Convergence of Belief Propagation

An empirical study on random graph

AM 254

Enmao Diao

Abstract

The Belief Propagation (BP), also known as Sum-Product Algorithm [1], is a popular algorithm for approximate inference on probabilistic graphical models (Bayesian networks and Markov random fields in particular). This algorithm is originally invented by Pearl in 1988 [2]. It has been extensively studied during the last 10 years. Its elementary version and other variations are applied in diverse fields such as error correction codes for Turbo Codes and Low Density Parity Check Codes [3], combinatorial optimization (satisfiability problems such as 3-SAT and graph coloring [4]) and computer vision [5][6]. In practice, BP not always converges. Significant progress has been made in recent years regarding the sufficient conditions of BP convergence to a unique fixed point. However, many theoretical conditions cannot be numerically solved and general solution is still open for investigation. In this report, we will focus on the convergence rate of BP aiming to indicate certain linear relationships between the basic parameters of random graphs and the convergence rate of BP. We will show that certain basic properties can have a strong impact on the convergence rate of BP and thus may cause BP to oscillate in larger random graph case. First, we will discuss some theoretical conditions of BP convergence and show their potential relationship to each other. Then we will explicitly introduce the way BP is implemented from data structure to test case. In the end, we will investigate the relationship between the convergence rate and the parameters of random graphs and their corresponding factor graphs.

Literature Review

It is firstly known that the decoding algorithm of Shannon-limit performance of "Turbo Codes" is equivalent to loopy belief propagation in a chain-structured Bayesian network. Since then, BP has attracted a lot of attention regarding the approximation scheme in loopy BP case and the underlying convergence condition of BP. Murphy and Weiss conducted an empirical study focusing on the accuracy of approximation of loopy BP [7]. In the paper, they used two synthetic and two real world network and compared the results to the junction tree algorithm which can yield exact inference. It was shown that when BP converges, it gives a surprisingly good approximation to the correct posterior marginal distribution. However, it can also oscillate within a range having little correlation with the correct marginals. They suggested that the cause of oscillation is not simply a matter of size of network but rather more sophisticated reason. Numerically, it may be possible to check whether loopy propagation is appropriate for a given problem after a few iterations.

Researchers also developed a series of theoretical conditions regarding convergence of BP to a unique fixed point. First, they concentrated on a special case of BP in Gaussian graphical model. In this case, Frey [8] and Van Roy [9] showed there exists a stable fixed-point such that BP can yield highly accurate approximations in loopy case. Weiss [10] suggests that there is a striking similarity between the results of Gaussians in arbitrary networks and the results for single loop of arbitrary distributions. However, the results cannot be extended into arbitrary graphs and arbitrary potentials since the exact expression derived from algebraic structure for the error in belief propagation at any iteration holds only for Gaussian variables [11].

The work of Tatikonda and Jordan [12][13] showed that the convergence of loopy BP to a unique fixed point may be considered in terms of a Gibbs measure on the graph's computation tree. As shown in Figure 1, for a graph with cycles, one may show an equivalence between n iterations of loopy BP and the depth n computation tree. Ihler

[14] in his work suggested this leads to the result that loopy BP is guaranteed to converge if the graph satisfies Dobrushin's condition [15]. Acutally Tatikonda's work, by measuring the difference between two messages, is easier to check since Dobrushin's condition is a global measure, and difficult to verify. Based on Tatikonda's results, also known as Simon's condition, Ihler provides a stronger condition in the sense that it measures and creates a cruder bound of the error at each iteration than that of Simon's condition. However, both theoretical conditions require to run the algorithm for a few iterations in order to measure and bound the error at each iteration.

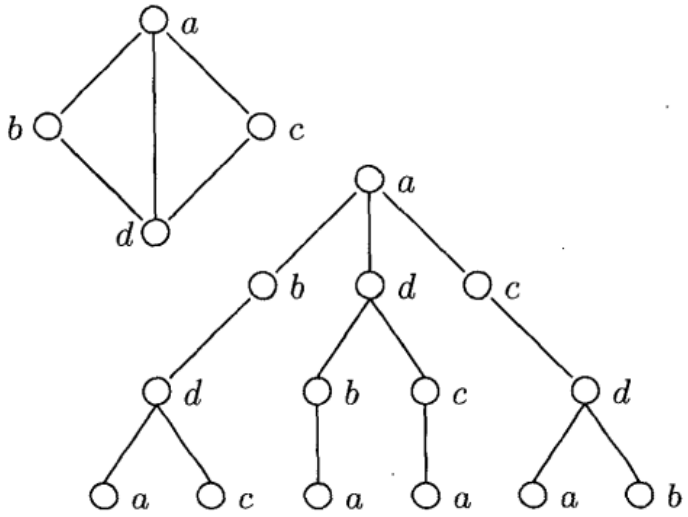

Figure 1. Equivalence between 3 iterations of loopy BP and the depth 3 computation tree [13].

Heskes [16] obtained sufficient conditions of the uniqueness of the BP fixed point with a completely different methodology. Several relatively strong conditions are drawn by studying the arbitrariness in the definition of the Bethe free energy when incorporated the constraints and exploiting the relationship between properties of the Bethe free energy and the BP algorithm. However, Mooij and Kappen [17] suggested that whether the uniqueness of the fixed point also implies convergence of BP seems to be an open question.

Mooij and Kappen suggested that both Tatikonda's and Ihler's results are only

formulated for pairwise, positive factors. Their derived condition is much stronger and is also applicable to arbitrary graphical models with discrete variables and nonnegative factors. The author also states that to solve the resulting optimization problem and thus to verify the condition, the cardinalities and connectivity should be small. The question of finding an efficient solution in the general case is left for future investigation.

Implementation Method

The general framework of our implementation is as follows.

1. Generate Erdos-Renyi undirected and directed random graph with a fixed number of nodes n and a uniformly generated probability p. Since we concentrate on the parameters of random graphs rather than the different types of graph, we make the graph generation process simple and general.

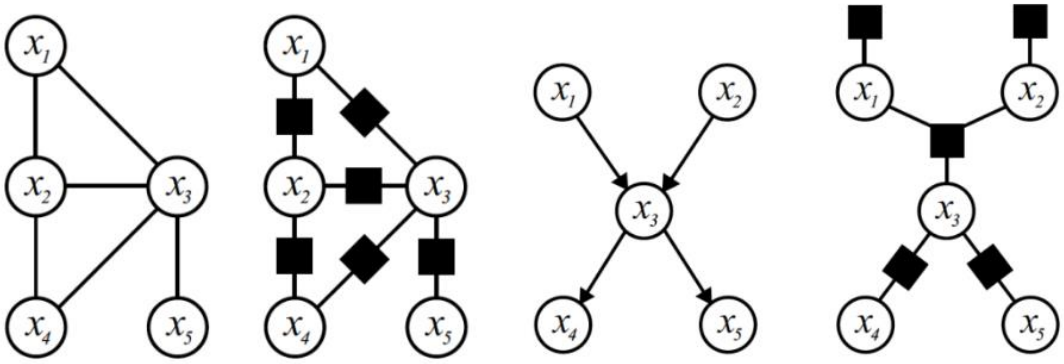

Figure 2. Transformation between undirected graphs and pairwise factor graphs, between directed graphs and high-order factor graphs.

2. In this step, we take in the adjacency matrix from step 1 and transform undirected and directed random graph to pairwise factor graph and high-order factor graph respectively as shown in Figure 2.

    a) The data structure of the factor graph is a Matlab struct with field *var* and *fac* representing variable nodes and factor nodes. Inside *var* field, there are *name*, *dim*, *id*, *nbrs_fac* and *observed* field representing name, dimension of prior probability distribution, id of itself, id of neighbor

factor nodes and observed value used for brute force method. In side *fac* field, there are *p*, *nbrs_var*, and *id* representing, potential matrix, id of neighbor variable nodes, and id of itself.

b) For better understanding, we initialize prior probability of each variable node for *outgoing, incoming,* and *oldoutgoing (for update step)* messages. The *outgoing*, *incoming,* and *oldoutgoing* message fields of each variable(factor) node are initialized as a cell array of length equal to the number of neighbor factor(variable) nodes. The prior probabilities of variable nodes are initialized inside each cell as a (*dim* x 1) column vector with each element equal to the *prior* value. The prior message of factor nodes initialized inside k^{th} cell is a (**Y** x 1) column vector and **Y** equal to the dimension of the variable node that has the id of kth neighbor variable nodes (k^{th} element of *nbrs_var*). The message inside k^{th} cell simply corresponds to the k^{th} element of *nbrs_var* or *nbrs_fac*.

c) For pairwise factor graph, the dimension of the potential matrix *p* is equal to 2 because there are only two variable nodes connected to each factor node and thus the length of *outgoing, incoming,* and *oldoutgoing* field inside each factor node is also 2.

d) For high-order factor graph, the first dimension of potential matrix corresponds to the variable node that is being pointed to as shown in Figure 2. Logically, the id of that variable node is the first element inside *nbrs_var* and the corresponding messages are also stored in the first cell of the three message fields. Here we define the length of *nbrs_var* as the *cardinality* of the node that is being pointed to. Later we will use this as one statistics for convergence rate analysis. One thing worth mentioning is that we create a new factor node connecting to those variable nodes that no other nodes point to.

e) We fix the number of variable nodes in factor graph to be the number of nodes in random graph since an isolated node in random graph does not produce a variable node in factor graph. If any isolated nodes were created, we restart from step 1.

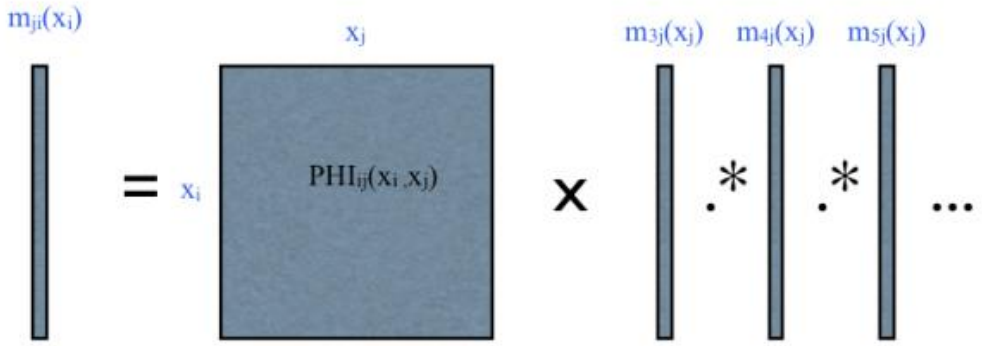

Figure 3. Depiction of belief propagation message passing rules, showing vector and matrix shapes.

3. After setup factor graph, we can run the BP algorithm according to following steps.

   a) Compute outgoing messages for factor nodes.

$$\widehat{\nu}_{a\to j}^{(t)}(x_j) \cong \sum_{\underline{x}_{\partial a\setminus j}} \psi_a(\underline{x}_{\partial a}) \prod_{k\in\partial a\setminus j} \nu_{k\to a}^{(t)}(x_k) \tag{1}$$

   Along the loop of each factor node, we store the outgoing message of factor nodes in the previous iteration to the *oldoutgoing* field. Then before we compute the outgoing message based on formula (1), we first have to consider how we can multiply high dimensional matrix with many column vectors along different dimensions as shown in Figure 3. We are not sure if there is a better way to handle this. The way we implement it are as follows. First, we use *repmat* to replicate column vectors along the dimension of potential matrix apart from its own dimension. For example, if the column vector **X** is from k^{th} incoming cell, we will do A = size(*p*);

A(k) = []; **Y**=repmat(X,[1 A]); The resulting **Y** matrix has the same size as potential matrix. Since now the first dimension of **Y** corresponds to k^{th} dimension of *p*, we have to inverse permute the first dimension to kth position with *ipermute()*. After we conduct this approach for k incoming cells, we will have k matrices with size equal to size(*p*). Now we will update for the outgoing message for the j^{th} variable node. We first element-wise multiply these k matrices except the j^{th} matrix because we do not want to take incoming message from the j^{th} variable node into account according to the formula. Then we need to sum up along all other dimensions to have a (dim_j x 1) vector. We can do this by permuting the j^{th} dimension of resulting size(*p*) matrix **Z** to the first dimension and use sum(Z, [2:end]). Then we can conduct this whole process for all k outgoing message cells for each factor nodes. The outgoing messages in each factor node are normalized by their values summing together.

b) Send outgoing messages from factor nodes to variable nodes.

Since the k^{th} outgoing message cell of the factor node with id_j corresponds to the variable node with id_m stored in the k^{th} position in *nbrs_var*. We extract the index j of the factor node with id_j in *nbrs_fac* inside variable node with id_m. We update the j^{th} incoming message cell of id_m variable node as the k^{th} outgoing message cell of factor node with id_j. This process is simple and maybe unnecessary but we add this step for better understanding and consistency.

c) Compute outgoing messages for variable nodes.

$$\nu_{j\to a}^{(t+1)}(x_j) \cong \prod_{b\in\partial j\setminus a} \widehat{\nu}_{b\to j}^{(t)}(x_j) \tag{2}$$

Based on formula (2), we do not need to permute any matrices in this step because all incoming messages of the j^{th} variable node are (dim_j x 1)

column vector. Therefore, if we compute the outgoing message for kth factor node among n factor nodes, we just need to element-wise multiply all n incoming column vectors except the k^{th} one.

d) Send outgoing messages from variable nodes to factor nodes.

Similar to step b), We update the k^{th} incoming message cell of id_j factor node as the j^{th} outgoing message cell of variable node with id_m. The outgoing messages in each variable node are normalized by their values summing together.

e) Check convergence

We check the L1 distance between outgoing and oldoutgoing messages of each factor node. If the L1 distances for all factor nodes are below the threshold, then we can state it converges and exit the iterations. Otherwise, we restart from step a).

4. Compute marginal probability and marginal expectation

a) Computing marginal probability is similar to 3.c) step. To compute the marginal probability, we do not exclude any incoming messages from k factor nodes. For marginal probability of the j^{th} variable node, we element-wise multiply all k (dim_j x 1) column vectors. The resulting one (dim_j x 1) column vector is the marginal probability.

b) To compute marginal expectation, we set the support of kth dimension equal to k. Then marginal expectation of the j^{th} variable node just equals to [1:length(dim_j)]*(dim_j x 1) column vector.

5. Check Correctness of results

We compare our results to the results from brute force method in small cases

(n=5). As soon as we have a larger graph, the brute force method fails due to memory limitation. Thus, we try to start on different priors and if they all converge to a unique fixed point then we believe the result is correct.

Empirical Study on Rate of BP Convergence

We conduct a convergence rate analysis based on the number of iterations of the network need to converge with respect to its corresponding properties. Initially, we set up many potential parameters and then narrow down into following 10 parameters that we will investigate in this section: number of edges, number of connected components, diameter (the longest shortest path), the average shortest path, radius (the minimum shortest path), number of size 3 loop, number of size 4 loop, number of independent loop, average cardinality, and maximum cardinality. We examine the statistics of these 10 parameters with MIT topology analysis toolbox [18]. The last two statistics regarding cardinality of variable nodes in the factor graph are already discussed in the previous section.

Notice that we fix the size of the random graph to be n throughout the experiment to investigate the parameters of random graphs in the same scale. As mentioned before, the number of variable nodes in the factor graph is equal to the number of nodes in random graph. Thus, the number of variable nodes in this experiment is n (=10). That is, the nodes of those generated random graphs have at least degree one.

Then we analyze the rate of convergence with respect to 10 parameters mentioned above. Totally we generate 500 copies for either type of random graphs. Since there are possibly multiple graphs having the same parameter value, for visualization simplification, we take average vertically such that we only have one average number of iterations with respect to one parameter value. We also illustrate the original plot of certain parameters for discussion.

Connectivity

Directed Graphs

Generally, the number of iterations to converge has a similar linear relationship with the number of edges and loops for undirected (pairwise factor) graph and directed (high-order factor) graph respectively as shown in Figure 4 (a-d) (See Appendix for Figures indexing from 4). The first intuition is that as the number of edges and loops grows, the graph becomes more and more connected and small connected components are connected to form a single connected component. However, this contradicts Figure 6(a) in which the number of iterations of directed graph is independent with the number of small connected components. When we dig into the data, we find that since we fix the number of variable nodes to be n, we add a bias to the generated random graphs and thus most of them are fully connected graphs (only one connected component) as shown in Figure 6(b).

Now consider the case where the directed graphs have many small connected components. In this case, the resulting high-order factor graph should be similar to the pairwise factor graph since the cardinality of each variable node is small. It makes sense to state that graphs with many small connected components converge faster than those with one giant connected component. From Figure 4(a) and (d), similar to the undirected graphs, there is a small increase of the number of iterations when we have very little edges and independent loops (many small connected components). However, comparing Figure 4(b) and 4(c), the average number of iterations is large even when we have very few number of size 3 and 4 loops. One possible explanation is that many graphs with a single connected component can also have a small number of size 3 and 4 loops and thus slow the convergence process. When we averaging them to make the simplified scatter plot, their large value of the number of iterations overwhelm the case where we have many small connected components and small number of size 3 and 4 loops. Figure 5(a) and (b), the original plot for size 3 and 4 loop, covers a large vertical

range in the beginning when having a small number of size 3 and 4 loops.

Then we consider the case where the directed graph is a fully connected graph. As the number of edges and loops grows, shown in the tail part of Figure 4(a-d), the number of iterations decreases. This illustrates an empirical phenomenon that the more connected the connected component is, the faster it will converge. This is related to the Murphy's study [7] in which he states that BP can work for graphs with small loops but should perform better with those having large cycles.

Summarize two empirical results stated above:

1) Graphs with many small connected components tend to converge faster than graphs with one giant connected component.
2) Graphs with one giant connected component which has large connectivity tend to converge faster than those with small connectivity. That is, connected graphs with more cycles tend to converge faster than those only having a small number of loops. (See Dimension and Cardinality section)

Combining these two results explains the contradiction that in Figure 6(a), graphs with one giant connected component also converge as fast as graphs with many small connected components on average. This is due to 2) connected graphs with large connectivity tend to converge faster and thus balance 1) the effect of graphs with many small connected components. This fact is also illustrated and confirmed by the original plot Figure 6(b).

One thing worth mentioning is that based on the first empirical results, can we conclude that considering graphs with one connected components and similar connectivity, the more nodes they have, the slower they converge? Since we fix the scale of random graphs in our experiment, we can only make a conjecture on this topic.

Undirected Graphs

Regarding undirected graphs, we can conclude a simple linear relationship from Figure 4 and 5. When 3) we have more edges and loops, we tend to converge much slower. First, undirected graphs with many small connected components also tend to converge faster as shown in Figure 6. However, the more connected graphs with one connected component are, the slower they converge, which is opposite to the directed graph case as shown in Figure 4 and 5. In the pairwise factor graphs, the cardinality is restricted to 2 and thus many loops in random graphs remain loops in factor graphs. Actually the reason I believe 2) is true is that when connected directed graphs tend to become more connected, their lower order factor nodes also tend to merge into high-order ones. This is illustrated in Figure 9(a-b) where large average and maximum cardinality of directed graphs tend to converge faster.

Shortest Path

We test rate of convergence with 3 parameters related to shortest path including diameter, the average shortest path, and radius. Generally, there are two phenomena that justifies the previous section results and one additional case that worth mentioning.

Notice that, in Figure 7 and 8, we make infinite shortest path to be zero indexed at the left of Figures. Infinite shortest path meaning the graph is disconnected and we have more than one connected components. In this case, Figure 7(a-c) partially justifies 1) in previous section. When the graph is disconnected, we tend to converge faster than when the graph is connected.

Besides, from Figure 7(a-b) and 8(a-b) we can see that when the longest and average shortest path is just 1, the directed graphs boil down to pairwise factor graph which has a large number of small loops and thus converge much slower. One typical phenomenon is that when the average shortest path just reaches beyond 1, the convergence rate drops

to the minimum and start to gradually increase. Combining the last paragraph, it clearly justifies 1) in the previous section.

Apart from justification for 1), when the graphs are fully connected, there is a substantial linear relationship from Figure 7(b) between the average shortest path and the rate of convergence. At first, we thought it indicated a similar structure as the connectivity, meaning longer shortest path indicates large connectivity. However, these three parameters regarding shortest path have a different effect on undirected graphs. As the average shortest path increases, undirected graphs tend to converge much faster. One possible explanation is that, when pairwise factor graphs have large average and especially minimum shortest path, there are not many small loops in the factor graphs and thus converge faster consistently. For high-order factor graphs, large shortest path actually means we have more 2-cardinality factor nodes. In this case, we do not take advantage of high-order factor nodes, and thus we tend to converge slower, just like pairwise factor graphs.

Dimension and Cardinality

Dimension is defined as the number of discrete value that the variable nodes can be and determines the outgoing and incoming messages to be a (dim x 1) column vector. Cardinality is the order of factor nodes. For example, if one factor node connects 9 variable nodes, then its potential matrix is a 9 dimensional matrix.

We run 100 iterations from dimension 2 to 5. Intuitively, dimension should have nothing to do with the rate of convergence and Figure 10(a-b) justifies that. As for cardinality, we do not discuss it with respect to undirected graphs, since pairwise factor graphs have cardinality always equal to 2. When average or maximum cardinality just equals to 2, directed graphs boil down to pairwise factor graph and thus converges as good as pairwise factor graphs. This is much slower than using high-order factor graphs. Because when cardinality increases, we eliminate potential loops in directed graphs by

merging low-order factor nodes into high-order ones. This phenomenon is illustrated by Figure 9(a-b).

Convergence Visualization

Originally, we implement visualization of convergence process graphically. However, convergence happens instantly and thus the change of coloring of nodes is not visible. Then we implement plot the marginal expectation across all variable nodes and the marginal distribution of one specific node as in Figure 11(a) and (b).

Summary

We implement a generalized version of Belief Propagation all the way from data structure to simulation. We transform (un)directed random graphs to (pairwise)high-order factor graphs and record some of their interesting properties like diameter and connectivity. We conduct an empirical study on convergence rate analysis with respect to those graph properties. Based on our analysis, we can summarize that certain properties can indicate the rate of convergence as follows:

1. For disconnected graphs, both directed and undirected graphs tend converge faster when they have more small connected components than one giant connected components.
2. For connected undirected graphs, more edges and small loops tend to slow the convergence process, since small loops in random graphs maintain the similar structure in pairwise factor graphs which have cardinality fixed to 2. Meanwhile, as the length of the shortest path increases, this kind of graph converges much faster, since longer shortest path indicates fewer cycles in the original graphs.
3. For connected directed graphs, more edges and small loops tend to accelerate the convergence process, since we take advantage of high-order factor graphs and merge many loops in random graphs into high cardinality factor graphs. Meanwhile, as the length of the shortest path grows, connected directed graphs

converge slower since part of the factor graphs have low cardinality and even may become pairwise, as you can see from the intersection of regression lines in Figure 7(a).

4. Generally, we if the graph is disconnected, we can analyze its connected component separately. Therefore, we should focus on connected graphs with nodes having at least degree one. We can use the number of edges, size 3,4, independent loops, diameter, the average shortest path, radius to estimate the convergence rate. However, based on our results, the best estimator of the rate of convergence with respect to this kind of graph is the average shortest path because it has the most prominent linear relationship with the rate of convergence.

There is also some improvement I wish I could do for this project. First, I wish to generate random graphs more wisely. Before this project, I have no idea what kind of graph I may work on and thus I choose the most general one. Now, if concentrating on only the connected graphs with all nodes at least degree one, we can even generate the graphs enumeratively.

Next, the scale of the random graph or the number of variable nodes is fixed throughout the experiment. This is partially due to the fact that the data structure we use requires Matlab to store all potential matrix in advance. However, we do not need them at the same time. In practice, we only need to store one potential matrix at a time and thus the memory consumption can be reduced to the maximum space taken by one particular potential matrix. It is also possible to have a situation where convergence rate of one large directed graph is satisfactory. However, there may exist a small number of very high-order factor nodes. Under this situation, we do need large memory space prepared. In fact, we could transform these particular part of high-order factor graphs into lower-order or even pairwise ones such that we can lower memory cost. Therefore, it is desirable to visualize the trade-off between convergence rate and memory space usage.

Reference

[1] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the Sum-Product Algorithm,” IEEE Trans. Inform. Theory, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[2] J. Pearl. Probabilistic reasoning in intelligent systems: Networks of plausible inference, 1992.

[3] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, “Turbo decoding as an instance of pearl’s ‘belief propagation’ algorithm,” IEEE J. Select. Areas Commun., vol. 16, pp. 140–152, Feb. 1998.

[4] A. Braunstein and R. Zecchina, “Survey propagation as local equilibrium equations,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2004, no. 06, p. P06007, 2004.

[5] K. Tanaka, “Statistical-mechanical approach to image processing,” Journal of Physics A: Mathematical and General, vol. 35, no. 37, pp. R81– R150, 2002.

[6] Sudderth, Erik B., et al. "Nonparametric belief propagation." Communications of the ACM 53.10 (2010): 95-103.

[7] Murphy, Kevin P., Yair Weiss, and Michael I. Jordan. "Loopy belief propagation for approximate inference: An empirical study." Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1999.

[8] B.J. Frey. Turbo factor analysis. In Adv. Neural Information Processing Systems 12. 2000.

[9] Rusmevichientong P . and Van Roy B. An analysis of Turbo decoding with Gaussian densities. In Adv. Neural Information Processing Systems 12. 2000.

[10] Y. Weiss. Correctness of local probability propagation in graphical models with loops. Neural Computation, to appear, 2000.

[11] Weiss, Yair, and William T. Freeman. "Correctness of belief propagation in Gaussian graphical models of arbitrary topology." Neural computation 13.10 (2001): 2173-2200.

[12] Murphy, Kevin Patrick. Dynamic bayesian networks: representation, inference and learning. Diss. University of California, Berkeley, 2002.

[13] Tatikonda, Sekhar C. "Convergence of the sum-product algorithm." Information Theory Workshop, 2003. Proceedings. 2003 IEEE. IEEE, 2003.

[14] Ihler, Alexander T., W. Fisher John III, and Alan S. Willsky. "Loopy belief propagation: Convergence and effects of message errors." Journal of Machine Learning Research 6.May (2005): 905-936.

[15] Georgii, Hans-Otto. Gibbs measures and phase transitions. Vol. 9. Walter de Gruyter, 2011.

[16] T. Heskes, “On the uniqueness of Loopy Belief Propagation fixed points,” Neural Computation, vol. 16, no. 11, pp. 2379–2413, Nov. 2004.

[17] Sudderth, Erik B., et al. "Nonparametric belief propagation." Communications of the ACM 53.10 (2010): 95-103.

[18] Bounova, G., de Weck, O.L. "Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles", Phys. Rev. E 85, 016117 (2012)

Appendix

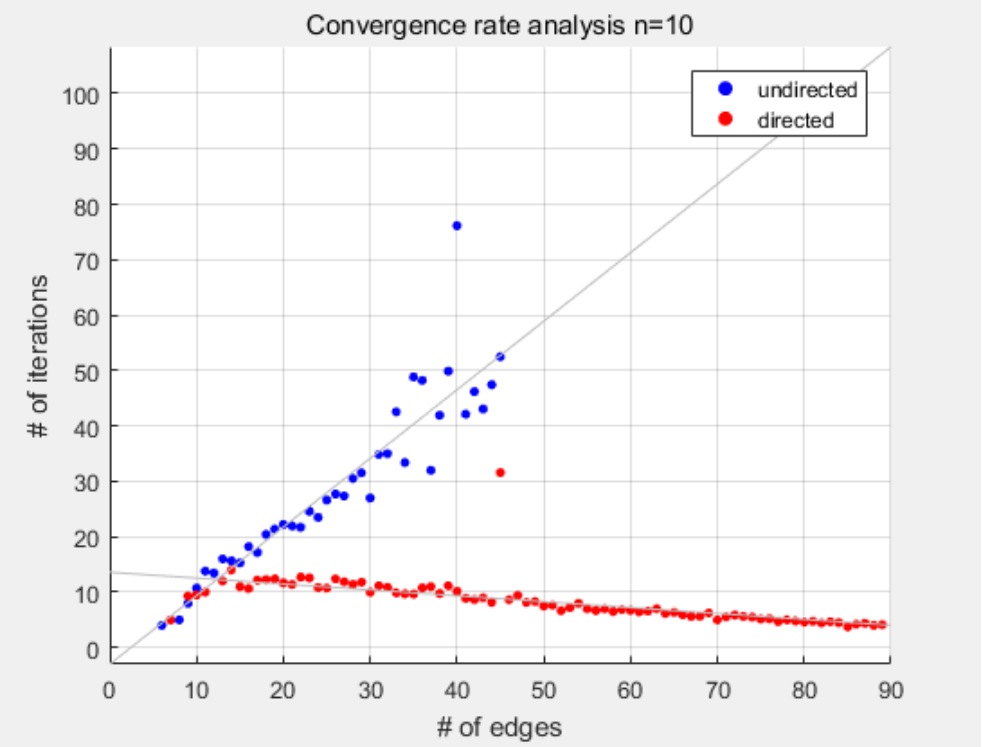

Figure 4(a). Averaged version, number of iterations vs. number of edges.

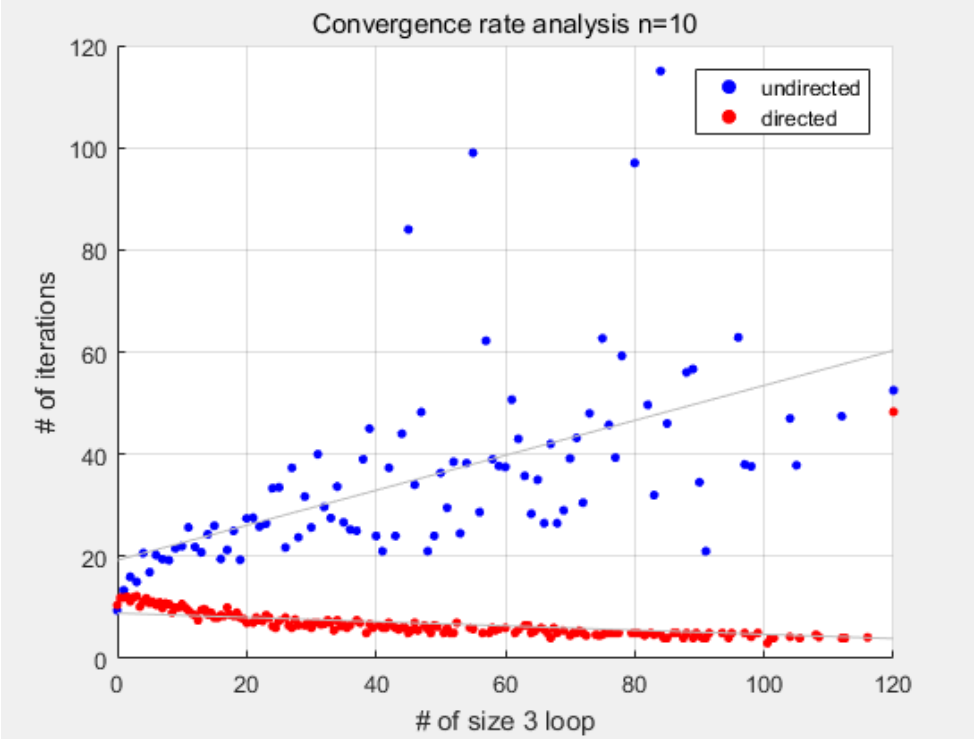

Figure 4(b). Averaged version, number of iterations vs. number of size 3 loop.

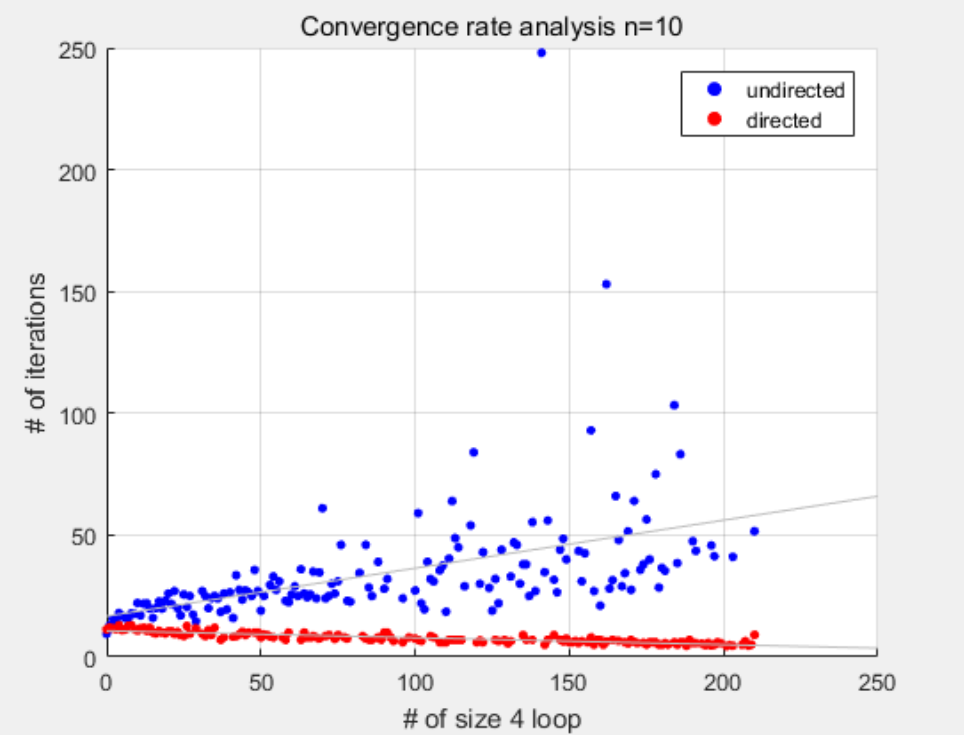

Figure 4(c). Averaged version, number of iterations vs. number of size 4 loop.

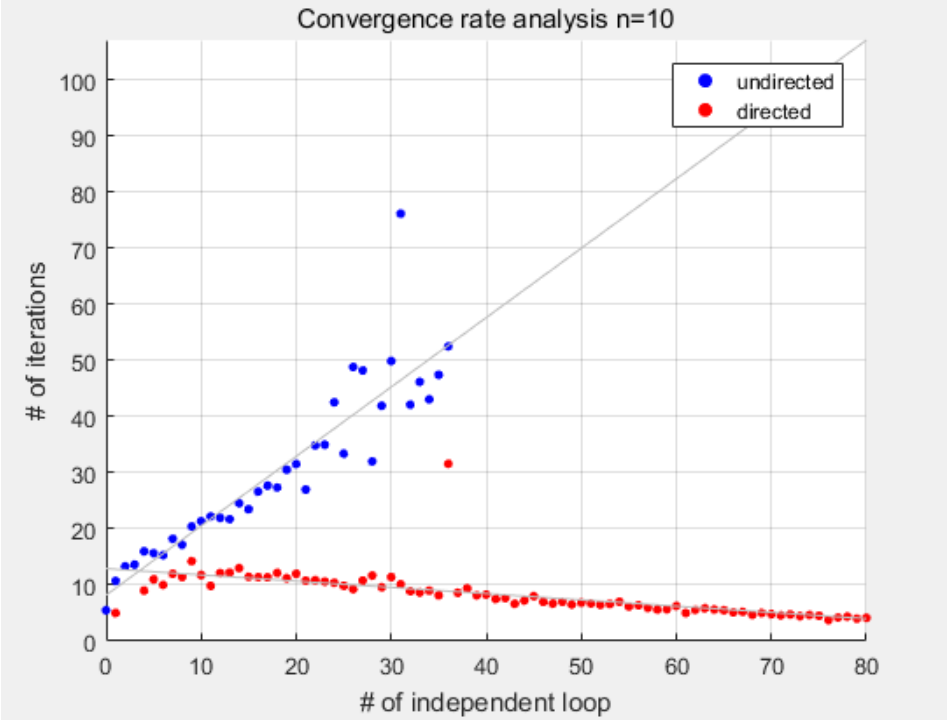

Figure 4(d). Averaged version, number of iterations vs. number of independent loop.

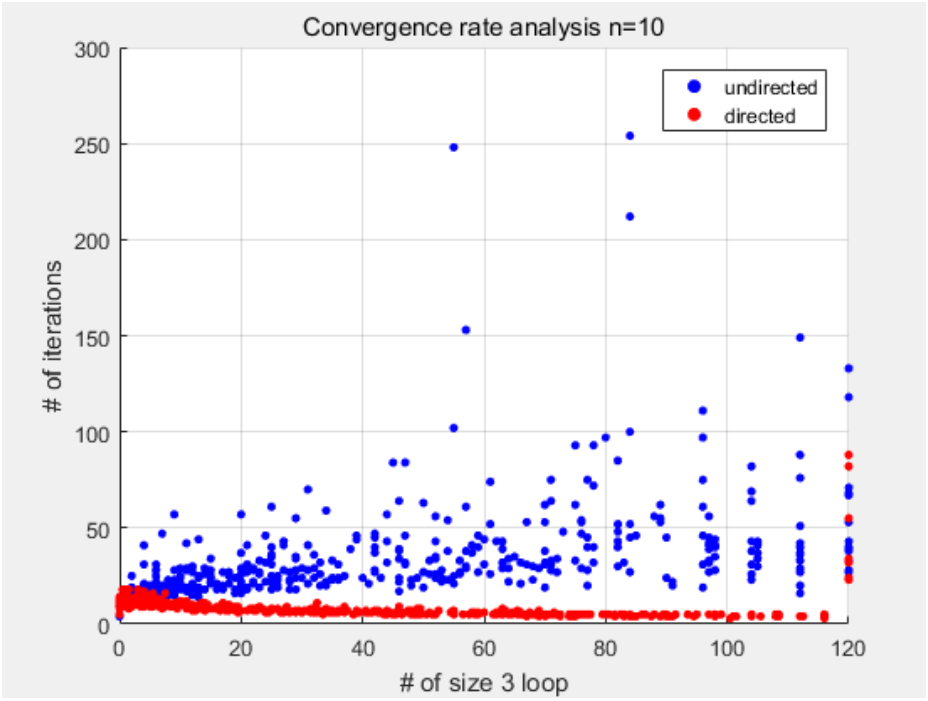

Figure 5(a). Original version, number of iterations vs. number of size 3 loop.

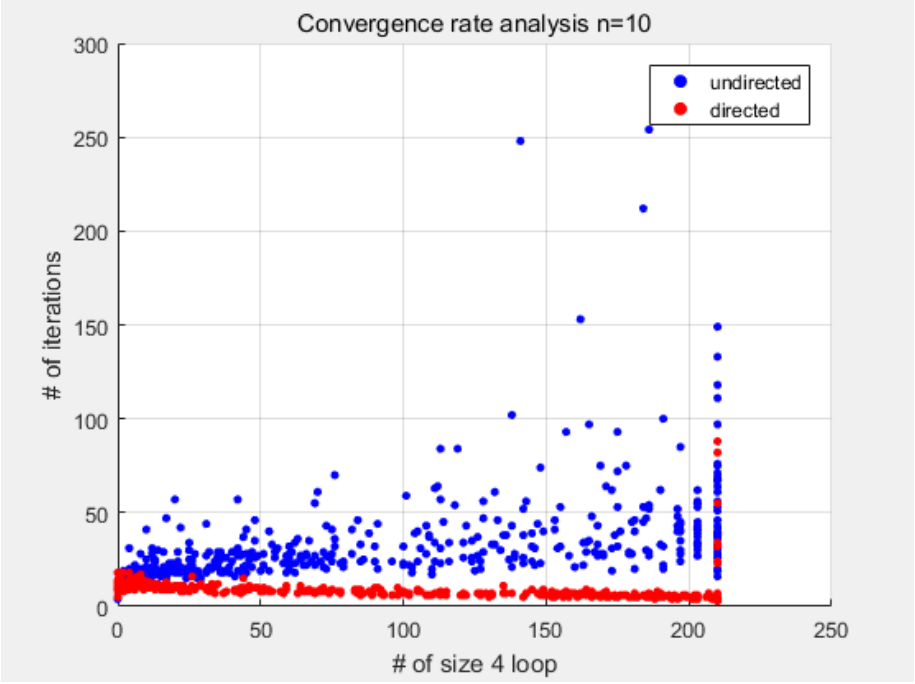

Figure 5(b). Original version, number of iterations vs. number of size 4 loop.

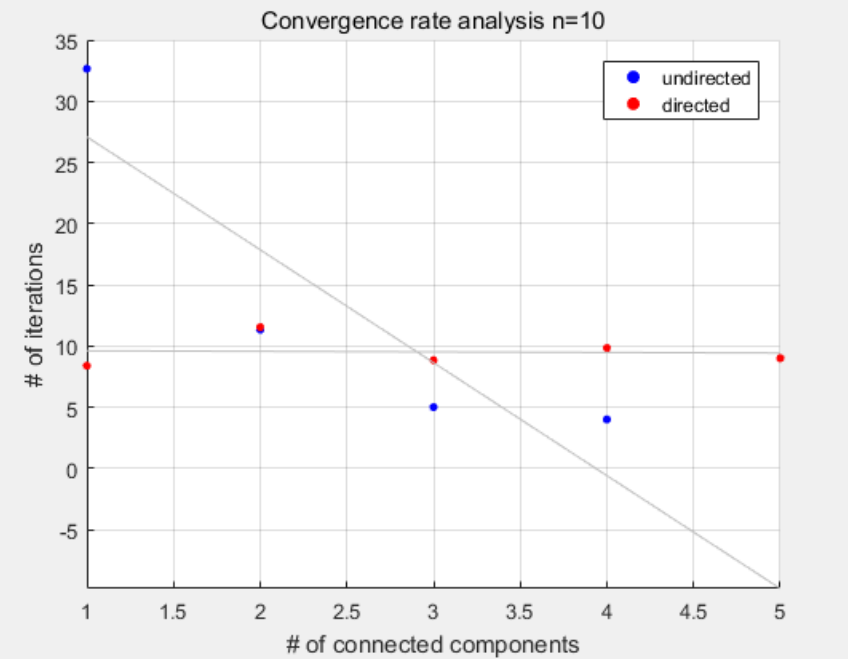

Figure 6(a). Averaged version, number of iterations vs. number of connected components.

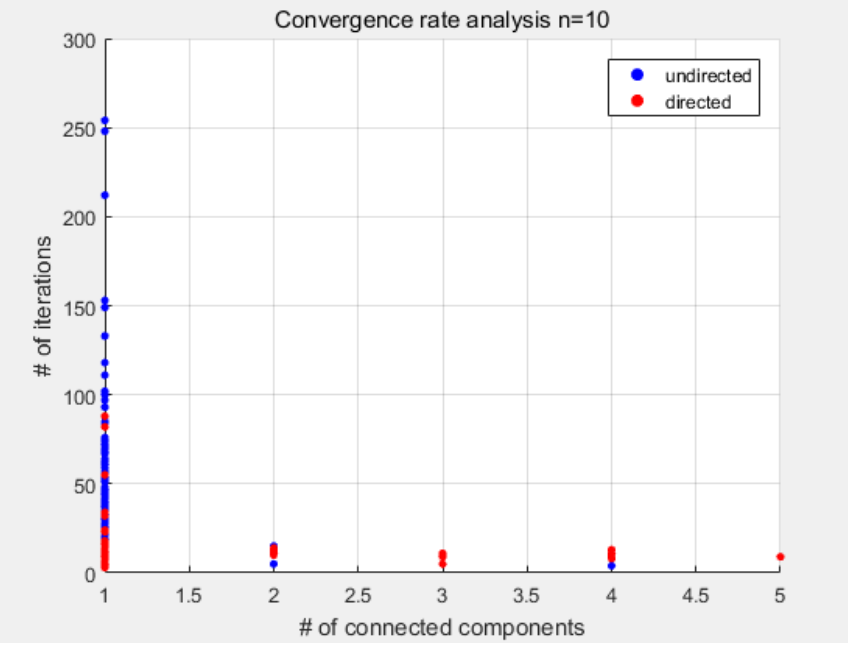

Figure 6(b). Original version, number of iterations vs. number of connected components.

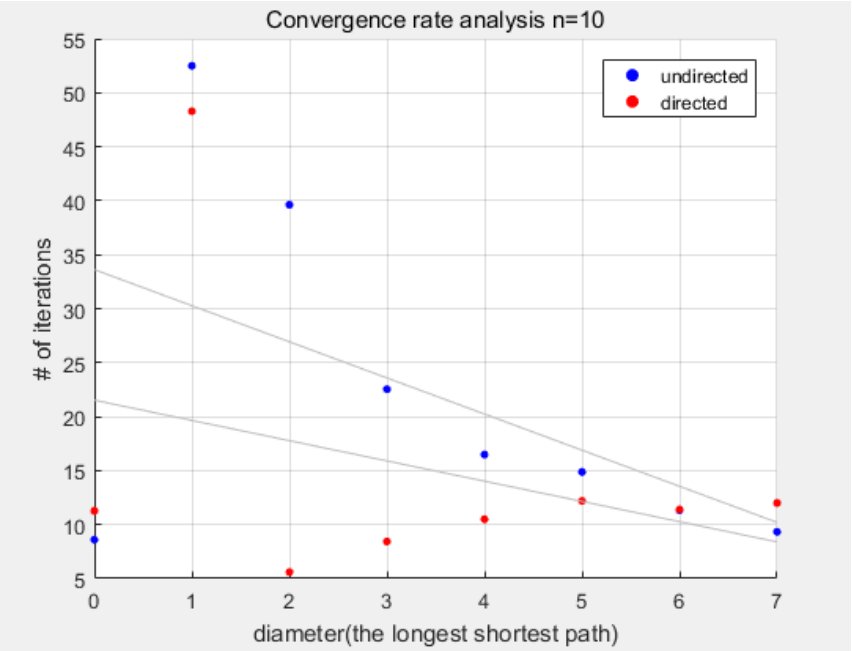

Figure 7(a). Averaged version, number of iterations vs. the longest length of the shortest path.

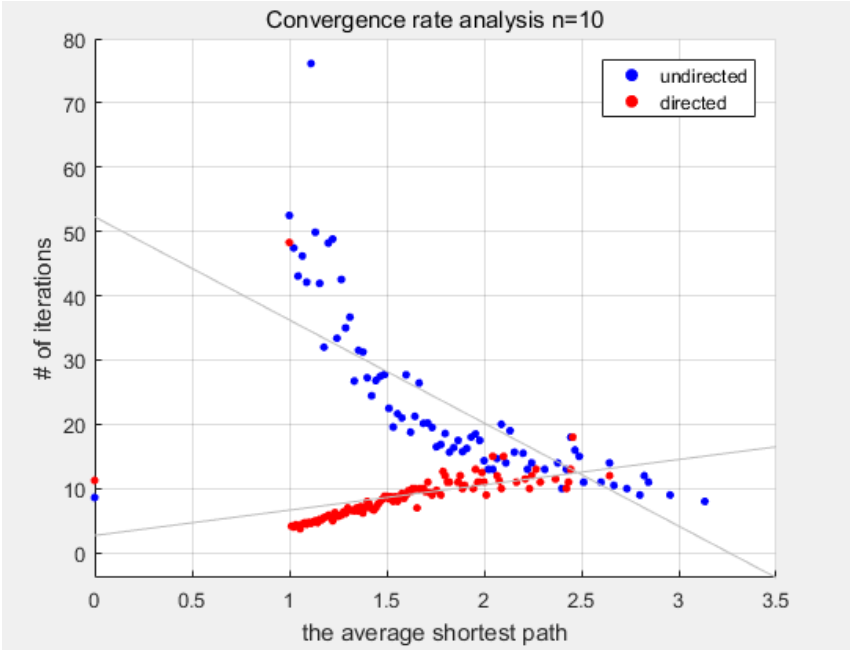

Figure 7(b). Averaged version, number of iterations vs. the average length of the shortest path.

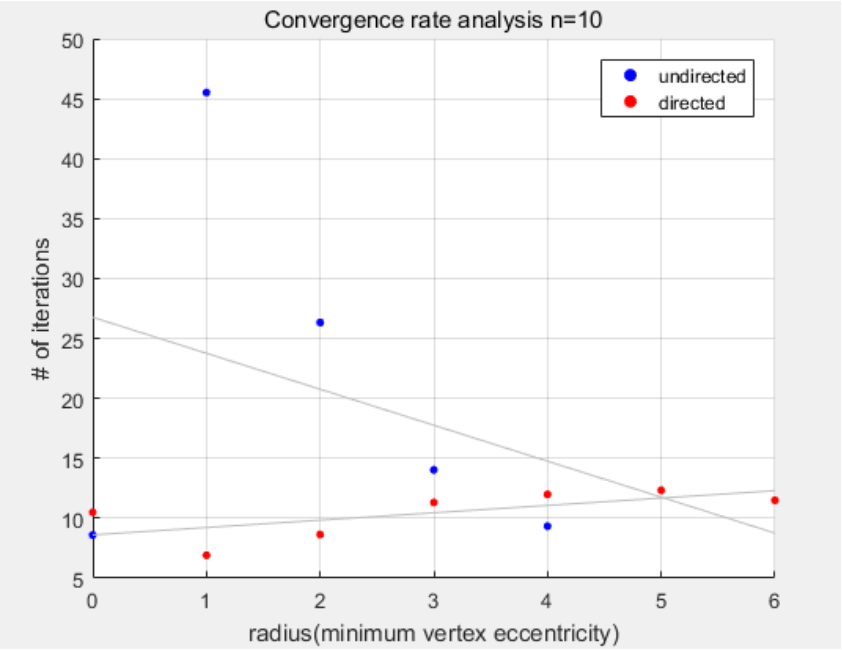

Figure 7(c). Averaged version, number of iterations vs. the minimum length of the shortest path.

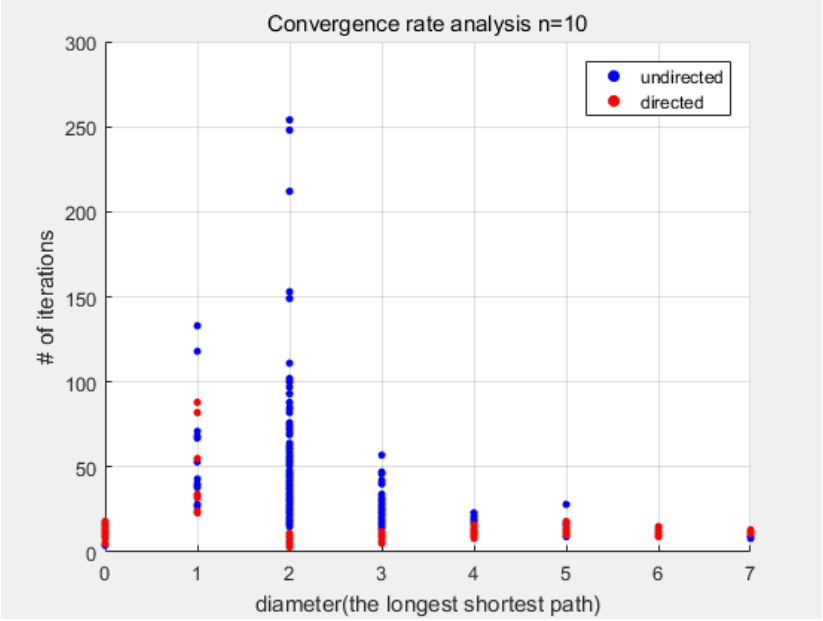

Figure 8(a). Original version, number of iterations vs. the longest length of the shortest path.

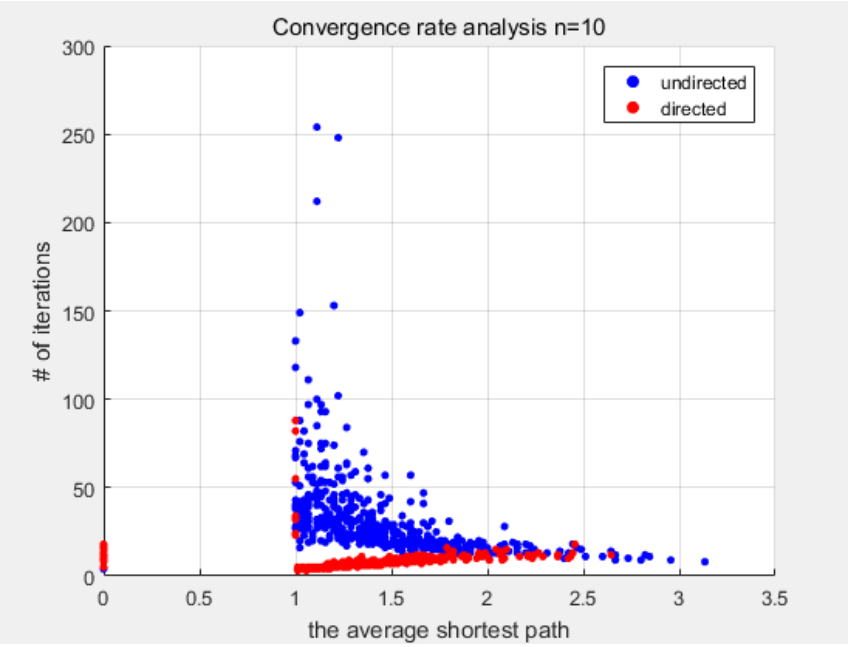

Figure 8(b). Original version, number of iterations vs. the average length of the shortest path.

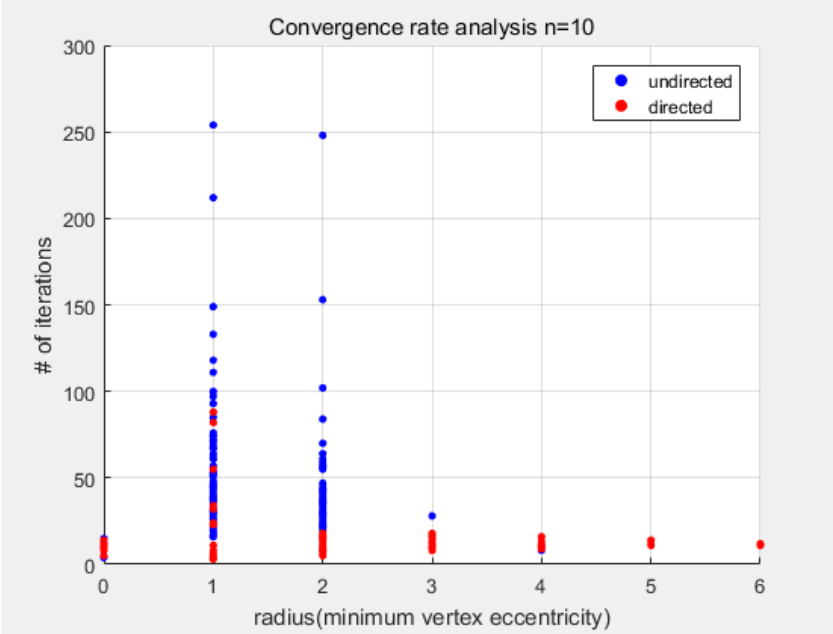

Figure 8(c). Original version, number of iterations vs. the minimum length of the shortest path.

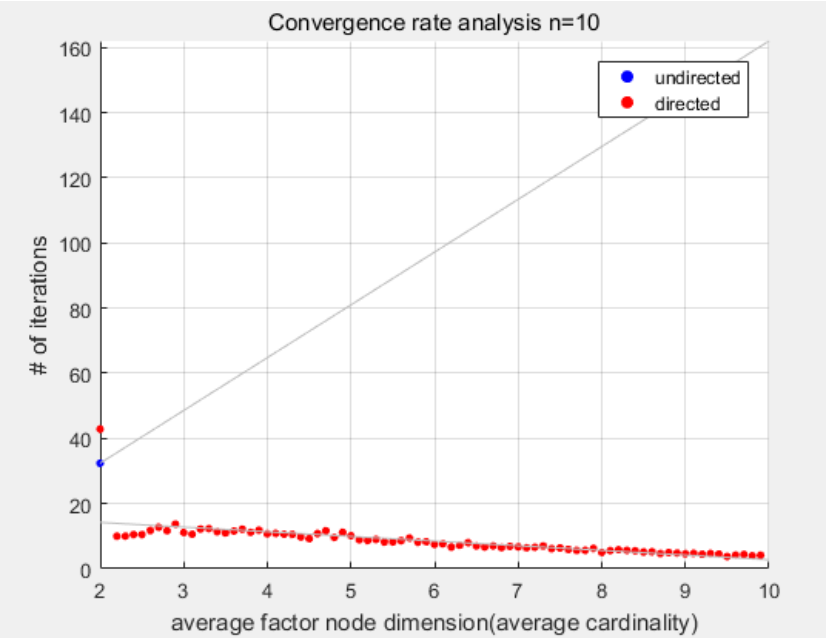

Figure 9(a). Averaged version, number of iterations vs. average cardinality.

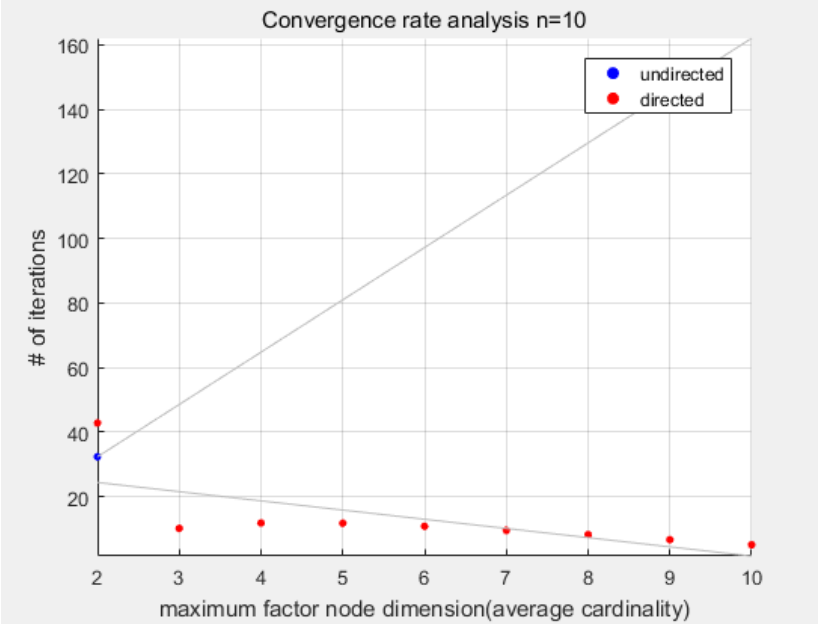

Figure 9(b). Averaged version, number of iterations vs. maximum cardinality.

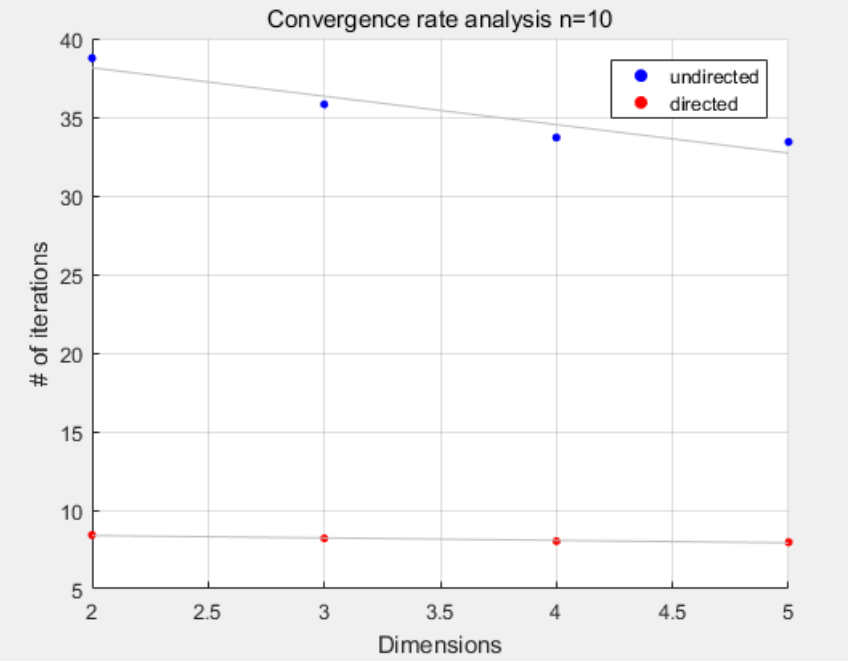

Figure 10(a). Averaged version, number of iterations vs. variable node dimension.

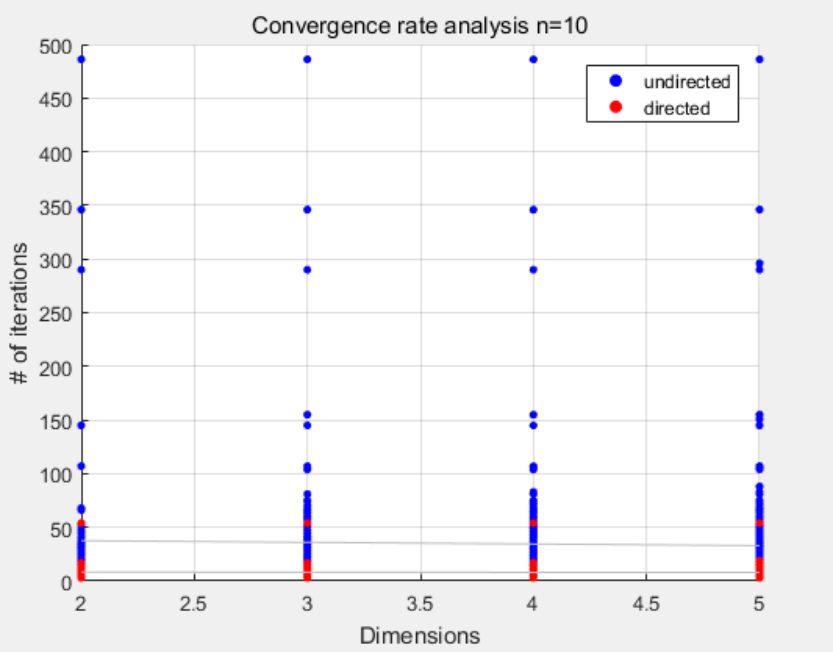

Figure 10(b). Original version, number of iterations vs. variable node dimension.

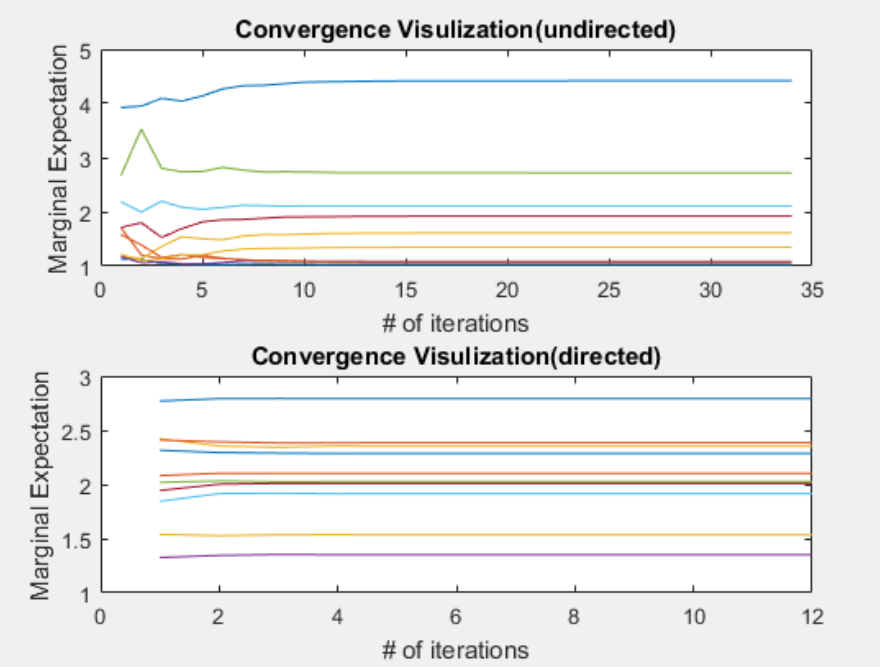

Figure 11(a). Marginal Expectation of all variable nodes across the convergence process.

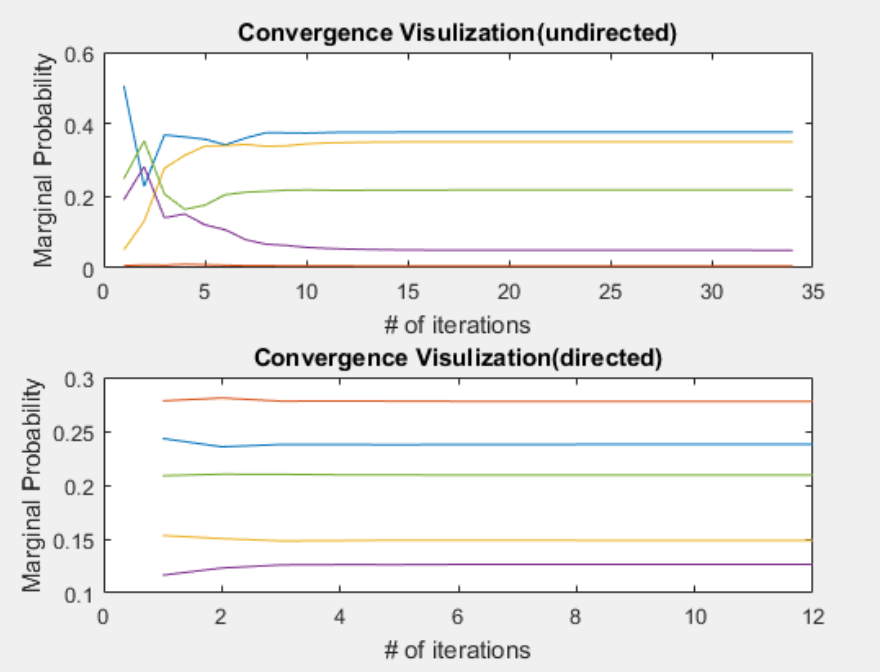

Figure 11(b). Marginal Probability of one variable node across the convergence process.