The Implementation and Application of Multiclass Belief Propagation on Hadoop

Guanyu Wang
Institute for Software Research
CMU
guanyuw@andrew.cmu.edu

Yuchen Tian
Institute for Software Research
CMU
yuchent@andrew.cmu.edu

Yu Su
Institute for Software Research
CMU
ysu1@andrew.cmu.edu

Huanchen Zhang
Institute for Software Research
CMU
huanchez@andrew.cmu.edu

December 6, 2012

1 Introduction

1.1 Belief Propagation

Based on the pairwise MRF graph, there is a message passing algorithm, called Belief Propagation[14] to compute the marginal possibility of unknown nodes.

The messages are defined as $m_{ij}(x_j)$, which is the chance that node j is in state x_j in node i's opinion.

Given node i is in state x_i, we can simply get $m_{ij}(x_j) \equiv \psi(x_i, x_j)$, where $\psi(x_i, x_j)$ is the constrain between unknown node i and unknown node j.

Then, expand x_i with the information of what other unknown nodes think node i:

$$m_{ij}(x_j) \equiv \sum_{x_i} \psi(x_i, x_j) \prod_{k \in N(x_i) \setminus j} m_{ki}(x_i)$$

Gather the internal constrains between the observation and unknown node i:

$$m_{ij}(x_j) \equiv \sum_{x_i} \phi(x_i)\psi(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{ki}(x_i)$$

This describes how we update the messages. Then the final probability of node i is in state x_i would be(k is used for normalization):

$$b_i(x_i) \equiv k\phi(x_i) \prod_{j \in N(i)} m_{ji}(x_i)$$

1.2 Fast Algorithm for Belief Background

Fast algorithm of BP approximates the BP algorithm with a matrix inversion problem[5]. It uses a linear system to approach the final solution to the Belief Propagation, call it Fast BP. Under the assumption that all probabilities are around 0.5, the approximation accuracy is quite good. The main equation for Fast BP is the following one:

$$W\mathbf{b_h} = \phi_\mathbf{h}$$

where W can be viewed as a function of the adjacency matrix of the graph, degree matrix as well as the homophily factor Then the Belief Propagation problem actually has been converted into a matrix inversion problem, computing $W^{-1}\phi_h$. One main limitation now for Fast BP is that it support only two classes.

2 Survey

Next we list the papers that each member read, along with their summary and critique.

2.1 Papers read by Yu Su

The first paper was the tutorial paper on Belief Propagation by Yedidia [14]

- *Main idea*: A lot of inference models such as Bayesian Networks, Pair-wise Markov Random Field Graph, Potts models and Factor Graphs were introduced in this paper. The author also showed how to convert different models into the Pair-wise Markov Random Field Graph. Then the Standard Belief Propagation algorithm was developed on Pair-wise MRF graph and it's intuitively exact on loop-free graphs. The author applied Bethe Approximation in free energy theory to prove that BP is exact on loop-free graphs. The observation from the Kikuchi Approximation in free energy theory led to the Generalized Belief Propagation algorithm, which is constructed on clusters of the original graph.
- *Use for our project*: This paper is a good start for those who do not know Belief Propagation before. It also introduced many models and showed how to convert them, which is helpful when we are constructing models on real data sets.
- *Shortcomings*: The regional graph method introduced to solve the accuracy problem suffers exponential computing complexity.

The second paper was a deeper discussion on BP by Yedidia [15]

- *Main idea*:
  This paper introduced Factor Graph and re-expressed Belief Propagation in this model. Then the author proved that BP is exact on loop-free graph by showing the equivalence of BP and Bethe Approximation in free energy theory. A more important contribution of this paper is that it pointed out what condition must be kept to get a valid approximation on general graphs. Several methods are introduced to get better accuracy

and more chance to converge on loopy graphs, such as region graph method, junction graph method, etc.
- *Use for our project*: This paper would be very useful if we want to do some extra work such as prove the accuracy of our new method.
- *Shortcomings*: There is still no systematic method to choose regions, which influence much on accuracy and complexity. People still have to tune the algorithm for different problems to get better answers.

The third paper was sum-product algorithm by Kschischang [6]

- *Main idea*: This paper introduced sum-product algorithm on factor graph to compute marginal functions. The prevalence of the application of factor graph and sum-product algorithm is demonstrated by generalizing forward-backward algorithm, Viterbi algorithm and Kalman Filtering into sum-product algorithm. This paper summarized many methods in applying sum-product algorithm on loopy graphs. Some of them utilized the intrinsic properties of the problem and get good result, such as by carefully scheduling message passing. Some methods translated loopy graphs into loop-free graphs, such as clustering or stretching nodes.
- *Use for our project*: Many background and examples are introduced in this paper. And since sum-product algorithm is similar to BP, this paper helps understand BP.
- *Shortcomings*: The graph translation methods might be too complex to compute that the accuracy improvements might be not reachable.

2.2 Papers read by Yuchen Tian

The first paper was the Pioneer Error Correcting Code paper by Thomas[1]

- *Main idea*: The main contribution in Thomas's paper is brought up the idea of using Error Correcting Code(ECOC) in multiclass classification. The basic idea behind this is to convert a multiclass problem into several binary classification problem by designing a N bit code for each class, then run a classifier N times to compute how likely for a given example the probability on the N_{th} bit is 1. The author then introduced a simple loss-based decoding to make decisions. In experiment, the author compared ECOC approach with One-Vs-All scheme and multiclass decision tree approach, with decision tree and neural network chosen as the underlying binary classifiers. The experiment showed that ECOC approach has better accuracy than the other two.
- *Use for our project*: It offers us a general way to solving a multiclass learning problem when we only have binary classifier(like BP in this case). It is also useful to help us get some idea how to extend the Fast BP into multiclass presentation. **More details about how this is related to BP can be found in the Methods section.**
- *Shortcomings*: Bad coding design may lead to excessive number of classifiers, thus make the computation very expensive.

The second paper was by Erin. It offers a general framework to reduce multiclass problem to binary classification problem.[7]

- *Main idea*: The main contribution in this paper is that the author presented a unifying approach of reducing multiclass classification problem to binary classification problems. By extending ECOC approach from binary to triple values and using loss-based decoding instead of hamming distance decoding, they demonstrated that One-Vs-All and All-Vs-All scheme are just special cases of ECOC approaches. This provides us a better way to understand ECOC, that it is a generalized way to solve multiclass labeling using binary classifiers.
- *Use for our project*: Unify some general and intuitive model under framework of Error Correcting Code, may simplify our assumption and design of algorithm.
- *Shortcomings*: Suffers the same problem with Thomas's paper, and loss-based decoding scheme may be expensive in Hadoop environment.

The third paper was by Ryan concerning mainly the efficiencies of various algorithms in comparing to general OVA scheme.[13]

- *Main idea*: In this paper, author argues that One-Vs-All scheme is just as accurate as other approaches and criticizes that the existing literature suffers from two major weakness: improper controlled or not well reported. If the right and well-tune classifier is chosen, then the difference between those approaches and One-Vs-All is very small. In order to hold a fair game, they chose the well tuned binary classifier such as SVM and using some statistical methods to measure whether the difference between the performance of OVA and ECOC approaches are statistically significant.
  Their conclusion comes that in performance OVA scheme has nearly identical performance as other approaches.
- *Use for our project*: A very thorough survey on comparing the efficiencies of various approaches. Very useful in helping choosing appropriate algorithm that is well-balanced on both efficiency and accuracy.
- *Shortcomings*: OVA scheme converges much slower than AVA in some datasets, while AVA scheme leads to many more classifiers to train.

2.3 Papers read by Huanchen Zhang

The first idea is brought up by Malewicz, introducing Pregel[8].

- *Main idea*: Pregel is in essence a message passing model developed by Google and inspired by Valiants Bulk Synchronous Parallel model, where vertices send messages to each other in a series of iterations called supersteps. The input to the Pregel framework is a directed graph. Within the graph, each vertex is associated with a modifiable, user defined value. The directed edges are associated with their source vertices, and each edge consists of a modifiable, user defined value and a target vertex identifier. The computations consist of a sequence of iteration called supersteps. In each superstep, the framework invokes a user defined function for each vertex. The function can read messages sent to V in superstep S-1, and send messages to other vertices that will be received at superstep S+1 and modify the state value of V and its

out going edges. Edges do not have associated computation. Vertices can deactivate themselves by voting to halt, which mean the vertex has no further work to do unless triggered externally, by a message from other vertex. The whole algorithm terminates when all vertices are inactive and there is no message in transit.
Out of Google, there is a similar open source project Apache Giraph.

- *Use for our project*: Pregel is a BSP framework on top of Hadoop for large-scale graph mining. Belief propagation is also a message passing algorithm on graph, so Pregel can be used to implement BP.
- *Shortcomings*:Bulk synchronous computation can be inefficient, because it needs to keep old and new messages, which means 2x overhead, and send redundant messages to vertices in the same node.

The second paper is written by U Kang's about PEGASUS[3].

- *Main idea*: PEGASUS is an open source Peta Graph Mining library implemented on top of Hadoop. Unlike the message passing model of Pregel, it views graph mining problems as a repeated matrix-vector multiplication and expresses a graph mining problem as a chained MapReduce. It provides a primitive called GIM-V (generalized iterated matric-vector multiplication), which is a generalization of normal matrix-vector multiplication. The usual matrix-vector multiplication is $M \times v = v'$ where $v'_i = \sum_{j=1}^{n} m_{i,j} v_j$. More specifically, GIM-V separates the usual matrix-vector multiplication with three functions combine2, combineAll, assign, which are implemented in MapReduce to achieve good parallelism.
  - combine2: multiply $m_{i,j}$ and v_j.
  - combineAll: sum n multiplication results for node i.
  - assign: overwrite previous value of v_i with new result to make v'_i.
- *Use for our project*: As discussed in the paper, by customizing these three operations, we can obtain different, useful algorithms including PageRank, Random Walk with Restart, connected components, and diameter estimation. Belief propagation algorithm can also be implemented using GIM-V as discussed in next paper.
- *Shortcomings*: With PEGASUS, graph mining algorithms are written as a series of MapReduce jobs, which requires passing the entire state of the graph from one state to the next. This can cost much communication and serialization overhead.

The third paper is U Kang's Paper about the implementation of BP on Hadoop[4].

- *Main idea*: In this paper, belief propagation is also formulated as a variant of GIM-V. First, the original undirected graph G is converted to a directed line graph $L(G)$. The directed line graph is a graph such that each node in $L(G)$ represents an edge in G, and there is an edge from v_i to v_j of $L(G)$ if the corresponding edges e_i and e_j form a length-two directed path from e_i to e_j in G. Then the belief propagation is formulated in GIM-V as

$$m(s)^{next} = A' \times_G m^{cur} \tag{1}$$

where A' is the adjacency matrix of $L(G)$.
Experiment shows when analyzing large scale graph which cannot fit in memory, belief propagation on Hadoop is the only solution. For medium-to-large graph whose nodes fit in memory but edges do not fit in memory, belief propagation on hadoop can also run faster than single machine BP. So, this method can be very useful if the graph is very large.

- *Use for our project*: This Hadoop-BP implementation can be used to analyze some interesting problems on our data sets.
- *Shortcomings*: Same potential problem as previous one. In the MapReduce computation framework, only the file systems (HDFS) can be used for communication and serialization. So, this chained MapReduce jobs can cost much communication and serialization overhead, because it needs multiple iterations and each iteration needs disk writes and reads.

2.4 Papers read by Guanyu Wang

The first paper was Fast approximation algorithm for BP by by Koutra, et al. [5]

- *Main idea*: They use a linear system to approximate the final solution to the Belief Propagation. There are mainly two key ideas for the correctness of the approximation: (1) Using the *odds ratio* instead of the original probabilities in all computations. (2) Using the Maclaurin expansion to linearize the consecutive product and taking the first-order approximation. Under the assumption that all probabilities are not far from a half, the approximation accuracy is quite good.

  Mainly, the **FaBP** solves the linear system

$$\mathbf{b_h} = [\mathbf{I} + \alpha\mathbf{D} - c'\mathbf{A}]^{-1}\phi_\mathbf{h} \tag{2}$$

  to approximate the Belief Propagation with n nodes, where $\mathbf{b_h}$ (what we want to achieve) is the "about-half" approximated odds ratio of final probabilities for two classes, $\mathbf{I}$ is the $n \times n$ identity matrix, $\mathbf{D}$ is the $n \times n$ diagonal matrix of degrees, $\mathbf{A}$ is the $n \times n$ symmetric adjacency matrix, and $\phi_\mathbf{h}$ is the "about-half" approximated odds ratio of prior probabilities.

  The most important approximation technique used in the approximation is that the original sum-product message updating rule and belief updating rule can be written as continued product with just odd-ratio variables.

$$m_r(i,j) \leftarrow B[h_r, b_r(i)/m_r(j,i)] \tag{3}$$

$$b_r(i) \leftarrow \phi_r(i)\Pi_{j\in N(i)} m_r(j,i) \tag{4}$$

  where the $< v >$ are odd-ratio for different variables, i.e. b, m, h, ϕ. $B(a,b)$ is the blending function $B(a,b) = \frac{ab+1}{a+b}$. Then using the "about-half" odds ratio to replace the common one:

$$v_r = \frac{v}{1-v} = \frac{1/2 + v_h}{1/2 - v_h} \approx 1 + 4v_h \tag{5}$$

Also, since all the updating equations are continued product, then using the Taylor expansion for logarithm after taking the logarithm on both side of Eq. (3) and Eq. (4). This finally provides us the linear update rules, which leads to the final linear system (2).

- *Why useful*: **FaBP** provides a brand-new idea that "compress" the information for any node into one variable, and using the first order approximation to simplify the computation while keeping accuracy. To know more about this point clear, refer [5] for details. The approximation technique and analysis for deriving this equation can become the fundament of our project.
- *Shortcomings*: There is an inherent property inside the **FaBP** ruins its direct extension to multi-classes problems: all the computations depends on the *odds ratio*. Obviously, $\mathbf{b_h}$ and $\phi_\mathbf{h}$ can maintain their definitions when there are only two types of probabilities. Another difficulty for the extension arises from the α and c in Eq.(2). They all depend on a homophily factor, which depicts the similarity between two connected nodes. If there are more than two classes, how to describe the homophily (or heterophily) is not clear.

The second paper was Decision Directed Acyclic Graph (**DDAG**) algorithm by by Platt et al. [11]

- *Main idea*: Based on the idea to convert the two-class classifiers into a multi-class classifier, they provides a framework called Decision Directed Acyclic Graph (**DDAG**), whose nodes contains just binary classifier, to do multiple classification. By combing many two-class classifiers with a directed acyclic structure, these classifiers can work with order as multiple-classifier.

  More precisely, the main result in this paper is the algorithm called Directed Acyclic Graph SVM (DAGSVM), which combines the results of many 1-v-1 SVMs. So constructing a Rooted Binary DAG first, in which all the nodes evaluate a binary function, which is actually one SVM. The node is then exited via the left edge, if the binary function is zero; or the right edge, if the output SVM result is one. This process is repeated on all the following children. Any time one SVM is activated, the algorithm can get a negative conclusion about one class, i.e. this input is not in this class. It is easy to see that the algorithm can reject more classes along it moves into deep layer of the constructed DAG and finally achieve the real class. According to their analysis and experiment. The DAGSVM algorithm is superior to other multiclass SVM algorithms in both training and evaluation time.

  This work brings great connections among one-versus-one classification, one-versus-rest classification and the multiple classification problems.
- *Why useful* **FaBP** can be viewed as a good two-classifier, this paper provides a clever way to use **FaBP** as elementary unit to construct more powerful classifiers: in the multiple classes case (assume there are n different classes), for any node, we can always choose any two classes and test which one this node prefer. After comparing all $\binom{n}{2}$ pairs of classes, we will have a complete partial order.

- *Shortcoming* There are mainly two weak points: (1) The **DDAG** algorithm needs a acyclic graph, however if we want to use similar ideas on Belief Propagation, there is a chance that we finally get a circle, then the original idea fails. (2) The number of binary classifiers needed to construct the multiple classes one is $N(N-1)/2$ for N classes. This quadratically increasing number may hurt the real implementation (e.g. on Hadoop).

The third paper was walk-sum interpretation of Gaussian **BP** by Malioutov et al. [9]

- *Main idea*: By decomposing the correlation between each pair of variables as a sum over all walks between those variables in the graph, they provide a walk-sum interpretation for the Gaussian Belief Propagation as well as loopy Belief Propagation.

  This work develops a "walk-sum" formulation for computation of means, variances and correlations as sums over certain sets of weighted walks in a graph. It first provides a description of the walk summability: a walk of $l > 0$ in a graph G is a sequence of nodes $\{w_0, w_1, \cdots, w_l\}$, $w_i \in V$ and any two consecutive nodes (w_k, w_{k+1}) represents an edge in this graph, i.e. $(w_k, w_{k+1}) \in E$. Define the weight of a walk to be the product of edge weights along the walk. There is a connection between the Gaussian inference and the walk. The covariance can be decomposed as a sum of infinite number of new matrices, and their new matrices' elements can be computed as a sum of walks' weight. If the sum over all walks define on a Gaussian distribution is well defined (converge to the same value for all possible summation order, etc.), then this distribution is walk-summable. The important part is that in walk-summable models, means and variances correspond to walk-sums over certain sets of walks, and the exact walk-sums over infinite sets of walks for means and variances can be computed efficiently in a recursive fashion. Finally they show that these walk-sum computations map exactly to belief propagation updates.
- *Why useful* It provides a new direction: using other expressions/understanding for Belief Propagation. We already know that **BP** can be presented as an energy minimizing problem (refer [2]). The energy function can be written as a format like $\min f(x) = g(x) + h(x)$ with differentiable part $g(x)$ and non-differentiable part $h(x)$. Usually the generalized descent gradient method can be used to solve it. At the same time, this function can hopefully be approximated by the second-order methods. Also, the walk-sum interpretation may also be written as an optimization problem with adjacency matrix and degree matrix, etc.
- *Shortcomings* This paper just discusses the Gaussian and Loopy Belief Propagation. Will similar analysis work for general **BP** is still obscure.

3 Proposed Method

3.1 Convergence Detection

The graph is loopy in most scenarios, which might cause Belief Propagation could not converge. Although Yedidia had proposed a theory proving when Belief Propagation would

converge and get acceptable results, its more practical to just run Belief Propagation and wait for it getting converged.

When running Belief Propagation, we can simply set the number of iterations after which BP terminates. And this is what PEGASUSs implementation of BP currently using. The algorithm terminates after fixed number of iterations, without knowing whether it converged or not.

We found that its a bit difficult for us to tell whether we got the stable answer from BP when we evaluate BP on different datasets. The result would vary if we adjusted the number of iterations. So, we think convergence detection would be very helpful for users running BP on their datasets. Its even essential for a BP implementation to produce the right answer.

Some discussions on convergence had been proposed by Mooij[10], they detect convergence by checking the difference between all the new messages and old messages. We followed their work and implemented convergence detection in the same way. Every iteration, we check the difference on every pair of new and old messages and the algorithm stops when all the differences are below threshold.

With above convergence detection mechanism, we can monitor the behavior of algorithm and tune the parameters and algorithm accordingly.

3.2 Message Smoothing

As discussed above, BP may not converge in some situations. If there are a lot of loops in the network, and even worse, many messages conflicts with their complement message from the other direction, there will be oscillatory behaviors during the message updating process and make BP never converge. To tackle this problem, Pretti[12] found a method named message smoothing. We implemented this algorithm on BP.

It reduces oscillatory behaviors in the message updating process by smoothing, or damping, messages between two iterations. In the original belief propagation, the messages which node i will send to other nodes are a function of messages sent to node i, which may cause oscillations in loopy networks. We can avoid changing messages too drastically by having a weighted average of the new message and the old message as follows. This can dampen oscillations and increase the chances that it converges.

Denoting $\bar{m}^t_{i,j}(x_j)$ as the updated message in iteration t obtained by message passing, λ as the message smoothing factor, the new message from node i to node j in iteration t, denoted by $m^t_{i,j}(x_j)$ is computed as weighted average of the old message and the updated message as

$$m^t_{i,j}(x_j) = (1-\lambda)m^{t-1}_{i,j}(x_j) + \lambda\bar{m}^t_{i,j}(x_j) \tag{6}$$

3.3 Methods for Extending Fast BP to Multiclass Problems

3.3.1 Error Correcting Code Methods

Motivation

Fast BP is now limited to solve problems with only 2 labels. Formally extending Fast BP to multiclass can be difficult. Instead of formally formulating the multiclass Fast BP, we emphasize BP as a tool to give the correct label rather than compute the probability. Thus we can divide a single multiclass problem into several binary problems so that we can use Fast BP to give the label.

General Definition

To solve a multiclass problem, which has k classes and l training example $(x_1, y_1)...(x_l, y_l)$, we can use error correcting code approach, which usually involves the following steps:

Code Design

In this step, for each class in the k classes, we design a unique N bits code. For example, for a problem recognizing hand writings of 10 digits, we can define the following 6 bits error codes, shown in Figure 1.

Class	Code Word					
	vl	hl	dl	cc	ol	or
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	0	1	0	0
9	0	0	1	1	0	0

Figure 1: Coding scheme for 10 digits recognizing problem

The meaning of those codes follows the description in the figure 2 from [1].
There are basically two principles for design a code[1]:

1. **Row Distance**: The hamming distance between each two classes' codes should be as large as possible to make classes more distinguishable from each other.

2. **Column Distance**: Each bit classifier should be independent from each other.

It is obvious that there is a trade-off between those two principles, which leads to different code design methods. We will adapt the one described in Thomas's Paper[1], which is every effective to generate compact yet distinguish code.

Column position	Abbreviation	Meaning
1	vl	contains vertical line
2	hl	contains horizontal line
3	dl	contains diagonal line
4	cc	contains closed curve
5	ol	contains curve open to left
6	or	contains curve open to right

Figure 2: Meaning of each bit

Running Fast BP for Bits

As we already described in the previous sections, after designing of code, we run Fast BP for each bits, note as $f_1...f_n$. We can interpret $f_i(x)$ as the probability that the number appear on the i^{th} bit of the code for a given example x is 1.

Deciding Label

For a node r, after computation, we get a vector of predictions of those functions:

$$f(r) = (f_1(r)...f_n(r))$$

In order to make final prediction we should find the class which is *closest* to $f(r)$.
In general, there are different approaches to achieve this:

1. **Hamming Decoding**: Transform $f(r)$ from real-value to binary value, and calculate the Hamming distance with code of each class. The label that has the shortest hamming distance is chosen.

2. **Loss-based Decoding**: This method is introduced in [7], which defines a loss function between probability made by f and the target bit of class, without transform $f(r)$ into binaries.

Practical Variations of ECOC

Error correcting code suffers from one major disadvantage, that is, if the number of classes is large, the number of bits will grow excessively, even exponentially. Sometimes, if the classification algorithm itself is time-consuming, such as Fast BP, even the number of bits is reasonable, ECOC becomes very expensive.

So we introduce a practical variation of ECOC, called One-VS-All schema. In [7], One-VS-All is proved to be a special case of ECOC.

It follows a very simple strategy: for every class, we train a binary classifier to distinguish whether a node is a class or not. The decision of final label is simple too, we just pick the class that has the highest YES probability. One-VS-All will significantly reduce the number

of bits involved, especially when k is large. And as [13] suggests, when the underlying binary classifier performs well, One-VS-All works as well as ECOC approach.

Actually, the two schemes are just special case of ECOC, but are much more straight forward.

3.4 FastBP for Multiple Classes

The original Belief propagation update rules are as follows:

$$b_i(x_k) = \eta \cdot \phi_i(x_k) \cdot \prod_{j \in N(i)} m_{ji}(x_k) \tag{7}$$

$$m_{ij}(x_g) = \sum_{x_k} \phi_i(x_g) \cdot \psi_{ij}(x_k, x_g) \cdot \prod_{n \in N(i) \setminus j} m_{ni}(x_k) \tag{8}$$

where x_k is one state for any node and $b_i(x_k)$ is the probability (belief) for node i staying at state x_k. So the first step of approximating the original rules is to remove the product operators. A very intuitive and common method is taking logarithms of both sides. For Eq.(7), it becomes

$$\log(b_i(x_k)) = \log(\eta) + \log(\phi_i(x_k)) + \sum_{j \in N(i)} \log(m_{ij}(x_k)) \tag{9}$$

Like the work [5] by Koutra, et al., we have to use a smart approximation for the logarithm. But let us skip this part for a moment. Assume we have a reasonable approximation of logarithms now, i.e. $F(v) \approx \log(v)$, if v is a vector, then $F(v)$ is also a vector whose each component is the approximation of the logarithm of corresponding v's component.

So we can rewrite Eq.(7) as $F(b_i(x_k)) = C_{ik} + \sum_{j \in N(i)} F(m_{ij}(x_k))$, where $C_{ik} = \log \eta + \log(\phi_i(x_k))$. Moreover, we use $\mathbf{b_i}$ as the status vector of beliefs of node i. For simplicity, assume there are only three classes, $x_k \in \{x_1, x_2, x_3\}$, i.e. $\mathbf{b_i} = \begin{bmatrix} b_i(x_1) \\ b_i(x_2) \\ b_i(x_3) \end{bmatrix}$ we can get the general equation for any node i,

$$F(\mathbf{b_i}) = \begin{pmatrix} F(\mathbf{m_{i1}}) & F(\mathbf{m_{i2}}) & \cdots & F(\mathbf{m_{in}}) \end{pmatrix} \mathbf{A_i} + \mathbf{C_i} \tag{10}$$

where $\mathbf{A_i}$ is the ith column of the adjacency matrix and $\mathbf{C_i} = \begin{bmatrix} C_{i1} \\ C_{i2} \\ C_{i3} \end{bmatrix}$ and $\mathbf{m_{ij}} = \begin{bmatrix} m_{ij}(x_1) \\ m_{ij}(x_2) \\ m_{ij}(x_3) \end{bmatrix}$, Now, replace $\phi_i(x_k) \prod_{n \in N(i) \setminus j} m_{ni}(x_k)$ with $b_i(x_k)/(\eta m_{ji}(x_k))$, Eq.(8) can be rewritten as $m_{ij}(x_k) = \sum_{x_g} \psi_{ij}(x_k, x_g) b_i(x_k)/(\eta m_{ji}(x_k))$, which means

$$\mathbf{m_{ij}} = \frac{1}{\eta} \Psi_{ij} \begin{bmatrix} b_i(x_1)/m_{ji}(x_1) \\ b_i(x_2)/m_{ji}(x_2) \\ b_i(x_3)/m_{ji}(x_3) \end{bmatrix} \tag{11}$$

where Ψ is the correlation (transition) matrix of every two nodes, i.e., under the three class assumption,

$$\Psi_{ij} = \begin{bmatrix} \psi_{ij}(x_1, x_1) & \psi_{ij}(x_1, x_2) & \psi_{ij}(x_1, x_3) \\ \psi_{ij}(x_2, x_1) & \psi_{ij}(x_2, x_2) & \psi_{ij}(x_2, x_3) \\ \psi_{ij}(x_3, x_1) & \psi_{ij}(x_3, x_2) & \psi_{ij}(x_3, x_3) \end{bmatrix} \tag{12}$$

It can also be written as

$$\mathbf{m_{ij}} = \frac{1}{\eta}\Psi_{ij} \begin{bmatrix} 1/m_{ji}(x_1) & 0 & 0 \\ 0 & 1/m_{ji}(x_2) & 0 \\ 0 & 0 & 1/m_{ji}(x_3) \end{bmatrix} \begin{bmatrix} b_i(x_1) \\ b_i(x_2) \\ b_i(x_3) \end{bmatrix} \tag{13}$$

Suppose all nodes are homogeneous, i.e. $\forall i, j\ ,\ \Psi_{ij} = \Psi$ is non-singular (which is also the common case), then we get a relation between the state of one node and two different direction messages on its one edge (since η is just a normalization constant, we ignore η in the following analysis):

$$\mathbf{b_i} = diag(\mathbf{m_{ji}})\Psi^{-1}\mathbf{m_{ij}} \tag{14}$$

Finally, we get the matrix equations for belief propagation,

$$\mathbf{b_i} = diag(\mathbf{m_{ji}})\Psi^{-1}\mathbf{m_{ij}} \quad \forall i \in [n],\ j \in N(i) \tag{15}$$

$$F(\mathbf{b_i}) = \begin{pmatrix} F(\mathbf{m_{i1}}) & F(\mathbf{m_{i2}}) & \cdots & F(\mathbf{m_{in}}) \end{pmatrix} \mathbf{A_i} + \mathbf{C_i} \quad \forall i \in [n] \tag{16}$$

3.4.1 Simple correlation matrix case

In many cases, one node is inclined to connect with other nodes which are in the same class as itself. For example, when we buy a book in Amazon, it is very possible that we also buy another book, which has close relation with the first one (e.g. supplementary textbook), or the one that is suggested by the suggestion system. Another instance is that when we post a scenery photo on facebook, it is very likely that at the same time you will post more scenery photos instead of other kinds of photos. Based on this fact, we assume a simple case that the correlation matrix is almost as an identity matrix. In this case, $\Psi^{-1} \approx \mathbf{I}$, then

$$\mathbf{b_i} \approx diag(\mathbf{m_{ji}})\mathbf{m_{ij}} = \mathbf{m_{ji}} \circ \mathbf{m_{ij}} \tag{17}$$

where "$\circ$" is the hadamard product (element-wise product). However, the identity correlation matrix has a problem: if the graph is a strong connected graph, there is no feasible solution exists according to Eq.(17), since all the nodes are connected and they can only all belong to the same class. To avoid this problem, we allow the deviation for all beliefs,

$$\mathbf{b_i} \leq \mathbf{m_{ji}} \circ \mathbf{m_{ij}} + \mathbf{s_i}$$

and $\mathbf{s_i} \geq 0$, i.e. every entry of $\mathbf{s_i}$ is greater than 0. But put penalty on these slack variables. Also note that $\mathbf{m_{ij}}$ and $\mathbf{m_{ji}}$ always come in pairs, we can use $\mathbf{m'_{ij}}$ $(= \mathbf{m'_{ji}})$ to replace $\mathbf{m_{ji}} \circ$

$\mathbf{m_{ij}}$ ($= \mathbf{m_{ij}} \circ \mathbf{m_{ji}}$), so basically we need to solve this optimization problem,

$$\begin{aligned}
\underset{\mathbf{s}}{\text{minimize}} \quad & \frac{1}{2} \sum_{k \in [n] \setminus V_0, j \in N(k)} ||\mathbf{b_k} - \mathbf{m'_{kj}}||^2 + \lambda \sum_{i}^{n} \mathbf{s_i} \\
\text{subject to} \quad & \mathbf{b_i} - \mathbf{s_i} \leq \mathbf{m'_{ij}}, \quad i \in V_0, j \in N(i) \\
& \mathbf{b_k} - \mathbf{s_k} \leq \mathbf{m'_{kj}}, \quad k \in [n] \setminus V_0, j \in N(k) \\
& \mathbf{s_i} \geq 0 \quad i \in [n] \\
& \mathbf{b_i} \geq 0 \quad i \in [n] \setminus V_0 \\
& \mathbf{m'_{ij}} \geq 0 \quad i \in [n], j \in N(i)
\end{aligned} \tag{18}$$

Note that the node set V_0 is the labeled node set, i.e. $\mathbf{b_i}$, $i \in V_0$ is given. λ is the adjustable weight which determines the importance of the deviation in our final result and remember $\mathbf{m'_{ij}} = \mathbf{m'_{ji}}$. If we want more initial labeled nodes can fit in this model, then give a larger value to λ, but meanwhile the estimation of the target nodes' beliefs tend to become worse. There are many methods to solve this standard optimization problem. We just give an example with Karush-Kuhn-Tucker (**KKT**) conditions which finally winds up into an equation system.

Add lagrangian multiplier $\alpha_{ij} \geq 0$, $\beta_i \geq 0$, $\mu_i \geq 0$ and $\nu_i \geq 0$, $i \in V_0$, $j \in N(i)$ here, we get

$$\begin{aligned}
& \frac{1}{2} \sum_{k \in [n] \setminus V_0, j \in N(k)} ||\mathbf{b_k} - \mathbf{m'_{kj}}||^2 + \lambda \sum_{i}^{n} \mathbf{s_i} + \sum_{i=1}^{n} \alpha_{ij} (\mathbf{b_i} - \mathbf{s_i} - \mathbf{m'_{ij}}) \\
& - \sum_{i \in [n]} \beta_i \mathbf{s_i} - \sum_{i \in [n] \setminus V_0} \mu_i \mathbf{b_i} - \sum_{i \in [n], j \in N(i)} \nu_i \mathbf{m'_{ij}}
\end{aligned}$$

It is easy to see that this optimization problem is a convex optimization problem, but we also need to assume there are positive feasible solution (i.e. strong convexity of optimization problem). Then according to the **KKT** conditions, we can get the global optimal solution.

$$\mathbf{b_k} - \mathbf{m'_{kj}} - \mu_k \vec{1} = 0 \qquad \forall k \in [n] \setminus V_0, j \in N(k) \tag{19}$$

$$-\mathbf{b_k} + \mathbf{m'_{kj}} - \alpha_{jk} \vec{1} - \nu_k \vec{1} = 0 \qquad \forall k \in [n] \setminus V_0, j \in N(k) \ \ and \ \ j \in V_0 \tag{20}$$

$$-\mathbf{b_k} - \mathbf{b_j} + 2\mathbf{m'_{kj}} - \nu_k \vec{1} = 0 \qquad \forall k \in [n] \setminus V_0, j \in N(k) \ \ and \ \ j \in [n] \setminus V_0 \tag{21}$$

$$\alpha_{ij} = \alpha_{ji} = 0 \qquad \forall i \in V_0, j \in N(i) \ \ and \ \ j \in V_0 \tag{22}$$

$$\lambda - \sum_{j \in N(i)} \alpha_{ij} \vec{1} - \beta_i \vec{1} = 0 \qquad \forall i \in V_0 \tag{23}$$

$$\alpha_{ij} (\mathbf{b_i} - \mathbf{s_i} - \mathbf{m'_{ij}}) = 0 \qquad \forall i \in V_0, j \in N(i) \tag{24}$$

$$\beta_i \mathbf{s_i} = 0 \qquad \forall i \in V_0 \tag{25}$$

$$\mu_i \mathbf{b_i} = 0 \qquad \forall i \in [n] \setminus V_0 \tag{26}$$

$$\nu_i \mathbf{m'_{ij}} = 0 \qquad \forall i \in V_0, j \in N(i) \tag{27}$$

Even though it seems very complicated, all of these equations are very simple, and we need all of these equations to get the optimal result. Eq.(19)$\sim$ Eq.(22) are from the stationary condition of **KKT** condition, which describe the relationship between any two nodes (there are three kinds of node pair (i, j) (1) i, j are both labeled nodes in V_0, ; (2) one of i, j is labeled node and another one is the target node; (3) i, j are both target nodes. The belief results are all $\mathbf{b_k}$, $k \in [n] \setminus V_0$.

3.4.2 Extended correlation matrix case

Before we talk about the common correlation matrix, we should think about why the optimization problem proposed above works. See Fig. 3 for an example part of a small **BP** network. The idea of the previous optimization problem comes from ignoring the real value of the m_{ij} and m_{ji}, but using a generalized variable m'_{ij} to describe the relationship between two nodes i and j.

Assume node i is a labeled node, i.e. $\mathbf{b_i}$ is known, according to last subsection, the value of $\mathbf{m'_{ij}}$ can be bounded by $\mathbf{b_i}$ and $\mathbf{s_i}$. On the other hand, if node j is an unknown (target) node, $\mathbf{b_j}$ can also be bounded by $\mathbf{m'_{ji}}$ and $\mathbf{s_j}$. The key point here is that $\mathbf{m'_{ij}} = \mathbf{m'_{ji}}$. This equation means we can approximate $\mathbf{b_j}$ with $\mathbf{b_i}$, $\mathbf{s_i}$ and $\mathbf{s_j}$. $\mathbf{s_i}$ and $\mathbf{s_j}$ are just the slack variables we defined, and we minimize all the slack variables in the objective functions. In one word, the captured relation between $\mathbf{m'_{ij}}$ and $\mathbf{m'_{ji}}$ can be used to describe the relation between $\mathbf{b_i}$ and $\mathbf{b_j}$, which is the objective to be optimized.

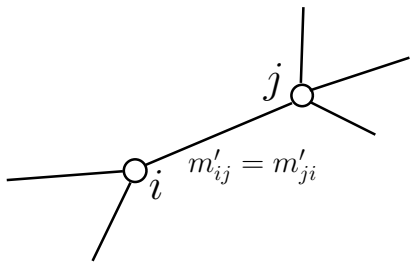

Figure 3: Small example of BP network

Recall that we actually use $\mathbf{m'_{ij}}$ to replace $diag(\mathbf{m_{ji}})\Psi^{-1}\mathbf{m_{ij}}$. Obviously that $\mathbf{m'_{ij}} = \mathbf{m'_{ij}}$ does not hold for common correlation matrix Ψ. But if we have

$$\mathbf{m'_{ij}} \approx \mathbf{m'_{ji}}$$

which holds in some cases, for instance that if Ψ is close to $\mathbf{I}$. This assumption means that the analysis in the simple correlation matrix case still works. Also, we need to think about that why the slack variables $\mathbf{s_i}$, $i \in [n]/V_0$ are necessary. Simply speaking, the main reason is that we just use Eq.(15) without Eq.(16) to formulate the optimization problem. Although Eq.(15) is derived by substituting $\phi_i(x_k)\prod_{n\in N(i)\setminus j} m_{ni}(x_k)$ with $b_i(x_k)/(\eta m_{ji}(x_k))$, which comes directly from Eq.(16), we still lose parts of all the information of the original belief updating rule.

Due to the limitation of time, there are still two unsolved problems which can be interesting topics for further study. The first one is that we provide an equation system which is used to solve the optimization problem. How to organize these equations into a more concise form is not a easy task. The second problem, how can we derive an upper bound for the distance between the solution to the optimization problem and the solution of the original Belief Propagation algorithm. For example, if we are given $||\mathbf{m'_{ij}} - \mathbf{m'_{ji}}||_F^2 \leq \epsilon$, (here is the componentwise norm, i.e. Frobenius norm), we hope to derive an error upper bound as a function of ϵ. Since we do not have time to implement some experiments to test the accuracy of this method, testing this result on different networks may be the next step.

3.4.3 Quadratic approximation

We also try another direction (relaxing the approximation assumption) to extend the **FaBP**, even though we do not really figure it out. A significant fact of the original **FaBP** algorithm is the assumption that all parameters are "about half", i.e. close to 1/2. This assumption provides a foundation for the correctness of Maclaurin series expansion used in the analysis. However, when there are more than two classes, without the concept of odd-ratio, we cannot achieve the similar assumption for any parameter. But we can use the quadratic approximation, i.e. second order Taylor approximation for the logarithms, i.e. instead of using $\log(x) \approx \log(1) + \log'(1)(x-1)$ with the assumption that x (the odd-ratio) is close to $\frac{1}{2}$, we use

$$\log(x) \approx \log(1) + \log'(1)(x-1) + \frac{\log''(1)}{2}(x-1)^2$$

without the "about half" assumption.

More precisely, in Eq.(16), $F(v) = (v-1) - \frac{(v-1)^2}{2}$

Still, for simplicity we assume there are three states for the node, i.e. $x_k \in \{x_1, x_2, x_3\}$. Then the left hand side of Eq. (16) is

$$\begin{bmatrix} b_i(x_1) \\ b_i(x_2) \\ b_i(x_3) \end{bmatrix} \circ \begin{bmatrix} 2 - \frac{1}{2}b_i(x_1) \\ 2 - \frac{1}{2}b_i(x_2) \\ 2 - \frac{1}{2}b_i(x_3) \end{bmatrix} - \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} = diag(\mathbf{b_i})(\begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} - \frac{1}{2}\mathbf{b_i}) - \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix} \tag{28}$$

note that

$$diag(\mathbf{b_i})\vec{\mathbf{1}} = \begin{bmatrix} b_i(x_1) & 0 & 0 \\ 0 & b_i(x_2) & 0 \\ 0 & 0 & b_i(x_3) \end{bmatrix} \vec{\mathbf{1}} = \mathbf{b_i} \tag{29}$$

Finally we get

$$diag(\mathbf{b_i})(2 \cdot \vec{\mathbf{1}} - \frac{1}{2}\mathbf{b_i}) - \frac{1}{2}\vec{\mathbf{1}} = \begin{pmatrix} \mathbf{M_{i1}} & \mathbf{M_{i2}} & \cdots & \mathbf{M_{in}} \end{pmatrix} \mathbf{A_i} + \mathbf{C_i} \tag{30}$$

where $\mathbf{M_{ij}}$ can also be written as $diag(\mathbf{m_{ij}})(2 \cdot \vec{\mathbf{1}} - \frac{1}{2}\mathbf{m_{ij}}) - \frac{1}{2}\vec{\mathbf{1}}$.

This new equation is still very complicated. But since the "about-half" assumption has been removed from this updating rule, it also has potential to be used in formulating a

optimization problem as we discussed in previous sections, and if it can be done, the accuracy of the optimization problem will be definitely improved. Even though we cannot find a good usage for this new approximation of the belief updating rule now, we still believe it is an attemptable research point.

4 Experiments

4.1 Datasets

4.1.1 DBLP networks

Description of Dataset

DBLP is an online publication database, focused on Computer Science.

A labeled subset of DBLP is used in our experiment. It consists of 14,376 papers, 14,475 authors, 8920 terms, 20 conferences in total. A fraction of nodes are labeled with which of the following four fields it belongs to, AI, DM,IR and DB. Among the labeled nodes, 4057 authors, 100 paper, 20 conference are included.

Graph Construction and Information

The graph is constructed as follows: we simply put all the nodes together. So it will be heterogeneous. The basic information about this graph is shown in the following table.

(a) DBLP Network

#nodes	#edges
4177	170794

(b) Label distribution

#DB	#DM	#AI	#IR
1230	763	1029	1155

4.1.2 Social Networking(KDD Cup 2012)

Description of Dataset

This dataset comes from Tencent Weibo, one of the largest micro-blogging website in China. It contains many different types of information and was chosen to be the material for KDD Cup 2012.

Despite the original purpose of the competition of KDD Cup, we found something interesting from the dataset: We can get the label of some users. There is a category system in which users are labeled with hierarchy categories. And there is a following history for each user.

Graph Construction and Information

As the BP algorithm is defined on undirected graphs, we need to fit the dataset into a more applicable format. The following relationship is naturally uni-directional, however, most people would follow back to his/her follower if he/she found that the particular follower is similar to himself/herself. Thus, we can keep the users and their bi-directional relationship to get a undirected graph in which edges indicates the similarity of two nodes.

After filtering the dataset, we extracted all users labeled "1.1", "1.4" or "1.6", constructing a graph with 1741 nodes and 18718 edges.

(c) Tencent Network

#nodes	**#edges**
1741	18718

(d) Label distribution

#1.1	**#1.4**	**#1.6**
687	524	530

Following is the number of edges between each categories:

	1.1	1.4	1.6
1.1	5976	66	19
1.4	66	2940	20
1.6	19	20	9592

Table 1: Transition Matrix of Tencent Network

4.1.3 Amazon Product co-purchasing networks

Description of Dataset

This network represents co-purchasing relationship between products on Amazon website. Most nodes fall into four major categories: Book, Music, DVD and Videos. The whole net work consists of 542684 nodes and 3387388 edges in all.

Graph Construction and Information

Although the relationship of this network is defined as co-purchasing, actually it is not symmetry. So we first filter a subgraph, in which each edge represents mutual co-purchasing relationship. The basic information of this mutual co-purchasing network is illustrated in the following tables.

Because this is a huge graph, the full scale picture of this graph mixed everything together and it is hard to tell any useful pattern from this graph, so we randomly sample a 5000 node subgraph, which is shown Figure 5.

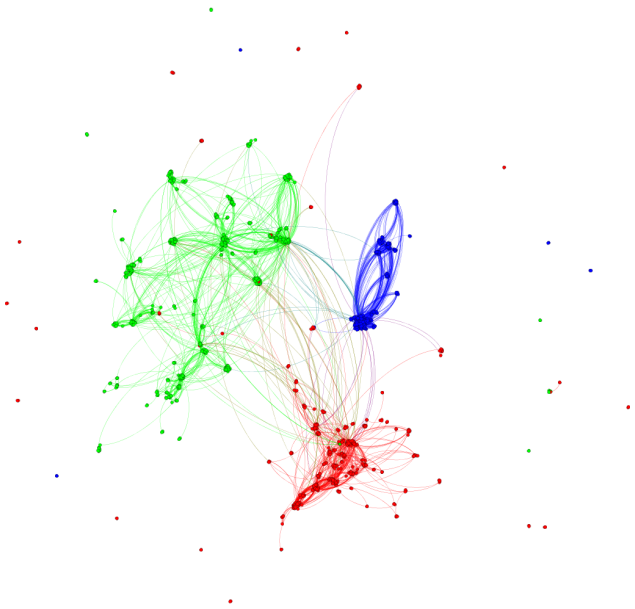

Figure 4: Visualization of Tencent network

(a) Co-purchasing Network

#nodes	#edges
393361	922867

(b) Label distribution

#Books	#Music	#DVD	#Video
287678	74867	18923	11893

4.1.4 Flickr photo network

Description of Dataset

This data set is from MIR (Multimedia Information Retrieval) FLICKR Retrieval Evaluation.In this dataset, photos are associated with one or more tags from Flickr users. Photos are also annotated manually by annotators with a category. We build this network of photos based on whether two photos share same tags.

There are 22,872 photos that have at least one tag in this dataset. Tags are from Flickr users, like "dog", "doors", "ocean" etc. For photos with at least one tag, each of them is associated with 9.8 tags on average. The photo with largest number of tags has 75 tags. The photo with smallest number of tags has only 1 tag. The number of photos only with 1 tag is 813. The number of photos with 2 tags is 987.

The photos are manually annotated with one or more of the 24 categories like "animals", "baby", "indoor" etc. There are 24,581 photos with at least one category. For photos with at least one category, each of them is associated with 3.8 categories on average. The most widely used category is "people" (10,373 times) followed by "structures" (9992 times).

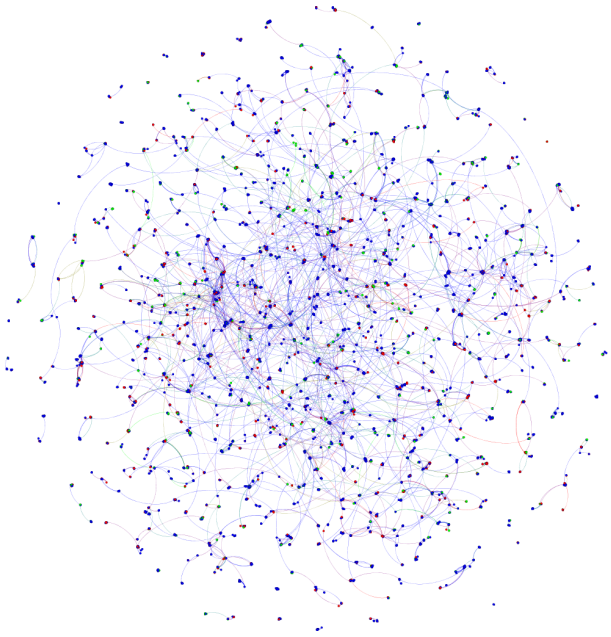

Figure 5: Visualization of Amazon network

Graph Construction and Information

We extracted a subgraph using photos with only one of the annotations "people", "structures" or "plant_life", and linked photos if they share at least three tags. In this graph, there are 6131 nodes.

(c) Flickr Network

#nodes	**#edges**
6131	114698

(d) Label distribution

#People	**#Structures**	**#Plant_life**
2030	1974	2127

Following is the number of edges between each annotation,

	people	structures	plant_life
people	19520	8742	9032
structures	8742	18050	18038
plant_life	9032	18038	41316

Table 2: Transition Matrix of Flickr Network

The network is illustrated in Figure 6.

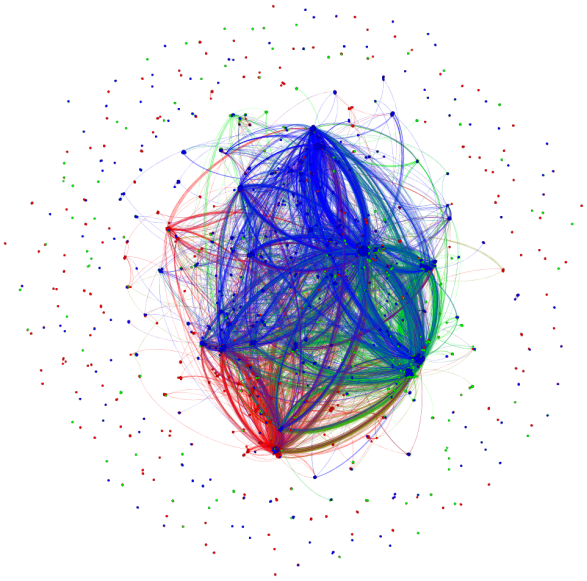

Figure 6: Visualization of Flickr network

4.2 Experiment Settings

4.2.1 ECOC approaches

Overview

In order to validate the performance of our proposed methods, on each graph we carried out the following experiments:

1. **MBP**: Multiclass BP
2. **OVA**: One-vs-All schema using **FaBP** as underlying classifier
3. **ECOC**: Error correcting code schema using **FaBP** as underlying classifier

Multiclass BP

In order to run Multiclass BP, we should first specify the following parameters:

- ϕ : Priors for all the nodes.
- ψ : The correlation matrix which specifies the *closeness* between two labels.

To estimate ϕ, before each experiment, we randomly select $p\%$ of the set of nodes whose labels are already known to us. The $p\%$ for each graph is listed in Table 3.

As to correlation matrix ψ, we estimate this matrix using the following strategy: for a pair of given labels, say A and B, we calculate all the edges between the two labels, denoted

Datasets	p
DBLP	20%
Flickr	8%
Tencent	5%
Amazon	30%

Table 3: Percentage of labelled data in prior ϕ

as $\#(AB)$. For each label, we also compute the number of edges associated with that label, which means all the edges that have at least one end being the given label. In our case, we denote them as $\#(A)$ and $\#(B)$. Then ψ is defined as follows:

$$\psi_{A\to B} = \frac{\#(AB)}{\#(A)}$$
$$\psi_{B\to A} = \frac{\#(AB)}{\#(B)}$$

Repeat the above procedure for each pair of labels, then we have the estimation of the correlation matrix.

One-VS-All with FaBP

FaBP algorithm is parameterized by one single important factor: the "about-half" homophily factor, denoted as h_h. The bigger h_h means the stronger the homophily phenomenon in the graph.

Datasets	h_h
DBLP	0.002
Flickr	0.2
Tencent	0.002
Amazon	0.01

Table 4: The homophily factor for each network

As to the prior ϕ, we follow the same settings with **MBP**.

When the algorithm finishes, for each node, say N, we combine all the k probabilities $P(N = k_i)$ in a vector, which specifies how likely the given node is within that class. Then we pick the class that has the greatest $P(N = k_i)$ as the label of that node.

ECOC with FaBP

The settings of prior ϕ and homophily factor h_h are the same as those in **OVA**.

When coming to the code design, we adopt the exhaustive algorithm described in [1]. When given k labels, each label is encoded using $2^{k-1}-1$ bits. For the first class, we encode it using all ones. For the second class, we set the first 2^{k-2} bits of its coding as zeros, then the next $2^{k-2}-1$ bits as ones. For the third class, we set the first 2^{k-3} bits as zeros, the following 2^{k-3} as ones, which are followed by $2^{k-3}-1$ zeros. Repeating the this procedure until all the classes are defined.

The code matrix when $k=4$ is illustrated in Table 5.

class	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	0	0	0	0	1	1	1
3	0	0	1	1	0	0	1
4	0	1	0	1	0	1	0

Table 5: Code design when $k=4$

As described in **Methods** section, we run **FaBP** for each bit. For each node, after we get the probability vector, we transform it into a binary vector, using the following strategy: for a given bit b_i, if $P(b_i=1)>0.5$, then the corresponding position of the binary vector is set to 1, else to zero. Then we calculated the Hamming distance between this vector and coding vector of each class, the one has the smallest distance is assigned to the node.

4.3 Experiment Results

4.3.1 Convergence Detection

We run general BP on all of our datasets and figure 7 is the plot of the percent of converged messages over each iteration:

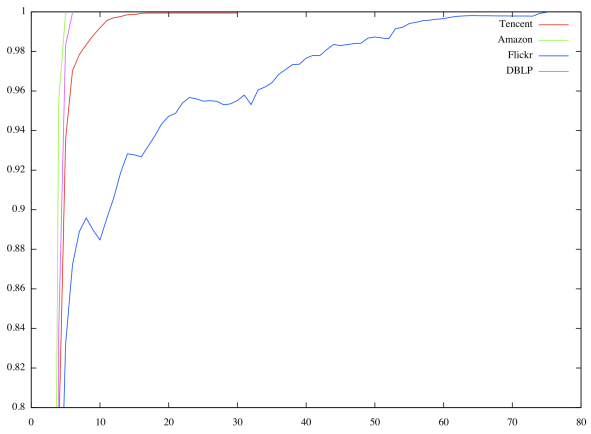

Figure 7: Convergence vs iteration

As the plot shows, some dataset needs more iterations to get a satisfying number of converged messages. This is why we think convergence detection is worth implementing.

There is a subtle problem of convergence detection that when we tried it on a large graph (Amazon co-purchase data), the graph got converged quickly even some messages were still thrashing. This is because most of the messages got stable and dragged the average difference lower enough to pass the threshold.

It would be better if we can detect whether there is a clique of messages which does not exceed the threshold.

4.3.2 Message Smoothing

In some cases where traditional loopy BP cannot converge, message smoothing can help the algorithm to converge. We implemented message smoothing on PEGASUS BP and run it on all of the datasets. We found when the BP can converge without message smoothing, message smoothing actually slows down the convergence, which makes sense because it dampens the message updating process and avoids changing messages too drastically.

Following graph shows this phenomenon, we can see, eventually both traditional BP and BP with message smoothing converges, but BP with message smoothing converges a little slower. Because BP converges on all of our datasets, we do not have a graph which shows message smoothing helps BP to converge.

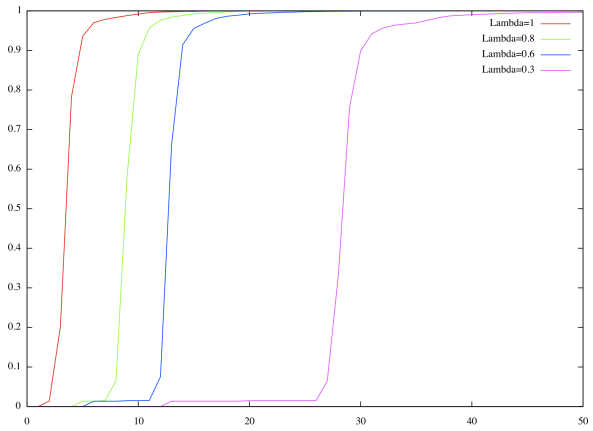

Figure 8: BP with Message Smoothing

4.3.3 Multiclass extension evaluation

All the results are illustrated in the following table. The metric we use here is the accuracy of label prediction of the nodes whose labels are not used as prior in the experiment.

Datasets	MBP	OVA	ECOC
DBLP	0.17	**0.776**	0.726
Flickr	**0.653**	0.421	0.13
Tencent	0.953	**0.961**	0.924
Amazon	**0.736**	0.636	0.626

Table 6: Experiment result for different networks

Result Analysis

From Table 6, we can see that **MBP** achieves highest accuracy in Flickr and Amazon dataset, but did worst in DBLP dataset.

After further study of the result, we find out **MBP** gives every node the same label except those whose labels are already used as prior in Amazon and DBLP dataset. In DBLP network, **MBP** gives every node the label "DM", while in Amazon network, **MBP** labels every node as "Books". On the contrary, **OVA** and **ECOC** gives out much "smooth" results, a significant amount of "Music","DVD" and "Video" are correctly labeled. This tells us that accuracy alone is misleading. If the underlying label distribution is not as skew as the amazon dataset but more like the DBLP network in which there isn't a dominant label, then the performance of **MBP** worsens greatly.

On the other hand, the **OVA** scheme achieves best average performance over all datasets. Comparing to the other two methods, its performance is more stable. It is more "fault-tolerant" in the sense that it is immune from the underlying distribution of labels.

As to ECOC methods, it gains relative the similar accuracy with **OVA** method in 3 networks. In all the networks we tested, its performance is always slightly lower than **OVA** method. One possible explanation is that, the number of labels is relatively small in our experiment. When $k = 3$, the number of bits is also 3. So ECOC method can not set apart from **OVA**. Another possible explanation is that as the result gives by **FaBP** is not accurate and very close to 0.5, the more accurate technique that uses loss based function to decide label is useless in our setting, but the Hamming distance based method loses information from original probabilities, so the result is not as good as **OVA**.

We can also observe that those methods work better on some datasets than other. The possible reasons are as follows.

If the transition matrix of a network has significant difference between each value in one row, BP works well. For example, in a network with three labels A, B, C, if the transition probability from A to A, A to B, A to C defers a lot, BP performs well. We can see, from the network of Tencent, the transition probability from each label to itself is much larger than the transition probability to the other two labels. We can also see this property on the network plot of Tencent dataset: each label forms a cluster and separated from the other two labels. The result of BP on Tencent dataset is more accurate compared with other datasets. On contrary, from the Flickr dataset, we can see the three values in each row of the transition matrix are close to each other. For example, the transition probability from "structures" to

"structures" and the transition probability from "structures" to "plant_life" are almost the same. We can also see this property from the plot of Flickr network, in which nodes of the three labels are mixed together. As a result, the BP does not work well on Flickr dataset compared to on Tencent dataset.

For the one-vs-all method, we treat all labels other than A as NOT A. Then we run FastBP on this newly built network and get the probability of being A for each node. This method can be problematic if the transition probability of the newly mixed label NOT A is different from that of original labels. In our experiment, the transition property of each label in Tencent dataset is similar, so this one-vs-all FastBP achieves close results as BP on this dataset. On contrary, the transition properties of any two labels in the Flickr dataset are so different that they can hardly be merged without losing their own characteristics. So the result of one-vs-all FastBP is worse than result of BP.

5 Conclusions

In this project, we studied and explored several aspects of belief propagation. Specifically, we studied the principle of belief propagation and its convergent conditions and how to improve its convergence. We implemented convergence detection and message smoothing on PEGASUS implementation of BP. Experiment shows these two implementations work as expected.

We explored how to extend FastBP to multiclass using several different methods.

We designed and performed general ECOC algorithm and one-vs-all algorithm on four networks. The result shows that **OVA** method gains best accuracy among two of the four networks, and is slightly better than the general ECOC method. It also shows that **OVA**s performance is more stable than Multiclass BP, and gives more smooth predictions.

We also tried to approximate BP as an optimization problem, based on the assumption that the correlation matrix is close to an identity matrix. But there are still several problems exist. The first one is that the final solution to the formulated optimization problem still needs to be simplified. The second problem, how can we derive an upper bound for the distance between the solution to the optimization problem and the solution of the original Belief Propagation algorithm. Last but not least, a new approximation for the logarithm operation without any strict assumption may improve the accuracy of the approximation.

References

[1] Thomas G. Dietterich.et.al. Solving multiclass learning problem via error-correcting output codes. *Journal of Artifical Intelligence Research*, 1995.

[2] Pedro Felzenszwalb, , Pedro F. Felzenszwalb, and Daniel P. Huttenlocher. Efficient belief propagation for early vision. In *In CVPR*, pages 261–268, 2004.

[3] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 229–238, Washington, DC, USA, 2009. IEEE Computer Society.

[4] U Kang.et.al. Inference of beliefs on billion-scale graphs. *Large-scale Data Mining: Theory and Applications 2010, in conjunction with KDD 2010*, 2010.

[5] Danai Koutra, Tai-You Ke, U. Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *ECML/PKDD (2)*, pages 245–260, 2011.

[6] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 47:498–519, 1998.

[7] Erin L.Allwein.et.al. Reducing multiclass to binary : A unifying approach for margin classiers. *Journal of Machine Learning Research*, 2000.

[8] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[9] Dmitry M. Malioutov, Jason K. Johnson, and Alan S. Willsky. Walk-sums and belief propagation in gaussian graphical models. *J. Mach. Learn. Res.*, 7:2031–2064, December 2006.

[10] Joris Mooij and Hilbert Kappen. Sufficient conditions for convergence of loopy belief propagation. In *Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 396–403, Arlington, Virginia, 2005. AUAI Press.

[11] John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems 12*, pages 547–553, 2000.

[12] Marco Pretti. A message-passing algorithm with damping. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11):P11008, 2005.

[13] Ryan Rifkin.et.al. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 2004.

[14] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Exploring artificial intelligence in the new millennium. chapter Understanding belief propagation and its generalizations, pages 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[15] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51:2282–2312, 2005.

A Appendix

A.1 List of Innovations

- Evaluate BP and Fast BP on many real world datasets.
- Improve BP on Hadoop with convergence detection.
- Use ECOC to extend Fast BP to multiclass.
- Use quadratic approximation to extend Fast BP to multiclass.

A.2 Labor Division

The team will perform the following tasks

- Implementation of BP on Hadoop with convergence detection [Yu Su,Huanchen Zhang,1 weeks]
- Parse data and build network from datasets mentioned above [All, almost done]
- Evaluate BP and Fast BP on large graphs [All, already have preliminary results, 1 week]
- Research on extending Fast BP algorithm to Multiclass Problem [Guanyu Wang,Yuchen Tian, 2 weeks]
- Implement and evaluate multiclass Fast BP algorithm [Yu Su,Huanchen Zhang,1 weeks]

A.3 Activities

- Guanyu Wang: Performed dataset analysis and network building. Read papers and tried to deduct quadratic approximation for FaBP extension.
- Yuchen Tian: Performed dataset analysis and network building. Proposed error correcting code methods for FaBP extension.
- Yu Su: Performed dataset analysis and network building. Did some preliminary experiments.
- Huanchen Zhang: Performed dataset analysis and network building. Did some preliminary experiments.

A.4 Previous Labor Division

The team will perform the following tasks

- Implementation of BP on Hadoop [Yu Su,Huanchen Zhang,2 weeks]
- Apply BP on Large Graph [All,2 weeks]
- Research on extending Fast BP algorithm to Multiclass Problem [Guanyu Wang,Yuchen Tian,2 weeks]

Contents