinity metric.

5.2.1 Synthetic Data

We created synthetic data by sampling 50 training data points from two spherical Gaussians with means at $(3,3)$ and $(-3,-3)$. We then sampled 50 testing data points from similar Gaussians, but translated along the x-axis by $+8$. This is a severe case of translation where it is obvious that KNN could be fooled by the proximity of the testing Gaussian to the training Gaussian of the wrong class. Figure 6 shows the points we sampled from this setup. As expected, KNN could only achieve 65% accuracy on this data for the optimal setting of k, while b-matching classified the points perfectly for all settings of b.

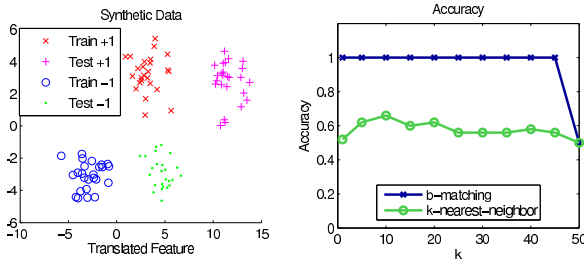


Figure 6: Left: Synthetic classification data where the test data is translated to the right along the x-axis. Right: Accuracy for k-nearest neighbor versus b-matching for various k (or b).

5.2.2 MNIST Digits With Backgrounds

We sampled the MNIST digits dataset of 28×28 grayscale digits. For each of the digits 3, 5, and 8, we sampled 100 training points from the training set and 100 from the testing set. We average accuracy over 20 random samplings to avoid anomalous results. We cross-validate over settings of b and k in the range $[1, 300]$ and save the best accuracy achieved for each algorithm on each sampling.

We examine the case where training and testing data are collected under drastically different conditions (e.g. non-IID sampling) by simulating that the testing digits are printed against various backgrounds. We replace all white (background) pixels in the testing images with these backgrounds and attempt classification using both algorithms on these new datasets. Figure 7 contains examples of digits placed in front of backgrounds.

The results show that b-matching outperforms k-nearest-neighbors, sometimes by significant margins. On the unaltered white background and the diagonal lines, both algorithms classify at slightly higher than 90% accuracy. On the grid, white noise, brushed metal, wood and marble backgrounds, b-matching is invariant to the transformation while k-nearest-neighbors suffers a drop in accuracy. Figure 7 contains a plot of both algorithm's average accuracies for each background type.

Since the background replacement is like a translation of the image vectors that preserves the general shape of the distribution, b-matching is a more useful tool for connecting training examples to testing points than k-nearest-neighbors.

6 CONCLUSION

We proved the convergence of BP for maximization on a graphical model for the weighted b-matching prob-

lem. We provided an efficient method to update the $\binom{n}{b}$-length messages. The practical running time of this algorithm is hundreds of times faster than the classical implementations we have found. This makes b-matching a more practical tool in machine learning settings and lets us tackle significantly larger datasets.

Furthermore, by formulating weighted b-matching in this manner, we can take advantage of the inherent parallel computation of belief propagation. In addition to the empirical evidence of a faster constant on the asymptotic running time, the procedure could be distributed over a number of machines. For example, n servers and n clients could run this algorithm in parallel to optimize throughput.

Theoretically, the convergence of loopy BP on this graph and the ability to update the $\binom{n}{b}$ length messages are pleasant surprises. We have tried to generalize this method to all graph structures, not just bipartite graphs, but found that there were cases that led to oscillation in the belief updates. This is due to the fact that cycles can be both even and odd in general graphs. Generalizing to such cases is one future direction for this work.

In addition to being limited to bipartite graphs, belief propagation methods have difficulty with functions that have multiple solutions. This is why we assume there is a unique solution in the proof. In a way, our convergence bound still holds when there are multiple solutions, in the sense that the ϵ value is in the denominator, so the limit as ϵ goes to zero implies infinite iterations to convergence. Belief propagation cannot handle these ties in the function due to the fact that the information being passed between variables is only the maximum value, and no information about the actual variable values is sent. Therefore, the algorithm has no mechanism to distinguish between one maximum and the other. We are currently investigating if tree-based reparameterization techniques or other graphical model inference methods can solve the multiple-optimum or non-bipartite b-matching cases.

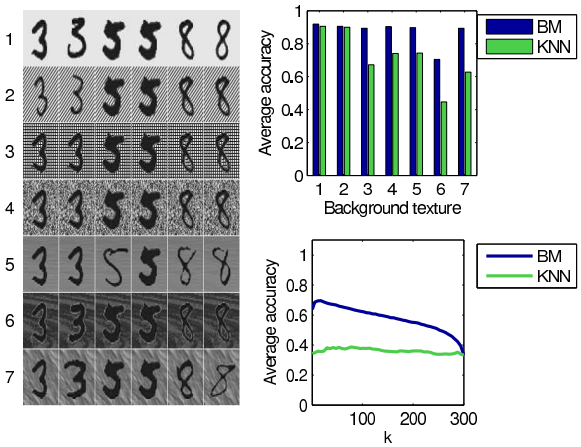


Figure 7: (Left) Examples of digits placed on backgrounds: 1-unaltered, 2-diagonal lines, 3-grid, 4-white noise, 5-brushed metal, 6-wood, 7-marble. (Top right) Average accuracies over 20 random samplings for optimal setting of b or k. (Bottom right) Average accuracy on wood background for various settings of b or k.

Acknowledgments

We thank Stuart Andrews, Andrew Howard and Blake Shaw for insightful discussions. Supported in part by NSF grant IIS-0347499.

References

[1] 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.

[2] C. Fremuth-Paeger and D. Jungnickel. Balanced network flows. i. a unifying framework for design and analysis of matching algorithms. *Networks*, 33(1):1–28, 1999.

[3] B. Frey and D. Dueck. Mixture modeling by affinity propagation. In *Advances in Neural Information Processing Systems 18*, pages 379–386. MIT Press, 2006.

[4] T. Jebara and V. Shchogolev. B-matching for spectral clustering. In *Proc. of the European Conference on Machine Learning, ECML*, 2006.

[5] J. Mooij and H. Kappen. Sufficient conditions for convergence of loopy belief propagation. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 396–40, 2005.

[6] M. Müller-Hannemann and A. Schwartz. Implementing weighted b-matching algorithms: Insights from a computational study. In *ACM Journal of Experimental Algorithmics, Volume 5, Article 8*, 2000.

[7] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[8] S. Tatikonda and M. Jordan. Loopy belief propagation and Gibbs measures. In *Proc. Uncertainty in Artificial Intell., vol. 18*, 2002.

[9] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 49(5), 2003.

[10] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.

[11] Y. Weiss and W. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEETIT: IEEE Transactions on Information Theory*, 47, 2001.