Graz University of Technology

Master Thesis

Message Scheduling in Loopy Belief Propagation

conducted at the
Signal Processing and Speech Communications Laboratory
Graz University of Technology, Austria

by
Michael Rath

Supervisor:
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf

Graz, March 10, 2015

ABSTRACT

A popular method to efficiently perform probabilistic inference in graphical models is belief propagation, which is also called message passing. Thereby, messages that represent local beliefs are introduced into the graph and have to be periodically updated until convergence. When applied to graphs containing loops, it is called Loopy Belief Propagation (LBP) and only approximate inference is possible, whereas convergence is not guaranteed. By introducing methods of message scheduling that regulate the order in which messages are updated, the convergence rate can be increased significantly while still providing good estimates for the marginals. The most efficient scheduling method is that of Residual Belief Propagation (RBP), where the message that changes the most is selected for update. We apply RBP to estimate the marginals of difficult grid graphs with Ising factors and analyze the non-convergent cases. We observe the problem of message oscillation, where the same small set of messages repeatedly gets selected for update. We introduce and evaluate two different methods to suppress / avoid the oscillations. The first method called RBPnoise injects noise into the message update when oscillation is detected. The second method called RBPnUp takes the number of message updates into account when choosing the next message for update. We show that both methods increase the convergence rate while also giving better marginals than standard RBP. Thereby, RBPnoise improves the convergence rate the most, while RBPnUp gives the best marginals.

KURZFASSUNG

Eine gängige Methode um effizient probabilistische Inferenz in grafischen Modellen durchzuführen ist Belief Propagation, das auch unter dem Begriff Message Passing bekannt ist. Dabei werden Nachrichten, die lokale Beliefs repräsentieren, in den Graphen eingeführt und diese müssen wiederholt aktualisiert werden bis sie konvergieren. Wenn diese Methode auf Graphen mit Loops angewandt wird, wird es auch Loopy Belief Propagation (LBP) genannt, wobei nur approximierte Inferenz möglich ist und die Konvergenz nicht garantiert ist. Durch das Einführen von Message Scheduling Methoden, welche die Reihenfolge in der die Nachrichten aktualisiert werden festlegt, kann die Konvergenzrate entscheidend erhöht werden, wobei trotzdem gute Werte für die geschätzen Randwahrscheinlichkeiten erreicht werden. Die derzeit effizienteste Scheduling Methode ist Residual Belief Propagation (RBP), bei welcher immer die Nachricht mit der größten Änderung zur Aktualisierung gewählt wird. Wir verwenden RBP um die Randwahrscheinlichkeiten von schwierigen Grid-Graphen mit Ising Faktoren zu bestimmen und analysieren die Fälle wo keine Konvergenz erreicht werden kann. Dabei beobachten und präsentieren wir das auftretende Problem der oszillierenden Nachrichten, wobei ein bestimmtes kleines Set von den selben Nachrichten wiederholt zur Aktualisierung bestimmt wird. Wir stellen zwei Methoden zur Unterdrückung / Vermeidung von Oszillationen vor und evaluieren diese. Die erste Methode wird mit RBPnoise bezeichnet. Dabei wird, wenn Oszillationen erkannt werden, ein kleiner Zufallswert, hier "Noise" genannt, zum neuen Wert der Nachricht bei der Aktualisierung hinzugefügt. Die zweite Methode wird mit RBPnUp bezeichnet. Dabei wird, bei der Auswahl der zu aktualisierenden Nachricht, die Anzahl der Aktualisierungen mitberücksichtigt. Wir zeigen in Experimenten, dass beide Methoden die Konvergenzrate erhöhen und gleichzeitg auch bessere Werte für die Randwahrscheinlichkeiten geben als standard RBP. Dabei erhöht RBPnoise the Konvergenzrate am meisten, während RBPnUp die besten Randwahrscheinlichkeiten erreicht.

Acknowledgments

I would like to thank my supervisor Franz Pernkopf for his help and support throughout the "lifespan" of this thesis. I also thank him for giving me the opportunity to temporarily work as an "Studienassistent" for the SPSC institute.
I thank Sebastian Tschiatschek for the help in finding a way to improve the existing BP methods.
I thank Christian Knoll for the corrections and help with the refinement of the thesis.
I thank my parents for all their support during the (numerous) years of my studies.
And last but not least, I thank all the members and friends from our beloved "SPSC Wohngemeinschaft" for making the times at university (even) more fun. This includes Linda Lüchtrath, Mario Watanabe, Georg Kapeller and Tobias Schrank.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

date	(signature)

Contents

1 Introduction

Probabilistic graphical models are used in machine learning as an descriptive mechanism to model joint distributions over random variables and their connections to each other. Representations such as *factor graphs* [1,2] allow to greatly simplify reasoning in graphical models. This is known as *probabilistic inference* and opens the door to a powerful inference method which is known as *belief propagation* (BP) introduced by Pearl [3].
BP uses the factors of a joint probability distribution and their dependencies on variables to define *messages* that are sent between variable and factor nodes. A message over a connection of two nodes represents the belief the sender node has about the state of the receiver node. *Sending a message* means to recompute its value using all messages the sender node receives and this is also called the *message update*. All nodes send messages until the beliefs converge, hence this method is also called *message passing*. The messages have to be sent until they converge to their fixed points. Applying message passing to tree structured graphs, exact inference can be performed with a low computational cost.
However, many graphs that represent a domain of the real world may have an arbitrary structure, including loops. Applying BP to a graph with loops is known as *loopy belief propagation* (LBP). In general, it is only possible to perform approximate inference using LBP and convergence is not guaranteed [4,5]. Yet, Weiss showed that LBP converges for the single loop case [6] and in multiple publications it was shown empirically that LBP can still give good results when applied to graphs with an even more complicated structure, containing many loops [7–9].
There are various approaches that aim to correct for the presence of loops such as using *cavity distributions* as shown by Mooij et al. [10, 11] or the *truncated loop series BP* introduced by Gómez et al. [12]. There are also many publications relating the fixed points of BP to extrema of approximate free energy functions from statistical physics [13, 14]. Yedidia et al. [15] showed that the extrema of Bethe free energy approximations correspond to the fixed points of LBP. Using the more precise Kikuchi free energy function, they developed *Generalized BP* (GBP) [16], which significantly improves the convergence rate compared to standard LBP. Yuille [17] provided the alternative method of *CCCP algorithms*, that is applied to Bethe and Kikuchi free energies to obtain the extrema, resulting in slightly better results than those found by GBP.
Another very simple, yet effective, approach to improve the performance of LBP, can be achieved by introducing methods of *message scheduling*, that regulate the order in which messages are sent. Elidan et al. [18] presented a method using a dynamic intelligent message schedule called *residual belief propagation* (RBP) that chooses the message that changes the most for update. This leads to a remarkable improvement in terms of the convergence rate and speed while still

providing good marginals. Sutton and McCallum [19] directly improved upon RBP in terms of the convergence speed by approximating the message updates.
In this thesis, we apply RBP to difficult graphs containing many loops and analyze the non-convergent cases. In these cases we observe the problem of *message oscillation*, where a small set of messages repeatedly gets selected for update because they periodically assume the same message values. Our assumption is that breaking these oscillations leads to a better performance. Therefore we introduce and evaluate two different methods. The first method called *RBPnoise* detects message oscillations and injects noise into the message update once an oscillation has been detected. The second method called *RBPnUp* takes the number of individual message updates into account when choosing the next message for update. These methods are used to perform approximate inference in graphs that are standard benchmarks for BP methods. In particular, grid graphs with Ising factors are used. We show that both methods increase the convergence rate while also giving better marginals than standard RBP. Thereby, RBPnoise improves the convergence rate the most, while RBPnUp gives the best marginals.

Organization of the Thesis

The thesis is organized the following way: Chapter 2 provides the theoretical background and presents the used notations. A short overview of concepts from probability theory and graph theory is given. Then representations of probabilistic graphical models, namely Bayesian networks and factor graphs are presented. Probabilistic inference leading to (loopy) belief propagation and the respective formulas are introduced. In Chapter 3 the concepts of Message Scheduling are presented by providing a description of *synchronous belief propagation* (SBP), *asynchronous belief propagation* (ABP) and RBP. This Chapter is concluded by presenting enhancement methods for residual belief propagation, i.e. noisy and weighted RBP. Chapter 4 shows experiments providing empirical results comparing general SBP and ABP with RBP and the enhanced RBP versions. Two experimental setups were used, adapted from other scientific publications. We evaluate the performance of the methods and provide a detailed discussion. Finally, Chapter 5 gives a summary of the methods and results. Conclusions and an overview of possible future work as well as connections to related research are presented.

2 Probabilistic Graphical Models

In this chapter, the basics of probability theory and graphical models are described and the respective notations that will be used throughout the thesis are introduced. This will then be followed by a description of factor graphs and the application of loopy belief propagation.

2.1 Probability Theory

This section provides an overview of basics of probability theory needed to understand the later described concepts. The notations are based on [20] and [21] and for a more detailed description the reader is referred to these publications.

2.1.1 Probability Distribution

Let Ω be an *outcome space*, that is, a non-empty set containing elements representing possible events (e.g., $\Omega = \{1, 2, 3, 4, 5, 6\}$ for a game with six sided dices). Let S be an *event space* containing elements that are all possible subsets of Ω (e.g., $S = \{2, 6\}$ is the event that a 2 or a 6 have been rolled in the aforementioned dice game).
A *probability distribution* over (Ω, S) is then a mapping $P : S \mapsto \mathbb{R}$ that satisfies the conditions

- $P(\alpha) \geq 0$ for all $\alpha \in S$,
- $P(\Omega) = 1$.
- If $\alpha, \beta \in S$ and $\alpha \cap \beta = \emptyset$, then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$.

These conditions are also known as *Kolmogorov's axioms* III-V [22]. The triplet (Ω, S, P) is called a *probability space*.

2.1.2 Random Variable

To add structure to the outcome space and thus the probability space, *random variables* that describe certain attributes or domains of the outcome space are introduced. Mathematically, this means that the random variable X is a function over the outcome space and maps it to a

measurable space $\mathbb{S}$, or more formally $X : \Omega \mapsto \mathbb{S}$. $\mathbb{S}$ can be a small countable set of numbers (*discrete random variable*) or subset of $\mathbb{R}$ containing an infinite range (*continuous random variable*).
E.g., when dealing with medical records, one is presented with patients of varying age, which is one attribute of the outcome space of their health state. This attribute could be captured by a single random variable, that is called say A, that can assume all different age values.
Throughout this thesis, uppercase roman letters (X, Y, Z) will be used to denote random variables. For variable assignments, lowercase letters are used to refer to a generic value, e.g., $P(X = x)$ for all $x \in Val(X)$. For sets of random variables calligraphic letters are used, e.g., $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$. Alternatively, when referring to sets of random variables represented by graphs, vector notation with boldface letters is used, e.g., $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ for variables and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ for their assignments. Also, in several cases, $P(x)$ will be used as a shorthand notation for $P(X = x)$.

2.1.3 Marginal and Joint Probability Distributions

With the definition of random variable X, one can consider the distribution over all events that can be described solely using X. This distribution is called the *marginal distribution* over X and denoted by $P(X)$. Evidently, marginal distributions have to satisfy the properties defined in Section 2.1.1, the only change being the restriction to a subset of S that can be described by the respective random variable. E.g., in the medical records example, the marginal distribution over the *age* random variable $P(A)$, illustrates for all medical records the probability of a certain age to occur.

When looking upon events captured by several random variables, one has to consider the *joint distribution* over these variables. More specifically, one considers the set of variables $\mathcal{X} = \{X_1, \ldots, X_N\}$ with the distribution $P(X_1, \ldots, X_N)$ that assigns probabilities to events that are specified in terms of these random variables. E.g., in the medical example, one can add various variables to more specifically describe patients and their medical history, such as a gender variable, or binary variables describing symptoms and overcome diseases.
The marginals can be obtained from a joint distribution by summing out all other variables i.e.,

$$P(x_i) = \sum_{\sim x_i} P(x_1, \ldots, x_n) \tag{2.1}$$

$$= \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} P(x_1, \ldots, x_n). \tag{2.2}$$

Here, the summations mean, that all possible values of the respective random variables have to be captured. For continuous random variables, the summation has to be replaced by integration in a straightforward matter.
Throughout the thesis, when talking about random variables X_i, or $P(x_i)$ in the shorthand notation, unless stated otherwise, the variable is assumed to be discrete.

2.1.4 Conditional Distribution

A further combination of multiple random variables to a probability distribution is the *conditional probability distribution* (CPD) over a random variable, given the observation or knowledge of a certain assignment of a different random variable, which is denoted by $P(x|y)$. A CPD over a single variable is different than the marginal for this variable, as can be seen by using the medical example again. E.g., the probability of having a certain disease might drastically change, having information about the age of the patient.
An important property combining the regarded types of distributions is the chain rule defined

as

$$P(x, y) = P(x|y)P(y). \tag{2.3}$$

Another useful formula called the *Bayes' rule* can be obtained by using the chain rule on a different ordering of the variables from the joint distribution given by

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.4}$$

$$= \frac{P(y|x)P(x)}{\sum_x P(y|x)P(x)}. \tag{2.5}$$

2.1.5 Statistical Independence

In certain situations, having knowledge of the assignment of a certain random variable might not change the distribution of a different random variable. In this case those variables are called *statistically independent* denoted by $(x \perp y)$ and they have to satisfy

$$P(x, y) = P(x)P(y), \tag{2.6}$$

$$P(x|y) = P(x). \tag{2.7}$$

In practice, having complete statistical independence is a rare occurrence. A more common and thus useful property is independence between variables given a different variable. This is called *conditional statistical independence*, denoted by $(x \perp y \,|\, z)$ and has to satisfy

$$P(x, y|z) = P(x|z)P(y|z), \tag{2.8}$$

$$P(x|y, z) = P(x|z). \tag{2.9}$$

In other words, by learning what value Z assumes, any information about Y will not change the probability of X. For the first case of independence, not having a condition on any other variable, one could also write $(x \perp y \,|\, \emptyset)$, which is called *marginally independent*.
The same rules apply to sets of random variables and the independence statement is a universal quantification over all possible values of the random variables.

For properties of conditional independence such as decomposition, weak union or contraction, the reader is referred to [20] or [23].

2.2 Basics of Graph Theory

In this section, an overview over concepts from graph theory is given and many terms that will be used throughout the thesis are introduced.

2.2.1 Basic Graph Structure

A graph is a data structure $\mathcal{G}$ consisting of a set of *vertices*, also more commonly denoted as *nodes* and a set of *edges*. Throughout this thesis, the set of nodes will be denoted by $\mathbf{X} = \{X_1, \ldots, X_N\}$ to keep the analogy with the random variables. Any pair of nodes X_i and X_j, $i \neq j$ can be connected by a *directed edge* $X_i \to X_j$, which represents an asymmetric relationship between the nodes, or an *undirected edge* $X_i - X_j$, which represents a symmetric relationship between the

nodes, altogether forming a set of edges $\mathcal{E}$. Two nodes having a connection in general, without specifying the type, will be denoted by $X_i \rightleftharpoons X_j$. There can only be one type of edge between two nodes, there can however be multiple types of edges within a graph. Depending on what types of edges appear in a graph, the graph is either called directed, undirected or mixed.
Graphically, nodes are represented by circles or squares, undirected edges by lines and directed edges by arrows. For both types, one specific graph type will be discussed and presented in the next sections.

2.2.2 Node Neighborhood

Nodes connected by undirected edges are called *neighbors*, where all nodes connected to a single node X_i form the set $\mathbf{Nb}_{\mathcal{G}}(X_i)$. The source-nodes of directed edges are called *parents* and destination-nodes are called *children*, thus forming the sets $\mathbf{Pa}_{\mathcal{G}}(X_j)$ for all parents of X_j and $\mathbf{Ch}_{\mathcal{G}}(X_i)$ for all children of X_i.

2.2.3 Long-range connections

Long-range connections between nodes form *paths* and *trails*. A trail consists of uninterrupted connections between multiple variables, regardless of the edge type. That means that $X_1 \dots X_k$ form a trail if, for every $i = 1, \dots, k-1$, there is a connection $X_i \rightleftharpoons X_{i+1}$. A path is the same as a trail, albeit if there are directed edges, they must have one consistent direction. This means that $X_1 \dots X_k$ form a path if, for every $i = 1, \dots, k-1$, there is either a $X_i - X_{i+1}$ or a $X_i \to X_{i+1}$ connection.

2.2.4 Node Ancestry

Having defined these long-range connections allows the introduction of many useful terms. For any two nodes X_i and X_j, if there exists a directed path starting at X_i and ending at X_j, then X_i is an *ancestor* of X_j and X_j is a *descendant* of X_i. Accordingly, all ancestors of X_j form the set $\mathbf{Anc}_{\mathcal{G}}(X_j)$ and all descendants of X_i form the set $\mathbf{Desc}_{\mathcal{G}}(X_j)$.

2.2.5 Cycles

A *directed cycle* is a directed path that starts and ends at the same node. A directed graph that contains no cycles is called a *directed acyclic graph* (DAG) because there exist undirected cyclic graphs.

2.3 Bayesian Networks

Probably the most intuitive of the probabilistic graphical models are *Bayesian Networks* (BN). In this section BNs are defined and discussed.

2.3.1 Definition

A BN is a DAG $\mathcal{G}$. It consists of nodes and arrows between them, such that there is no directed cycle. Thereby, the nodes represent discrete random variables from a joint probability distribution and the arrows represent the statistical relationship between the variables, where

they are associated with a conditional probability distribution. The network can be viewed as a compact representation of

1. a factorization of a joint distribution,
2. a set of conditional independence assumptions about a joint distribution [20].

The former view is a good tool to construct the graph:
Given is a factorization of a joint probability distribution into CPDs $P_{X_1}, \ldots, P_{X_N}$. To construct a graphical model $\mathcal{G}$ as a BN representing this distribution, the following rule has to be applied: *For each node X_i described by the CPD $P_{X_i}(X_i \mid \mathbf{Pa}(X_i))$, one has to add directed edges from each of its parents. This results in adding all $X_n \to X_i$, $X_n \in \mathbf{Pa}(X_i)$.*
On the other side, given a BN, one can immediately obtain all contained factors. Each one being the CPD over a node variable, given all variables represented by its parent nodes. This results in the definition of the joint probability distribution

$$P(X_1, \ldots X_N) = \prod_{i=1}^{N} P_{X_i}(X_i \mid \mathbf{Pa}(x_i)). \tag{2.10}$$

2.3.2 Example

A sample graph for a simple factorization, including its simplified factorization formula according to Eq. (2.10), is depicted in Fig. 2.1.

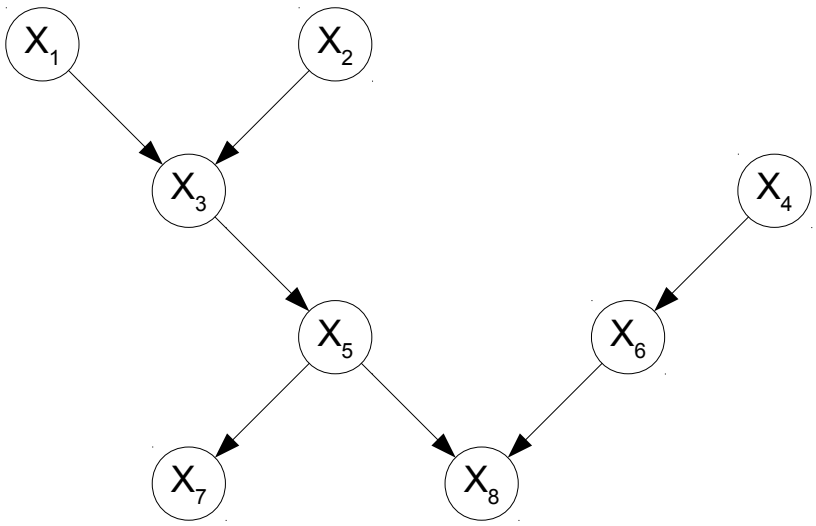


Figure 2.1: Example for BN with the represented joint distribution.

2.3.3 Captured Independence Properties

As mentioned, BNs can also be viewed as a representation for conditional independence assumptions over a joint distribution. For this, the rule that applies is:

Each variable is independent of all other variables in its set of non-descendants given its parents. More formally that means,

$$X_i \perp \mathbf{NonDesc}_{\mathcal{G}}(X_i) \,|\, \mathbf{Pa}_{\mathcal{G}}(X_i) \quad \forall i. \tag{2.11}$$

For a proof the reader is referred to [21]. It should also be noted that these local independence assumptions imply the aforementioned factorization properties. Hence, for the second way of constructing a BN from scratch, one can start with a completely connected BN where no independencies are captured and remove edges to include independencies. It is important to *not* include any independencies that are inconsistent. Leaving some out is not a problem, since it would only result in redundancy. It can still represent the joint distribution, albeit not the exact same factorization. Constructing a BN in this way is very intuitive, since one can see each node as actors or events in the real world and think of their influence to each other in an explicit, descriptive manner.
Since, for the concepts described in this thesis, a given factorization of a joint probability distribution is assumed to be given, the reader is referred to [20] for a detailed description of techniques to effectively construct BNs that provide good factorizations.

2.4 Factor Graphs

The graphical model used in this thesis is that of a *factor graph* (FG). This model directly represents a given factorization of a joint probability distribution. The models and resulting formulas are adaptations from [1] and another good presentation is given in [2].

2.4.1 Definition

In a similar fashion as in BNs, one has given a factorization of a joint probability distribution, that is

$$P(X_1, \ldots, X_N) = \frac{1}{Z} \prod_{d \in \mathcal{D}} F_d(\mathbf{X_d}), \tag{2.12}$$

where $\mathcal{D}$ is a discrete index set, each $F_d(\mathbf{X_d})$ is a factor over a subset $\mathbf{X_d} \subseteq \{X_1, \ldots X_N\}$. Z is the normalization constant, obtained by summing over all possible assignments for $\{X_1, \ldots X_N\}$.
A FG is then a bipartite graph, that is, a graph $\mathcal{F}$ of two different types of nodes: one for each random variable X_i of the set $\{X_1, \ldots, X_N\}$ depicted as circles containing the labeling of the variable and one for each factor F_d of the set $\{F_1, \ldots, F_D\}$ depicted as squares and, in this thesis, also containing the labeling of the factor. These form two disjoint subsets, such that there can only be edges between nodes from each subset, never from in-between a subset.
One might define the set $\mathbf{X_d}$ via $\mathbf{Nb}_{\mathcal{F}}(F_d)$, making use of the notation for graphical models. This allows, for a given FG, to easily obtain the factorization formula.

2.4.2 Example

A sample graph for the same factorization used in the BN on Fig. 2.1, is depicted in Fig. 2.2.

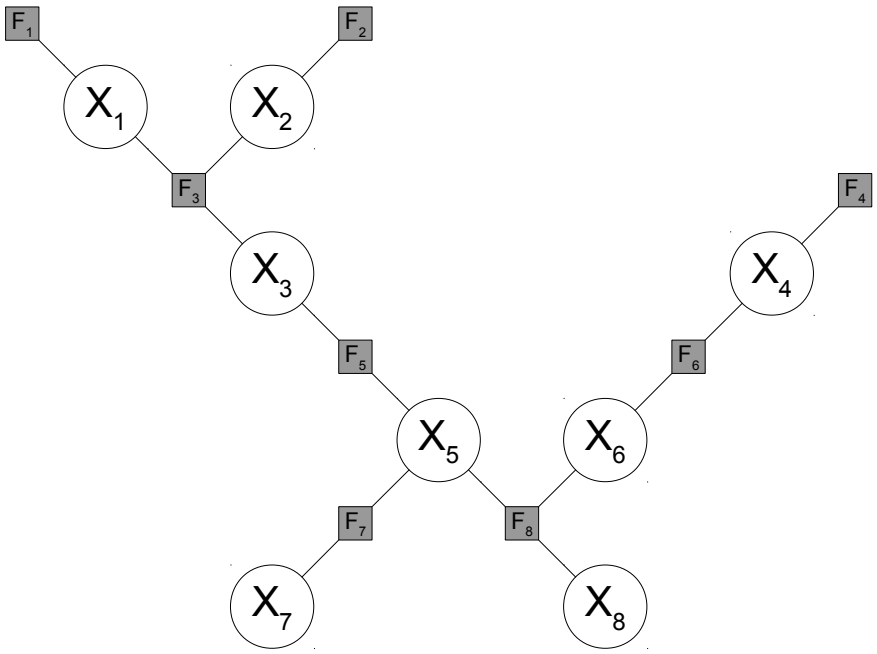


Figure 2.2: Example for FG of the joint distribution from Fig. 2.1.

2.5 Probabilistic Inference

This section explains *Probabilistic Inference*, where one uses a joint probability distribution over random variables representing a certain domain, possibly from the real world, to answer queries of interest.

2.5.1 Probability Queries

In order to properly formulate a *probability query*, one has to partition the set of all random variables $\mathbf{X}$ into the disjoint subsets $\mathbf{Q}$, $\mathbf{O}$ and $\mathbf{H}$.

- $\mathbf{Q}$ represents variables of interest, called *query variables*, that is, variables that represent the unknown part of the considered domain that one wants to know more about.
- $\mathbf{O}$ represents known variables, called *observed variables*, which represent the part of the considered domain where one is aware of its state.
- $\mathbf{H}$ represents latent variables, called *hidden variables*, which are variables representing a unknown part of the considered domain as well, however the state of it is irrelevant or not desired to be known.

The task of a probabilistic inference is then to compute

$$P(\mathbf{Q} \mid \mathbf{O} = \mathbf{o}). \tag{2.13}$$

This is also called the *marginalization query*, since it results in the marginal over $\mathbf{Q}$, in the distribution one obtains by conditioning on $\mathbf{o}$ and it is also called the *posterior distribution*.

Applying the chain rule given by Eq. (2.3) to Eq. (2.13) one obtains

$$P(\mathbf{Q} \mid \mathbf{o}) = \frac{P(\mathbf{Q}, \mathbf{o})}{P(\mathbf{o})}. \tag{2.14}$$

Each term of this fraction can be calculated by marginalization by applying Eq. (2.1), which results in

$$P(\mathbf{Q}, \mathbf{o}) = \sum_{\mathbf{h}} P(\mathbf{Q}, \mathbf{o}, \mathbf{h}), \tag{2.15}$$

$$P(\mathbf{o}) = \sum_{\mathbf{q}} P(\mathbf{q}, \mathbf{o}). \tag{2.16}$$

Since the second equation marginalizes over the distribution obtained by the first equation, one may rewrite it as

$$P(\mathbf{o}) = \sum_{\mathbf{q}} \sum_{\mathbf{h}} P(\mathbf{q}, \mathbf{o}, \mathbf{h}). \tag{2.17}$$

On the computational side of things, this is equivalent to the marginalization over a certain set $\mathbf{X_i}$, which in the worst case would be a single variable X_i, that is calculated as described in Eq. (2.1). Because the bulk of the computational cost lies in this term, the focus of this thesis lies solely on how to obtain estimates for this equation. One can either directly compute this marginal by making use of *exact inference*, or use methods that approximate it called *approximate inference*. In the latter case, the obtained estimate for the marginal will be called *belief*.
In general, one can do *exact inference* for a single variable X_i by summing over all other, possibly unknown, variables, i.e.,

$$P(X_i = x_i) = \sum_{\sim x_i} P(x_1, x_2, \ldots, x_n) \tag{2.18}$$

$$= \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} P(x_1, \ldots, x_n). \tag{2.19}$$

Assuming that each variable can be in one of k states, the evaluation of the marginal over X_i requires $\mathcal{O}(k^{|\mathbf{X}|-1})$ summations, that means an evaluation of a sum with exponentially many terms in the number of variables is required. This quickly becomes computationally infeasible, when the number of variables gets larger. In fact, the direct calculation is NP-complete and for a precise complexity analysis the reader is referred to [20].
Hence, the task of methods that aim to perform exact inference is to simplify the used joint distribution. Popular methods are *cluster tree algorithms* like the Junction tree algorithm (JCT) [24], which will be used in the experiments for comparison in Chapter 4.

One method that can do both exact and approximate inference depending on the structure of the graph representing the distribution, will be described in the following section.

2.5.2 Belief Propagation

Having access to a factorization of a joint probability distribution and thus a *factor graph*, one can use this structure to apply the method of *belief propagation* (BP). It was introduced by Pearl in [3] and for a more general detailed description the reader is also referred to this publication.

For each two nodes of the factor graph a *message* is defined. A message from node i to node j,

represents the belief for node j, given all available information, represented by all the messages that node i receives. It should be noted, that there exists a message for both directions from i to j, as well as from j to i.
The value of a message is determined by the update equations, which are different for variables and factors (derived from [1]).
The message from variable node x_i to factor node f_j is given by

$$\mu_{x_i \to f_j}(x_i) = \prod_{f_k \in \mathbf{Nb}(x_i) \diagup f_j} \mu_{f_k \to x_i}(x_i). \tag{2.20}$$

The message from factor node f_i to variable node x_j is given by

$$\mu_{f_i \to x_j}(x_j) = \sum_{\sim x_j} f_i(\mathbf{x} \in \mathbf{Nb}(f_i)) \prod_{x_k \in \mathbf{Nb}(f_i) \diagup x_j} \mu_{x_k \to f_i}(x_k). \tag{2.21}$$

To get a clearer view, the update procedure for each type of message is depicted in Fig. 2.3 and 2.4.
The messages from variables to factors are initialized with 1 and the messages from factors to variables are initialized with the respective factor values. Each node in the graph *sends messages*, this means applying the update equations. The messages have to be sent until convergence has been reached, that is, until the messages reached their fixed points. Hence, this method is also called *Message Passing*.
In this thesis, the focus lies on computing the marginals over a single variable. Hence, the update equations are combined and the notation is changed to be more compact. This results in a single update equation for *incoming messages* at variable nodes, given by

$$\mu_{f_j}(x_i) = \sum_{\sim x_i} f_j(\mathbf{x} \in \mathbf{Nb}(f_j)) \prod_{x_k \in \mathbf{Nb}(f_j) \diagup x_i} \prod_{f_l \in \mathbf{Nb}(x_k) \diagup f_j} \mu_{f_l}(x_k). \tag{2.22}$$

Each of these messages is sent over a certain factor node f_j, that collects all relevant information and arrives at the variable node x_i.
To further simplify this equation, the set of messages that directly influences x_i will be denoted by $\mathcal{D}_i$. Consequentially, the double product on the right-hand side of Eq. 2.22 can be simplified to

$$\mu_{f_j}(x_i) = \sum_{\sim x_i} f_j(\mathbf{x} \in \mathbf{Nb}(f_j)) \prod_{x_k \in |\mathcal{D}_i|} \mu_{f_j}(x_k). \tag{2.23}$$

A depiction of this update equation can be seen in Fig. 2.5.
Using all incoming messages, the marginal for this variable can be computed as

$$P(x_i) \propto \prod_{f_j \in \mathbf{Nb}(x_i)} \mu_{f_j}(x_i). \tag{2.24}$$

A visual representation of this computation is depicted in Fig. 2.6.
If the graph has a tree structure, exact inference is possible, whereas the messages are guaranteed to converge. Otherwise, if the graph contains loops, convergence is not guaranteed. This means that the fixed points can not be reached and in this case the message passing should be performed until the message value changes are negligible. This procedure is then called *loopy belief propagation* (LBP) [9]. In this case, the resulting marginals are an approximation of the true marginals, but can also give good results.
The question that has been left open until now is in what order the messages should be sent,

which will be the main point of this thesis. Existing methods and variations will be described in the following chapter.

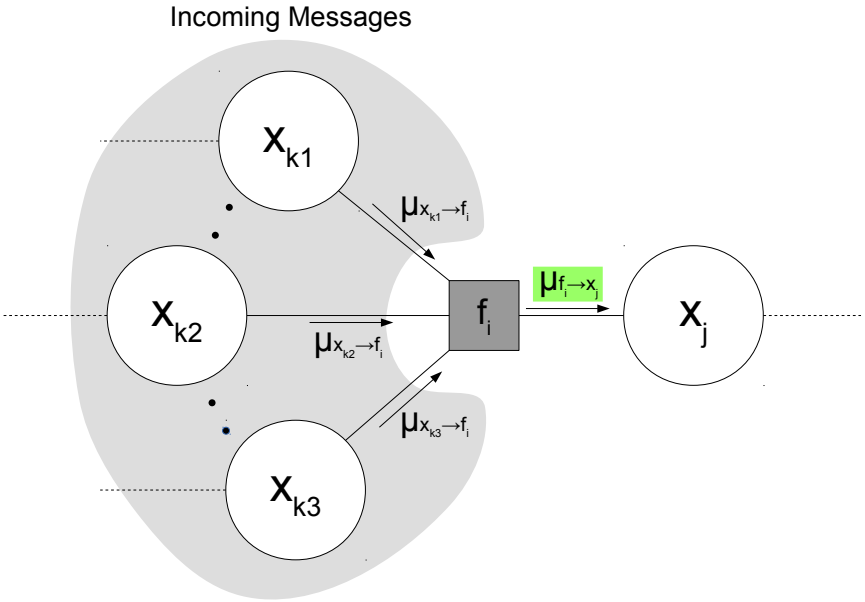


Figure 2.3: Factor to variable message update.

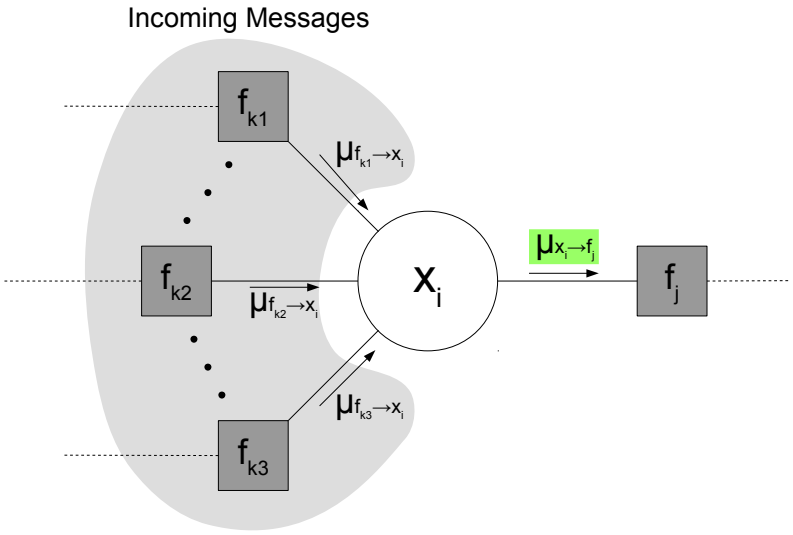


Figure 2.4: Variable to factor message update.

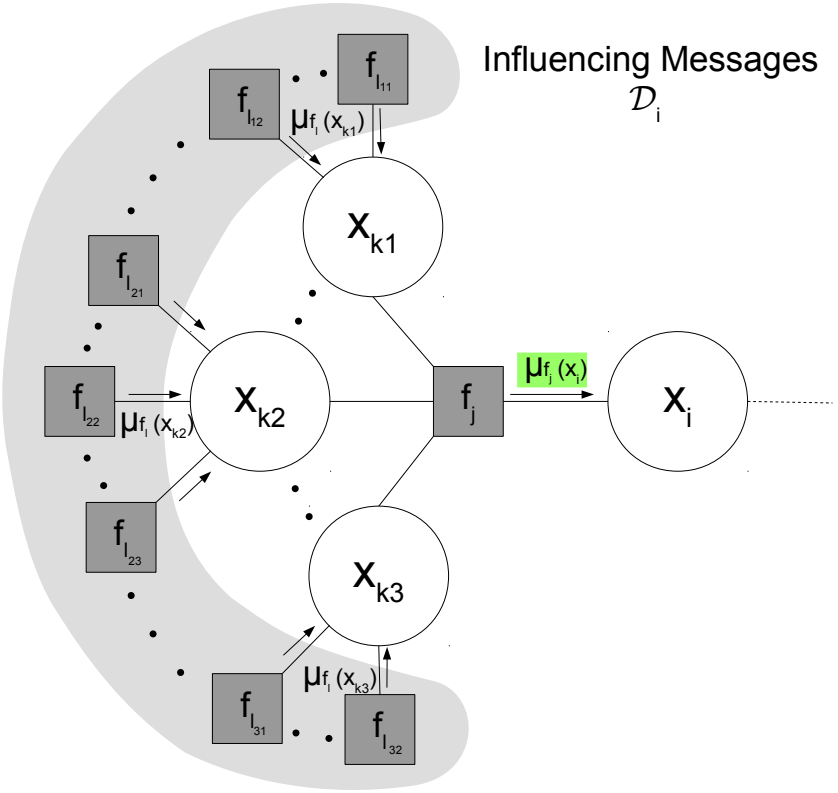


Figure 2.5: Message update via all influencing incoming messages.

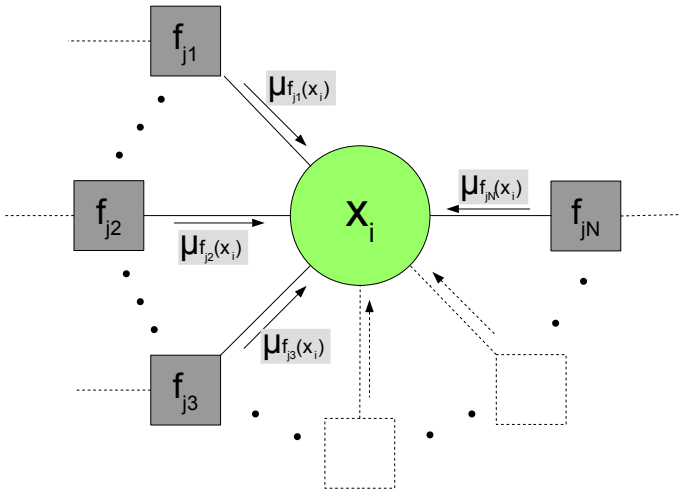


Figure 2.6: Computation of the marginal via all incoming messages.

3 Message Scheduling

In this chapter, we will discuss how to determine the order of message updates for the messages defined in the previous chapter. We will define methods of *message scheduling* such as static scheduling methods used by standard LBP. The more efficient dynamic scheduling method of residual belief propagation will be described and two methods that improve it will be presented.

3.1 Functional Scheduling Notation

To better describe the scheduling, an abstract functional notation adapted from the one described in [18] is used.
Each message is viewed as residing in some *message space* $\mathcal{R} \subset (\mathbb{R}^+)^d$, whereby one graph contains the set of messages denoted by $\mathcal{M}$, that are a subset of $\mathcal{R}^{|\mathcal{M}|}$. For simplicity it is assumed that all messages have the same dimension d, also the cardinality of $\mathcal{M}$ will be denoted by $M = |\mathcal{M}|$.
Individual messages are indexed via $\mathbf{m}_i \in \mathcal{R}$ and they are stacked together to form the message vector $\mathbf{m} \in \mathcal{R}^M$. The update equation can then be seen as a mapping function $f_i : \mathcal{R}^M \mapsto \mathcal{R}$. This mapping function makes use of the set of influencing messages $\mathcal{D}_i \subset \mathcal{M}$, introduced in Eq. (2.23)
Multiple mappings can then be combined to a global update function $F : \mathcal{R}^M \mapsto \mathcal{R}^M$. The difference between the scheduling methods lies in what individual updating functions are included in this global function. In the following, a short descriptive summary of the introduced vectors and the resulting update function is provided.

Individual Message: $\mathbf{m}_i = \begin{bmatrix} m_i^{(1)} \\ m_i^{(2)} \\ \vdots \\ m_i^{(d)} \end{bmatrix}$

Stacked Message Vector: $\mathbf{m} = \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_M \end{bmatrix}$

Update Function: $\mathbf{m}_i = f_i(\mathbf{m})$

3.2 Static Scheduling Methods

As the name implies, *static scheduling* means that one has a fixed global updating function, or in other words, a fixed order of updating the messages.
The most basic method updates all messages at once synchronously, thereby not even having a schedule to speak of and is thus called *synchronous belief propagation* (SBP). This results in the global update function

$$F^s(\mathbf{m}_1, \ldots, \mathbf{m}_M) = (f_1(\mathbf{m}), \ldots, f_M(\mathbf{m})). \tag{3.1}$$

An actual schedule is used in *asynchronous belief propagation* (ABP), where messages are updated in a certain fixed order, one after another. Thereby a global update function is used that only updates one certain message, given by

$$F_i^a(\mathbf{m}_1, \ldots, \mathbf{m}_M) = (\mathbf{m}_1, \ldots, f_i(\mathbf{m}), \ldots, \mathbf{m}_M). \tag{3.2}$$

Static methods usually use a round-robin schedule to make sure that all messages are updated regularly. A common schedule is to update each message that has received a new incoming message and thereby assumes a different value, which at first glance sounds like a dynamic schedule. This rule however results in a certain cycle of message updates that depends solely on the network structure. In most publications, when talking about BP, this "naive" ABP variant is meant. Hence, for the experiments this method has been implemented and a pseudo-code will be presented at the end of this chapter.
Regardless of what asynchronous schedule is chosen, all these methods have in common that they try to capture all information updating all messages in a specific manner, without having any selective message scheduling process behind them.
Nevertheless, [18] shows that the convergence rate of asynchronous scheduling methods is at least as good as that of synchronous methods, which will be confirmed later in Chapter 4.

3.3 Dynamic Scheduling: Residual Belief Propagation

In practice, many messages already converge after a short period of time. Using this observation, the idea is that the updating should concentrate on the remaining messages, instead of doing redundant updates. In order to do this, the actual message values are taken into consideration, or rather, how much each message changed after the update.
This leads to the method introduced by [18] called *residual belief propagation* (RBP). In this context the *residual* of a message is defined as the distance measure obtained from the difference of its value before and after the update, defined as

$$r_i(\mathbf{m}) = ||f_i(\mathbf{m}) - \mathbf{m}_i||_n. \tag{3.3}$$

Messages are updated one after another, however the introduced scheduling selects messages depending on their current message values, in that the message with the highest *residual* is chosen to be updated.
In [18] the used norm for the residual $||\cdot||_n$ is not specified, nevertheless the infinity norm was used in the experiments. This matches the measure that was used in [19], where another method

of improvement is described.
Having access to the residuals of each message in the network, the RBP algorithm uses the residuals to select the message to be updated as

$$\hat{u} = \text{argmax}_u \, r_u(\mathbf{m}). \tag{3.4}$$

Using this technique, one can formulate the RBP algorithm as it will be used in the experiments, which will be done in the following section.

3.4 Pseudo-Codes for SBP, ABP and RBP

In this section, pseudo codes for the SBP, ABP and RBP algorithms, as they will be used in the experiments, are described to provide a reasonable comparison.
All methods use the function `convergenceCriterion` to check if the iterations can be stopped. The criterion that will be used in the experiments is

$$\max[\mathbf{r}] < r_{thresh}, \tag{3.5}$$

where r_{thresh} was set to 10^{-3}, which conforms with [19]. Since the infinity norm is used for the residuals, if all messages have not changed any of their values up to the third decimal point, the algorithm is assumed to be converged.
Furthermore, the only thing needed as input for the algorithms is the set of discrete factors $\{F_1, \ldots, F_D\}$, given by a FG. All factors are provided by a set of matrices $\{\mathbf{F}_1, \ldots, \mathbf{F}_D\}$. Each matrix dimension is representative of one argument of the factor.

3.4.1 Message Update

At this point, one should clearly define the term of a *message update* used in the following descriptions and the evaluation of the methods. This definition is based on the one used in [19]. Each message has a current "old" value which is the value that has been set and is seen by the other messages. All current old message values together form the vector $\mathbf{m}^{old}$. Additionally each message has a "new" value which is the value that satisfies the update equations and thus the value given by the incoming old messages. It is obtained by using the mapping function $\mathbf{m}_i^{new} = f_i(\mathbf{m}^{old})$, altogether forming $\mathbf{m}^{new}$.
The important distinction to make here is that just *obtaining* the new message value is different than *assigning* it with the new value $\mathbf{m}_i^{old} = \mathbf{m}_i^{new}$. If a message is assigned a new value, the new value of all messages influenced by this message changes. When counting the *number of updates*, one refers to the number of re-computations, that means how many times a new message value has been obtained.
The update function `computeUpdate` used by all algorithm implementations, is described in Algorithm 1 and it is based on Eq. (2.23). It describes the update procedure of message $\mathbf{m}_i$, incoming to variable x_I, sent over factor f_j. To represent this factor, a factor matrix $\mathbf{F}_j$ is used. To implement the update equation, entry-wise products (Hadamard products) and products along one matrix dimension i denoted by $\times_i$ are applied to the factor matrix $\mathbf{F}_j$. These operations replace the summations from the original formula and also conform exactly to how it is implemented. In Line 3 the expression $incMsgIndex$ is used to obtain the indices of all messages arriving at the respective variable node. The normalization step in Line 5 guarantees that the tentative values of the message correctly represent a probability.
On a further note, sometimes the notation $\mathbf{F}_{m_i}$ will be used for the input factor matrix. This refers to the factor over which the message m_i is sent.

Algorithm 1: ComputeUpdate

input : $\mathbf{m}_i^{old}$, current value of message arriving at variable x_I
$\mathbf{F}_j$, matrix representing factor f_j that sends the message
output: $\mathbf{m}_i^{new}$

```
1 m_i^new ← F_j;
2 for each x_k = Nb(f_j) other than x_I do
3 |   product ← ∏_{l∈incMsgIndex(x_k)} m_l;
4 |   m_i^new ← m_i^new ×_{x_k} product;
5 |   Normalize m_i^new
```

3.4.2 SBP

In the following, Algorithm 2 describes the implementation of SBP.

Algorithm 2: SBP

input : Number of messages M
Set of factors $\{F_1, \ldots, F_D\}$
output: Converged stacked message vector $\mathbf{m}^{old}$

```
1 m^old ← uniform message array of size M;
2 k ← 1;
3 while k < k_max and convergenceCriterion(m^old, m^new) = false do
4 |   for i ← 1 to M do
5 |   |   m_i^new ← computeUpdate(m_i^old, F_{m_i})
6 |   for i ← 1 to M do
7 |   |   m_i^old ← m_i^new
8 |   k ← k + 1
```

In analogy with the former description, one can clearly see the difference between *obtaining* a new message value (Line 5) and *assigning* one (Line 7), where for each there is an individual loop over all messages. k_{max} specifies the maximum number of iterations.

3.4.3 ABP

Algorithm 3 describes the details of the used implementation of the "naive" ABP method.

The changes compared to Algorithm 2 (SBP) include an introduction of two additional message index arrays denoted by $\mathbf{u}$ and $\mathbf{v}$. The first array $\mathbf{u}$ holds the indices of all messages that should be updated and is initialized to contain only one random value in the beginning at Line 5. The second array $\mathbf{v}$ holds all indices of messages that have to be recomputed because they are influenced by updating the messages from $\mathbf{u}$. The set $\mathcal{D}_j$ that contains all messages that are influenced by $\mathbf{m}_j$ is used in Line 10 to get these indices. The used function $\texttt{push}(x, y)$ is a generic array function used to add element x to the array y.
In contrast to SBP, the aforementioned loops for obtaining the message values and for the assignment of the messages with their new values, have to only iterate over messages with indices from these arrays instead of all messages. Respectively, this is done in the loop at Line 12 and the loop in Line 8.

Algorithm 3: ABP

input : Number of messages M
Set of factors $\{F_1, \ldots, F_D\}$
output: Converged stacked message vector $\mathbf{m}^{old}$

$\mathbf{m}^{old} \leftarrow$ uniform message array of size M;
for i $\leftarrow$ 1 **to** M **do**
  $\mathbf{m}_i^{new} \leftarrow$ `computeUpdate(`$\mathbf{m}_i^{old}$, $\mathbf{F}_{m_i}$`)`
u, **v** $\leftarrow$ empty index array;
`push(Random(M),` **u**`)`;
k $\leftarrow$ 1;
while k $<$ k_{max} **and** `convergenceCriterion(`$\mathbf{m}^{old}$, $\mathbf{m}^{new}$`)` $=$ *false* **do**
  for i $\leftarrow$ 1 **to** `length(`**u**`)` **do**
    $\mathbf{m}_{u_i}^{old} \leftarrow \mathbf{m}_{u_i}^{new}$;
    for *all* j *where* $\mathbf{m}_{u_i}^{old} \in \mathcal{D}_j$ **do**
      `push(`j, **v**`)`;
  for i $\leftarrow$ 1 **to** `length(`**v**`)` **do**
    $\mathbf{m}_{v_i}^{new} \leftarrow$ `computeUpdate(`$\mathbf{m}_{v_i}^{old}$, $\mathbf{F}_{v_i}$`)`;
  u $\leftarrow$ **v**;
  v $\leftarrow$ empty index array;
  k $\leftarrow$ k + 1

3.4.4 RBP

The implementation of the RBP method, according to the description above is described in Algorithm 4. The implementation follows closely the description above, with an introduction

Algorithm 4: RBP (adapted from [18])

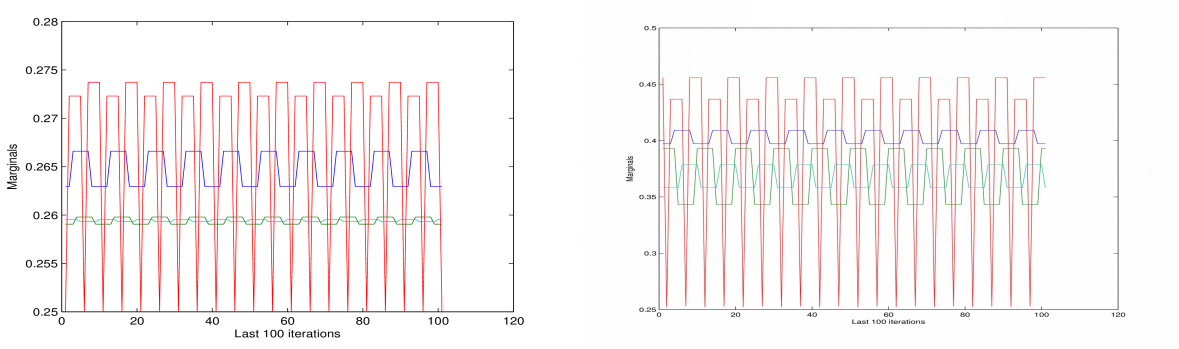
input : Number of messages M
Set of factors $\{F_1, \ldots, F_D\}$
output: Converged stacked message vector $\mathbf{m}^{old}$

$\mathbf{m}^{old} \leftarrow$ uniform message array of size M;
r $\leftarrow$ array of zeros with size M;
for i $\leftarrow$ 1 **to** M **do**
  $\mathbf{m}_i^{new} \leftarrow$ `computeUpdate(`$\mathbf{m}_i^{old}$, $\mathbf{F}_{m_i}$`)`;
  $r_i \leftarrow ||\mathbf{m}_i^{old} - \mathbf{m}_i^{new}||_\infty$
k $\leftarrow$ 1;
while k $<$ k_{max} **and** `convergenceCriterion(`$\mathbf{m}^{old}$, $\mathbf{m}^{new}$`)` $=$ *false* **do**
  u $\leftarrow$ `index_max(`**r**`)`;
  $\mathbf{m}_u^{old} \leftarrow \mathbf{m}_u^{new}$;
  $r_u \leftarrow 0$;
  for *all* j *where* $\mathbf{m}_u^{old} \in \mathcal{D}_j$ **do**
    $\mathbf{m}_j^{new} \leftarrow$ `computeUpdate(`$\mathbf{m}_j^{old}$, $\mathbf{F}_{m_j}$`)`;
    $r_j \leftarrow ||\mathbf{m}_j^{old} - \mathbf{m}_j^{new}||_\infty$
  k $\leftarrow$ k + 1

of array **r** containing the residuals of all messages. The function `index_max` is used to find the index of the message with the maximum residual.

3.5 RBP Improvements

When applying RBP to certain graphs, a rather low convergence rate may be observed. This is an occurrence observed in graphs containing many loops with unfavorable factors. In order to pinpoint the source of this behavior, the evolution of the message values has been thoroughly studied.
The regularly observed problem is oscillation of the message values. In such a case, the schedule picks the same messages periodically, because the messages assume approximately the same values periodically. However these alternating values happen to be different enough such that none of the message residuals satisfy the convergence criterion, hence convergence can not be reached.
In Fig. 3.1 this is demonstrated, by depicting excerpts from the evolution of the marginals of a fully connected Ising network with four variables and unfavorable factors. The assumption



(a) (b)

Figure 3.1: Two examples for oscillating marginals $P(X_i = 0)$, as a result of applying RBP.

is that in order to improve the performance one has to find a way to avoid these message oscillations. Therefore, two methods that aim to break the oscillation cycles will be presented.

3.5.1 Noisy RBP (RBPnoise)

The idea behind the first method is to directly influence oscillating messages, thereby not changing the scheduling at all in case there are no oscillations. To break the message value cycle, Gaussian noise is added to the updated message value of the oscillating messages. Hence, it is denoted by *RBPnoise*. The assumption is that the injected noise "kicks" the message out of its oscillating state.
Before changes can be applied message value oscillations have to be detected. In order to do that, during the scheduling process where the message with the largest residual is selected, its short message value history is checked for ambivalence. More specifically that means, the last L assigned values are compared up to a certain decimal point and if a duplicate is detected, the oscillation check is confirmed. In every iteration where this occurs, instead of doing the usual update by assigning the new message value, samples from a noise vector are added to the message.

Applying this to Algorithm (4) from above, Line 9 and 10 change to

```
 9 if isOscillating(m_u^old, L)
10     m_u^old ← m_u^new + N(0, σ)
11 else
12     m_u^old ← m_u^new
13 r_u ← 0
```

In Line 10 one can see the noise added to the original new message vector. In the experiments, zero mean Gaussian noise with $\sigma = 0.25$ was found to give the best results. For the history length L a value of 10 was found to be appropriate.
The addition of noise leads to changes in the scheduling, such that different regions of the graph are randomly "explored", which can lead to convergence. However, it should be noted that even though the residual is set to zero in Line 13, this is only done to indicate that it has been updated, the actual true residual would be the noise value.
The code for the complete algorithm was moved to Appendix A.

3.5.2 Update Frequency Weighted RBP (RBPnUp)

In the case of message oscillations, a select few messages are updated repeatedly. Considering the *number of updates* for each message, it is clear that it will be very high for the oscillating messages. The idea behind the second improved RBP method is that this *update frequency* should be taken into account in the message schedule to indirectly force a different scheduling where oscillations are avoided.
In order to do this, for each message, a counter is introduced that counts how often it has been updated. Instead of directly using the residual, each message is attributed with a *scheduling value*, that depends on the residual as well as the message update counter and is defined as

$$s_i = \frac{r_i}{numUpdates(\mathbf{m}_i)}. \tag{3.6}$$

In the selection process, this value is used instead of the residual to get the next message for update. This method is denoted by *RBPnUp* to indicate the influence of the number of updates onto the schedule.
Applying these changes, each message that is frequently selected for update will have a residual with reduced significance, without directly influencing its message value. As opposed to the other method, there is no external impact onto the message values, only the scheduling itself is affected, whereby the residuals are still a deciding factor. Since the measure that is used to determine which message is selected has changed, the schedule will always differ from the standard RBP schedule. On top of preventing oscillations, this results in a more evenly distributed message schedule.
Applying these changes to the pseudo-code from above, the residuals are replaced with the scheduling value $\mathbf{s}$ and the variable $numUpdates$ is introduced to implement the update counter. This results in the following adapted Algorithm 5.

Algorithm 5: RBPnUp

input : Number of messages M
Set of factors $\{F_1, \ldots, F_D\}$
output: Converged stacked message vector $\mathbf{m}^{old}$

$\mathbf{m}^{old} \leftarrow$ uniform message array of size M;
s, **numUpdates** $\leftarrow$ array of zeros with size M;
for i $\leftarrow 1$ **to** M **do**
    $\mathbf{m}_i^{new} \leftarrow$ `computeUpdate(`$\mathbf{m}_i^{old}$, $\mathbf{F}_{m_i}$`)`;
    $s_i \leftarrow ||\mathbf{m}_i^{old} - \mathbf{m}_i^{new}||_\infty$;
    $\mathbf{numUpdates}(i) \leftarrow 1$
k $\leftarrow 1$;
while k $<$ k$_{max}$ **and** `convergenceCriterion(`$\mathbf{m}^{old}$, $\mathbf{m}^{new}$`)` $= false$ **do**
    u $\leftarrow$ `index_max(`**s**`)`;
    $\mathbf{m}_u^{old} \leftarrow \mathbf{m}_u^{new}$;
    $\mathbf{numUpdates}(u) \leftarrow \mathbf{numUpdates}(u) + 1$;
    $s_u \leftarrow 0$;
    for *all* j *where* $\mathbf{m}_u^{old} \in \mathcal{D}_j$ **do**
        $\mathbf{m}_j^{new} \leftarrow$ `computeUpdate(`$\mathbf{m}_j^{old}$, $\mathbf{F}_j$`)`;
        $s_j \leftarrow \frac{||\mathbf{m}_j^{old} - \mathbf{m}_j^{new}||_\infty}{\mathbf{numUpdates}(u)}$
    k $\leftarrow$ k $+ 1$

4 Experiments and Results

In this chapter, the experimental setup and the results will be discussed. Two experiments have been performed to compare SBP, ABP, RBP and the improved RBP methods. Both use Ising factors, whereby in the first experiment small complete graph is used and in the second experiment grid graphs of varying size are used.

4.1 Experimental Setup: Ising Models

The algorithms and the models were implemented in MATLAB, according to the pseudo-codes presented in the previous chapter.
Discrete factors from a factor graph were used in the computation of the message updates. Each factor is represented by a multidimensional array $[\mathbb{R}^d \times \mathbb{R}^d \times \dots \mathbb{R}^d]$, according to how many variables are included.
Even though the usage of arbitrary dimensional factors is possible in the implementation, both experiments use binary random variables and only single and pairwise variable factors, that is $\mathbf{f}_{single} \in [\mathbb{R}^2]$, $\mathbf{F}_{pair} \in [\mathbb{R}^2 \times \mathbb{R}^2]$.
Structurally, each variable node x_k is connected to exactly one single variable factor f_k, which represents the "local evidence" and to multiple pairwise factors, which represent interactions between the variables. This is common practice in BP benchmarks and matches the experiments presented in [7, 18, 19, 25].
In the context of the algorithms described in the last chapter, the single variable factors can be treated as incoming messages, albeit with constant message values. Accordingly, they are used in computing the new message values, but never chosen to be recomputed. The used set of factors $\{F_1, \dots, F_D\}$ will be set to the pairwise factors $\mathbf{F}_{pair}$.
Following [18] and [19], both experiments use Ising models from statistical physics. Ising models are commonly used in experiments to evaluate BP. For more information about these models

the reader is referred to [26] or [27]. The factors are called potentials and denoted by

$$\mathbf{f}_{single} = \phi_i(x_i) = \begin{bmatrix} e^{h_i C} \\ e^{-h_i C} \end{bmatrix}, \tag{4.1}$$

$$\mathbf{F}_{pair} = \Phi_{ij}(x_i, x_j) = \begin{bmatrix} e^{J_{ij} C} & e^{-J_{ij} C} \\ e^{-J_{ij} C} & e^{J_{ij} C} \end{bmatrix}. \tag{4.2}$$

The used parameters $h_i \in \mathbf{h}$ for uni-variate potentials and $J_{ij} \in \mathbf{J}$ for pairwise potentials are chosen from a certain range according to the experiments, which will be defined at a later point, in the respective sections.
To control the difficulty of the inference tasks, the parameter C is included in all exponents, where higher values of C will increase the difficulty to reach convergence.

4.2 Complete Graph with Uniform Factors

The first simulation is based on the the numerical experiments from [7] and [25]. A complete graph of four nodes is used as depicted on Fig. 4.1. The main task of this experiment is to show the improvement of RBP over ABP.

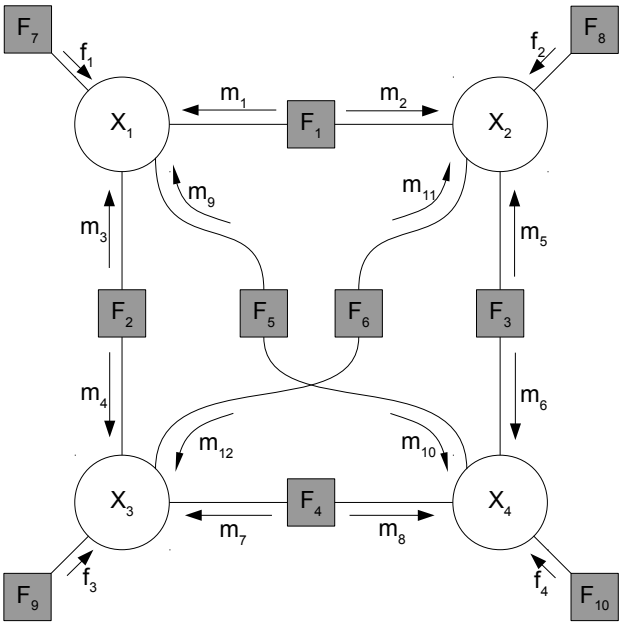
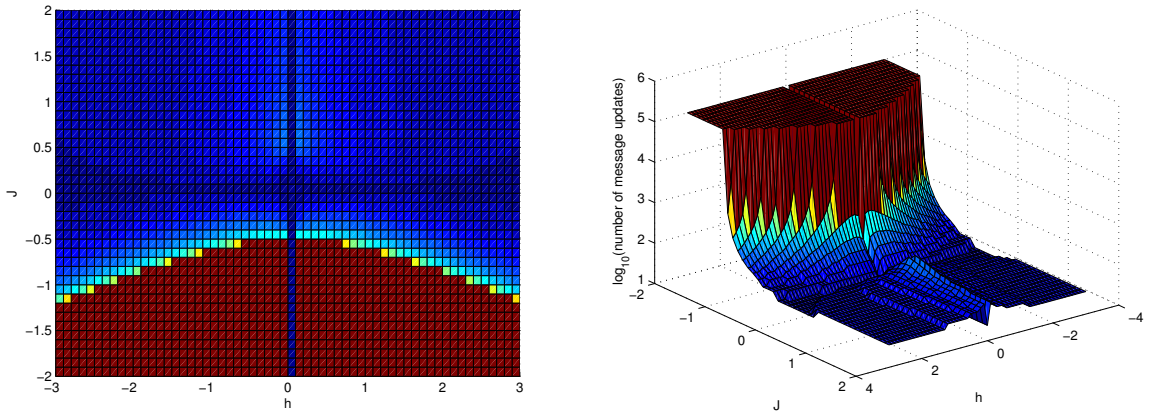


Figure 4.1: Complete Graph for four variables.

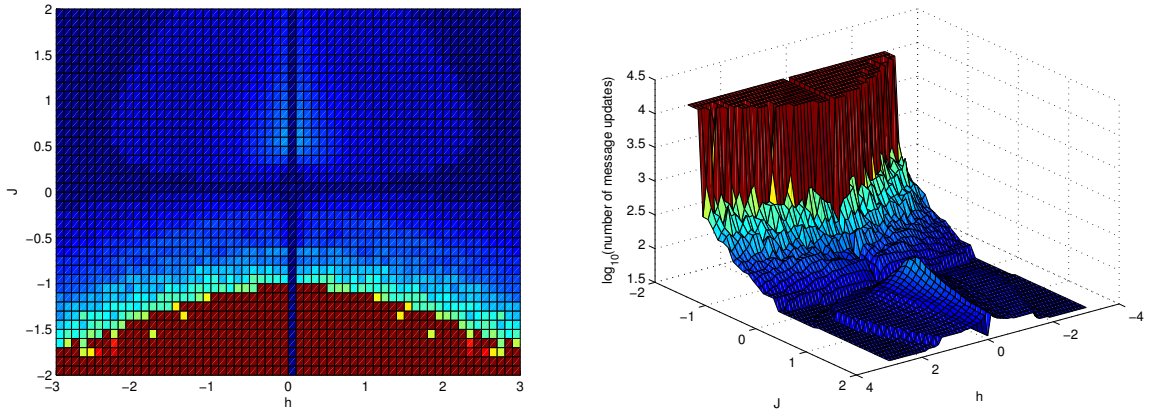
The parameters are chosen uniformly for all factors, i.e., all h_i and J_{ij} are the same for all nodes. They are chosen in the ranges $h \in [-3, 3]$ and $J \in [-2, 2]$, while leaving the difficulty parameter to not affect the simulations and thus $C = 1$.
The simulation is repeated using steps of 0.1 for the ranges with all combinations of the parameters, thus having 61 different values for h, 41 different values for J, resulting in a total number of 2501 simulations. The number of updates is counted each time if convergence was reached.

As a reminder, the number of updates is increased every time a new message value is computed, not when the message is assigned with its new value.
In Figures 4.2-4.5 the number of updates needed to reach convergence is depicted for each method. Two views for each combination of different values of J and h have been created, to match the Figures from [25].
In sub-figures (a) a top-down view is given, with the blue colors representing a low number of updates until convergence has been reached and the red colors a high number, respectively. In sub-figures (b) a different perspective is given, that more clearly shows the color coding of the number of updates. For the z-axis the decimal logarithm of the number of updates was used. The threshold for the number of updates when to stop the simulation because convergence can not be reached was chosen accordingly. The threshold was chosen higher for ABP, since one has a much higher number of updates when recomputing all messages at each iteration.
Reaching convergence is an inherent hard task for certain parameter configurations of h and J. Thus, common BP methods such as ABP do not find a fixed point in the shown area. Yet, using message scheduling clearly improves the convergence region while also reducing the number of required message updates. In [7], various convergence margins for the used graphs are described. Using RBP methods, the margins that define the largest region of convergence can be reached. Concerning the improved RBP methods, an additional slight improvement of the convergence region can be seen. However, a very large number of updates is required for the newly acquired regions. Hence, by the cost of computation time, the convergence rate can be increased. In the discussion of the second experiment, this performance result will be described in more detail.



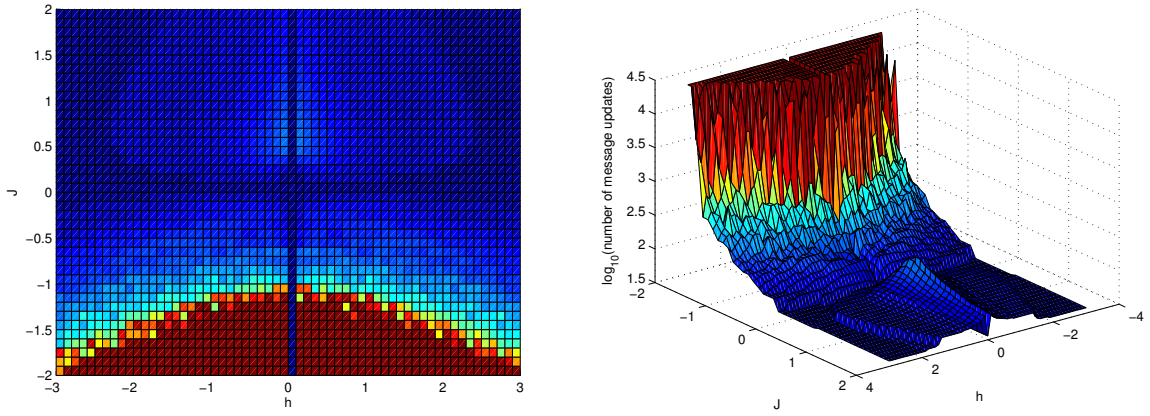
(a) (b)

Figure 4.2: ABP performance.



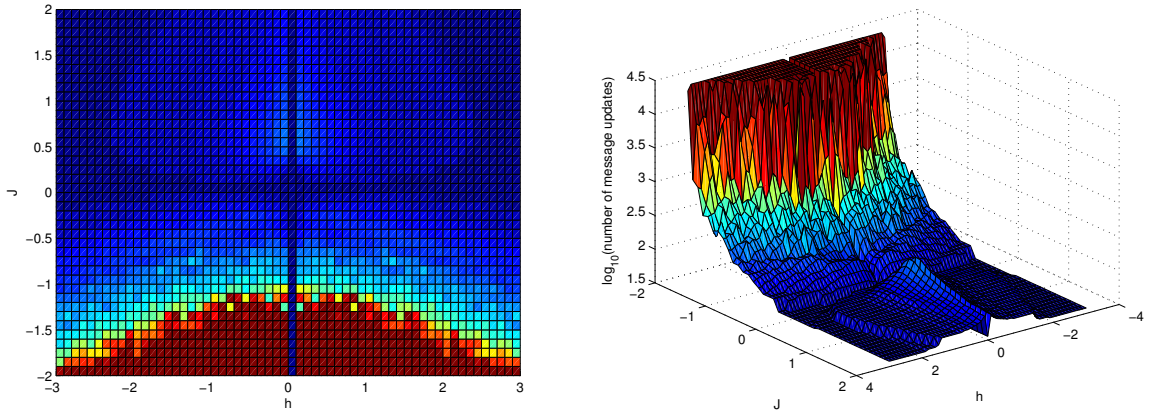
(a) (b)

Figure 4.3: RBP performance.



(a) (b)

Figure 4.4: RBPnoise performance.



(a) (b)

Figure 4.5: RBPnUp performance.

4.3 Grid Graph with Random Factors

In the second experiment $N \times N$ Ising grids were used to better demonstrate the differences between the methods in terms of performance. As mentioned, these graphs are the standard benchmark for evaluating BP algorithms, since the graphs contain many loops and are thus problematic to converge. They were also used in [12, 18, 19] among others.
The parameters $J_{ij} \in \mathbf{J}$ and $h_i \in \mathbf{h}$ are uniformly sampled in the range $[-0.5, 0.5]$ using a random seed for the random number generator, such that the parameters can be recreated for all different methods.
The experiments have been performed for graphs of different grid sizes of $N = \{7, 11, 9, 13\}$, the results of which will be compared in the performance evaluation. The challenge parameter was always set to the grid size N, which is in concert with the experiments presented in [18]. An outline of one of the used Ising grids with a size of 11×11 is depicted in Fig. 4.6. The simulation

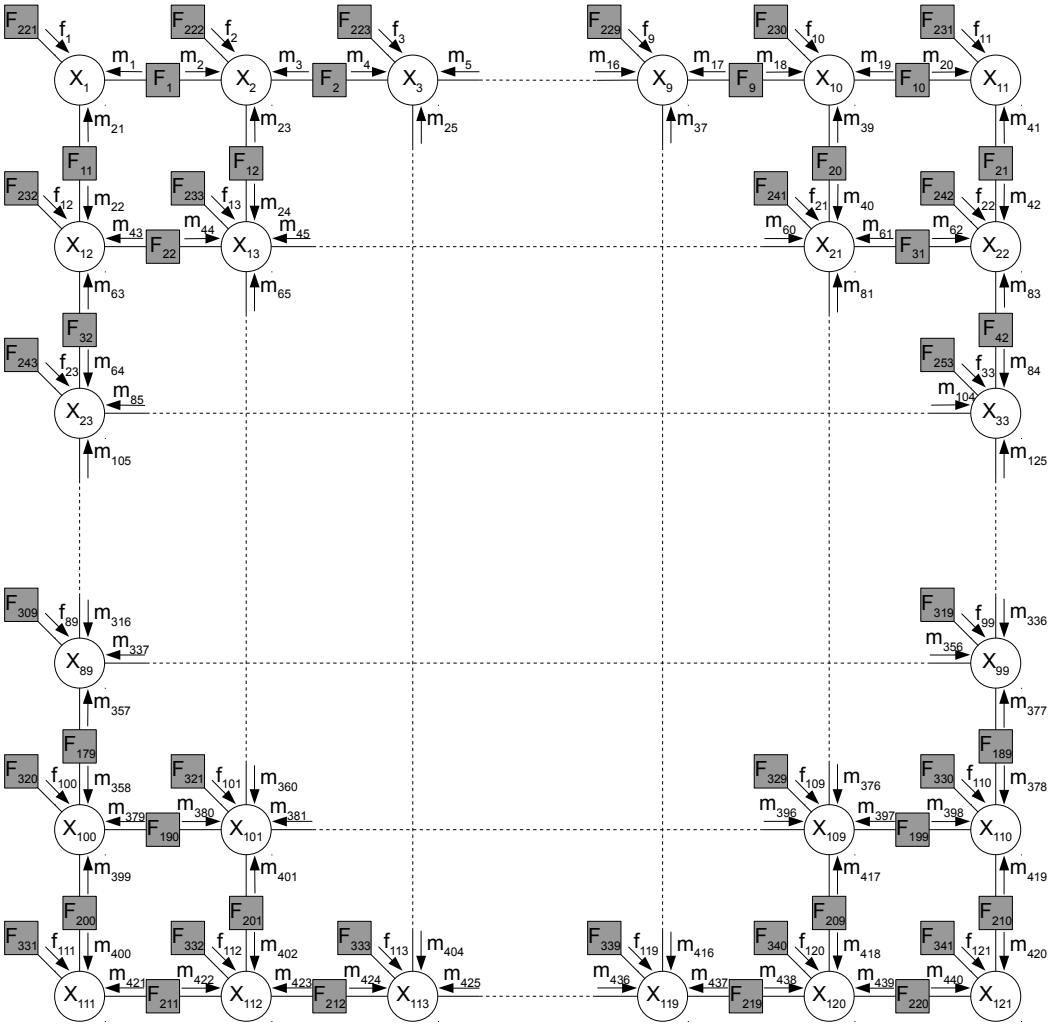


Figure 4.6: 11×11 *Ising grid.*

was performed 233 times with different random parameters for each of the methods, to get a

reasonable estimate of the performance. This number is a result of the number of available simulation hosts. For each run, the evolution of the marginals $P(x_i)$, $\forall i$ was recorded, including the number of updates at each iteration. These were used to evaluate various performance measures that are presented in the following sections. In 4.3.1, the evolution of the message values will be analyzed for 11×11 grid graphs. In 4.3.2, the speed for the rate of convergence will be discussed for all methods and all grid sizes. In 4.3.3, the obtained marginals will be compared to the exact marginals using different error measures. In conclusion, in 4.3.4 the convergence and quality measures will be summarized and discussed.

4.3.1 Convergence Properties: Evolution

This part of the performance evaluation was only performed for the 11×11 graphs. Since with this graph one has 440 messages to look at, to visualize the evolution of the message values, the marginals are considered because they are a direct result of the messages (see Eq. (2.24)).
For the visualization of the evolution of the marginals, all marginals of all 121 variables where plotted over the number of updates, but just for one variable value $P(X_i = 0)$, which is sufficient to evaluate the convergence properties because binary random variables are used. A few example runs were picked out to show the differences between the methods.

SBP / ABP compared to RBP

Before looking at the newly introduced methods, the difference of the already established methods in terms of evolution of marginals is presented.
Concerning the differences between SBP and ABP, in general, they have a very similar evolution with ABP being always faster than SBP in terms of number of updates and in a few instances ABP can reach convergence where SBP could not. One example of such an occurrence can be seen in the evolution plot in Fig. 4.7. One can see how a synchronous update of all messages, can lead to long term oscillations of the message values, such that convergence can not be reached. The previously described naive ABP, is superior to SBP and a popularly used method. That is why it will be used in all following comparisons.
There was not a single run among all simulation runs, where ABP did converge and RBP did not. This matches the remarks from [18], where it is shown that RBP performs as least as good as ABP. An example plot is shown in Fig. 4.8, where ABP is trapped in the repeated update of a certain set of messages, while the intelligent update of RBP results in a fast convergence.

RBP compared to improved methods

Analogously, in this section the convergence of the original RBP method is compared to that of both introduced improved RBP methods.
Since the RBP noise method only takes effect when oscillations are detected, all simulations without oscillations have the exact same evolution of the marginals as standard RBP. If oscillation is detected, the noise can "kick" the message into a different state where convergence is possible. One example for this can be seen in Fig. 4.9, where RBP is quickly trapped in the oscillation state, while the noise method repeatedly escapes it, until it arrives in a configuration where convergence is reached.
For RBPnUp, since one does always weight the scheduling values with the respective number of updates, the evolution of the marginals is never the same as with the original RBP method. In fact, for difficult graphs, this can lead to this method taking longer to reach convergence or even not reaching convergence at all, even though the original method did converge. One example of this can be seen in Fig. 4.10. On the good side, this method can influence long term

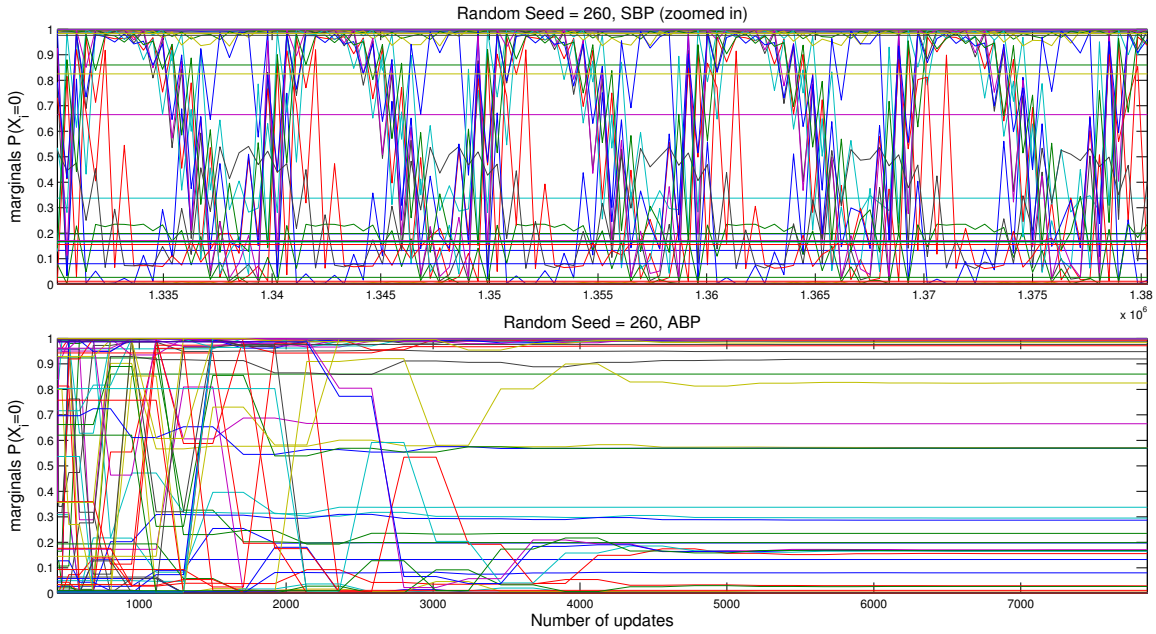


Figure 4.7: Only ABP converges (bottom) and SBP results in long-term oscillations (top).

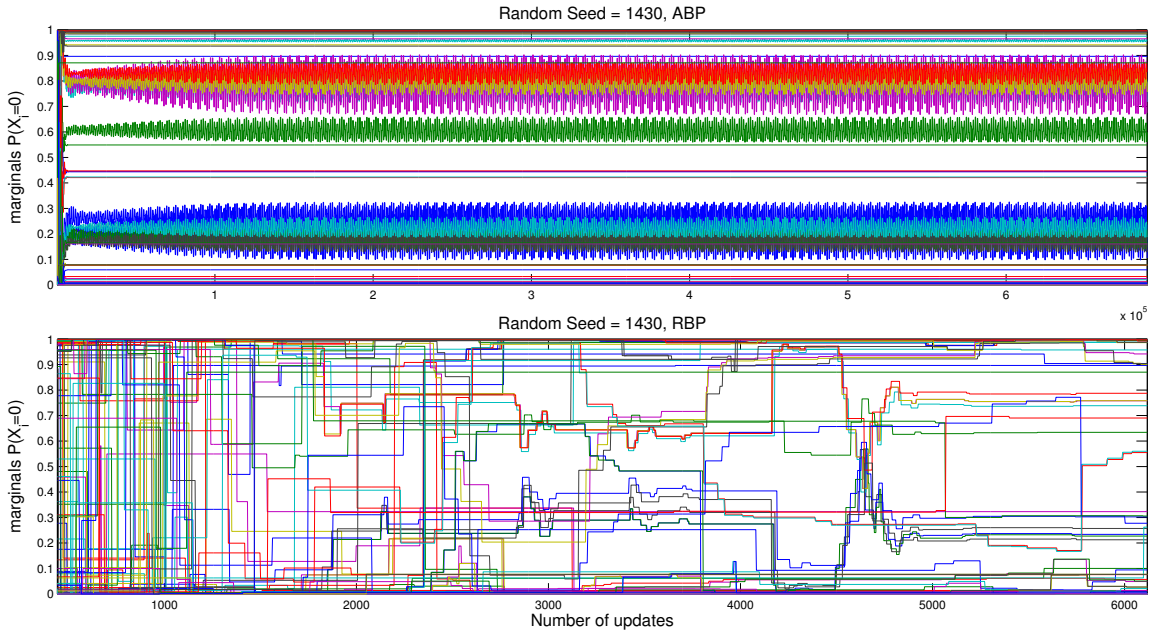


Figure 4.8: Only RBP converges (bottom) and ABP is quickly trapped in oscillations (top).

oscillations that the previous method was not able to detect. Hence, it can lead the respective messages to convergence before the noise method would even detect any oscillations, as can be seen in Fig. 4.11, where convergence is reached after 20000 updates, whereas RBPnoise would not detect oscillations until 40000 updates have been performed.
Additional evolution plots that demonstrate strengths of the different methods can be found in Appendix B.

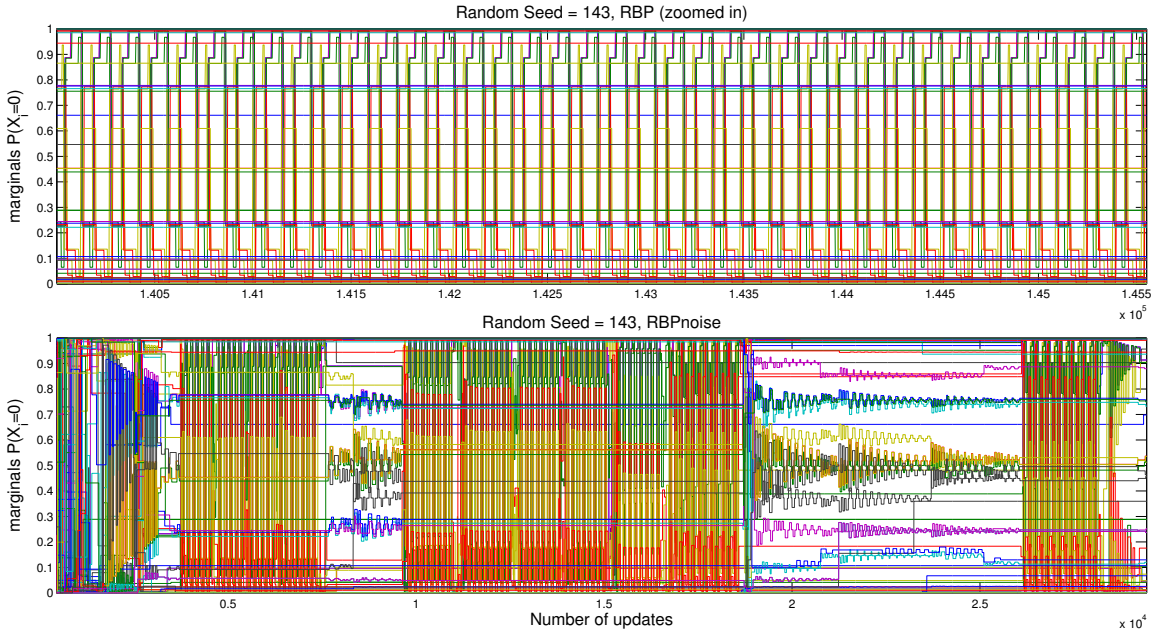


Figure 4.9: RBPnoise reaches oscillation multiple times until convergence (bottom) and RBP results in oscillations (top).

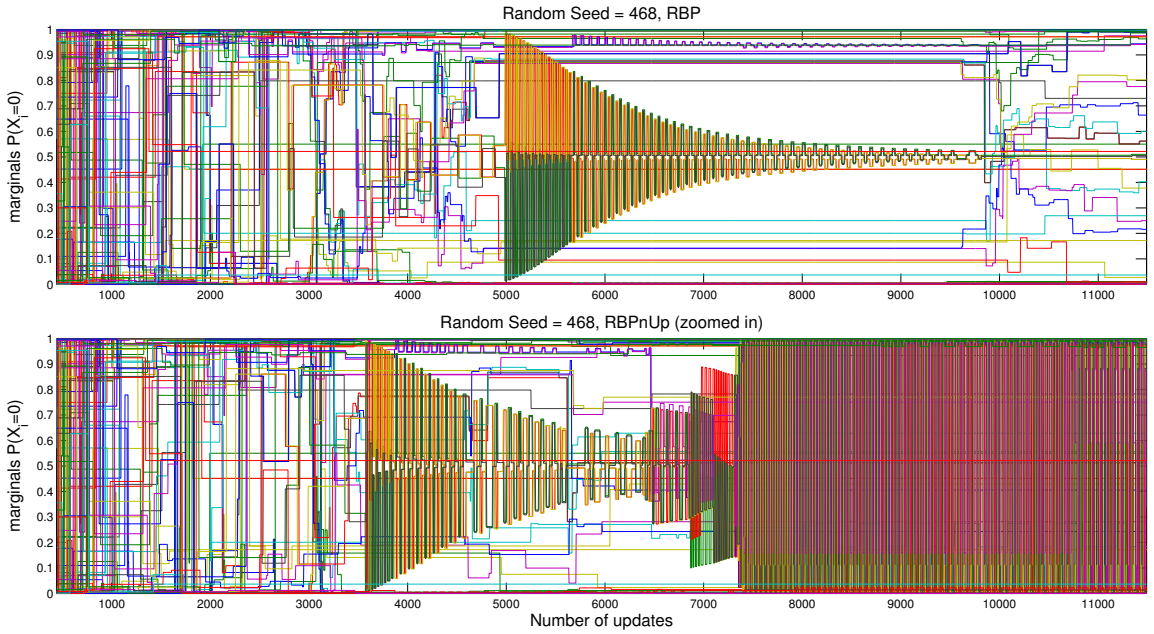


Figure 4.10: RBP convergences (top) but weighting results in unstable state (bottom).

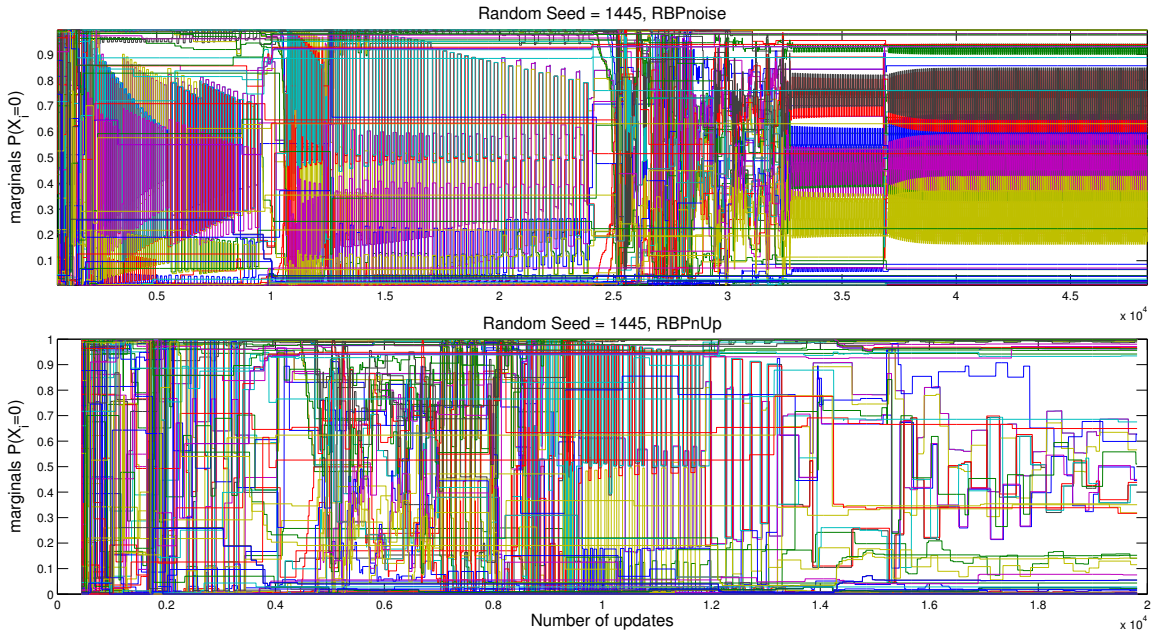


Figure 4.11: RBPnUp convergences (bottom) long before noise method would even detect any oscillations (top).

4.3.2 Convergence Properties: Speed

In order to visualize the average convergence speed as well as the achieved convergence rate for each method, the number of updates when convergence was reached has been marked for each run. Counting the number of converged runs for all update counts, convergence rate plots were generated for different grid sizes. Respectively, in Fig. 4.12 and in Fig. 4.13 the rates are plotted over the number of updates for smaller grids ($N = 7$ and $N = 9$) and larger grids ($N = 11$ and $N = 13$).
These plots show the increase of the convergence rate of the improved RBP methods over ABP and RBP by the cost of computation time. This is because the introduced methods are not trapped in oscillations and can run for a longer time. For larger and hence more difficult grids, the increase of the convergence rate is more apparent.
For smaller grids, represented by the left plot in Fig. 4.12, all methods seem to converge with about the same speed. The maximum number of needed updates to reach convergence resides at about 50000 message updates. On the other hand the reached convergence rate is also very similar across all RBP methods. Only ABP reaches a considerably lower convergence rate.
Increasing the size of the grid and thus the difficulty, the improved RBP methods gradually become slower compared to RBP and ABP. For the largest grid, seen in the right plot in Fig. 4.13, the maximum number of needed updates for the improved RBP methods amounts to about 250000, while ABP and RBP need about 150000. As already shown in [18], ABP is a lot slower than RBP, having a very low convergence rate when a low number of updates has been performed, i.e. after 50000 updates less than 5% of the runs reached convergence. In [19] Sutton and McCallum present another way to improve the convergence speed of RBP, where approximations of message values are used instead of doing the full re-computation.
Considering the improved RBP methods, it can be seen that RBPnoise has a higher convergence rate than RBP for all number of updates, no matter the grid-size. This is to be expected, because RBPnoise works exactly the same as RBP, unless oscillation has been detected. RBPnUP on the other hand, needs some time to surpass RBP when using larger grids, as can be seen in Fig. 4.13. However the improvement for the reached convergence rate also gets higher for larger grids. This is especially true for RPBnoise, which has the overall best performance in terms of convergence rate.
On a final note, in the left plot in Fig. 4.13 it can be seen that RBPnoise and RBPnUP reached about the same convergence rate for $N = 11$ grids. It is not clear why this is that much different than the results of the other grid sizes where RBPnoise is clearly better.

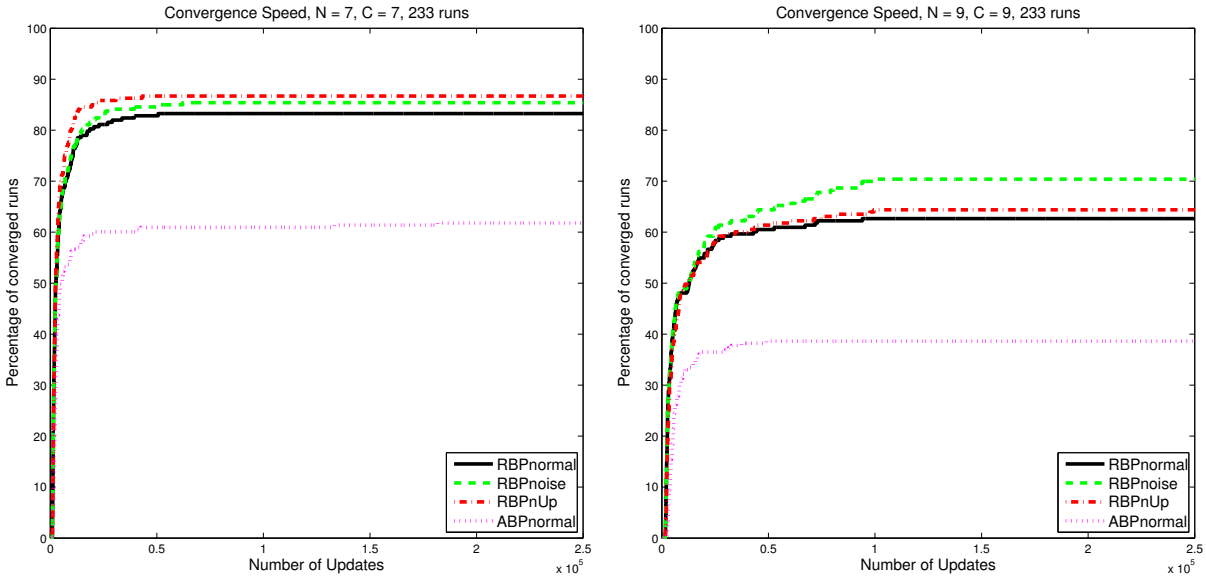


Figure 4.12: Convergence rate over number of updates for square grids with size $N = 7$ (left) and $N = 9$ (right).

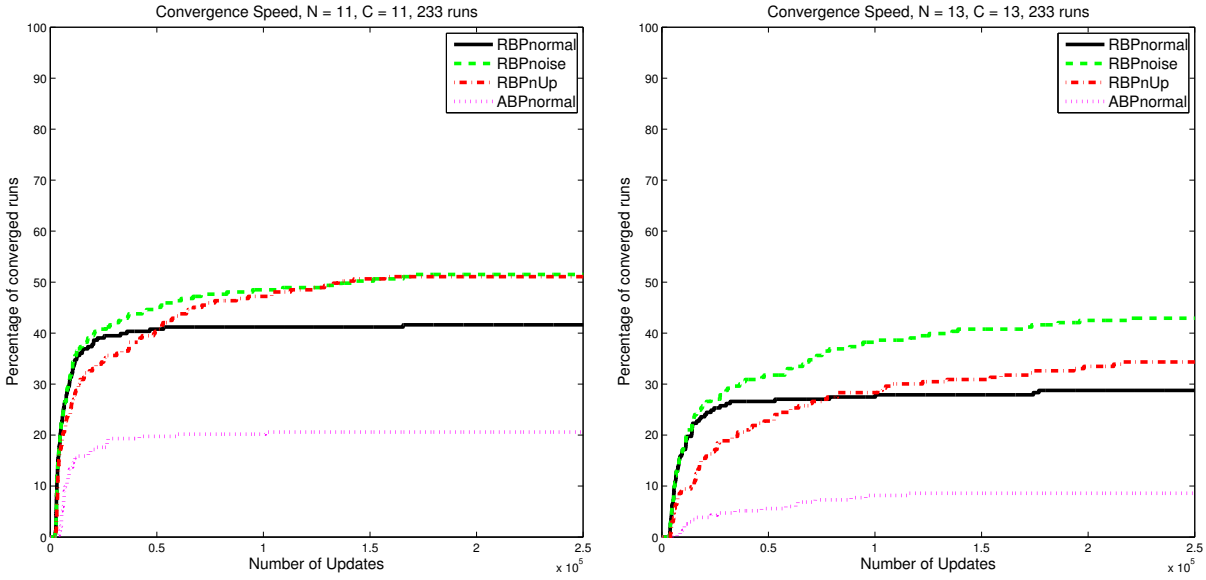


Figure 4.13: Convergence rate over number of updates for square grids with size $N = 11$ (left) and $N = 13$ (right).

4.3.3 Quality of Obtained Marginals

The resulting marginals were compared to the actual marginals by doing exact inference. The exact marginals were obtained using the Junction Tree Algorithm (JCT) via the C++ *libDAI* library created by Mooij [28]. In this section, several error measures were used to compare how well the different methods are suited to obtain good marginals that are close to the exact marginals. Again, only the results for the 11×11 grid will be discussed.

Mean Squared Error

From the experiments one has marginal probability distributions for each random variable X_i for $i = 1, \ldots, N$ denoted by $P(X_i)$. These represent the actual "true" distribution obtained from exact inference and $\hat{P}(X_i)$ representing the estimation of the actual distribution, obtained from the message passing algorithms.
The *mean square error* (MSE) is a measure that directly compares the actual probability values via

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \Big[P(X_i = 0) - \hat{P}(X_i = 0) \Big]^2 . \tag{4.3}$$

Since binary RVs are used, only one of the variable values is considered, the MSE for the other value is exactly the same.
In Fig. 4.14 the MSE of ABP against all improved RBP methods is depicted for each individual run. It is only plotted for the cases where both converge and where only the RBP methods converge, to see the improvements made by the methods. One can clearly see that the runs where only the RBP methods converge, give overall better estimates of the true marginals than ABP. This is indicated by the circle points lying below the median line. However, there are also several runs where ABP resulted in better estimates, even though it did not converge. This happens more so for RBPnoise, shown by all the circle points in the top left part of Fig. 4.14. In the worst case the MSE was above 0.1 even though convergence was reached.
To get a better contrast between the RBP and the improved methods, the same type of plots have been created for both methods and can be seen in Fig. 4.15. For the noise method, it is clearly visible that if both methods converge they usually result in the same error, because this means there were no oscillations detected. RBPnUp results in a higher number of runs were it alone converged that also have a lower error. Again, a few outlier runs can be seen, were RBP results in a lower error even though it did not converge. There are also a few runs were RBPnUp did not converge, but RBP did. Because of its rare occurrence this is not depicted.
Based on these results RBPnUp seems to be more promising overall.

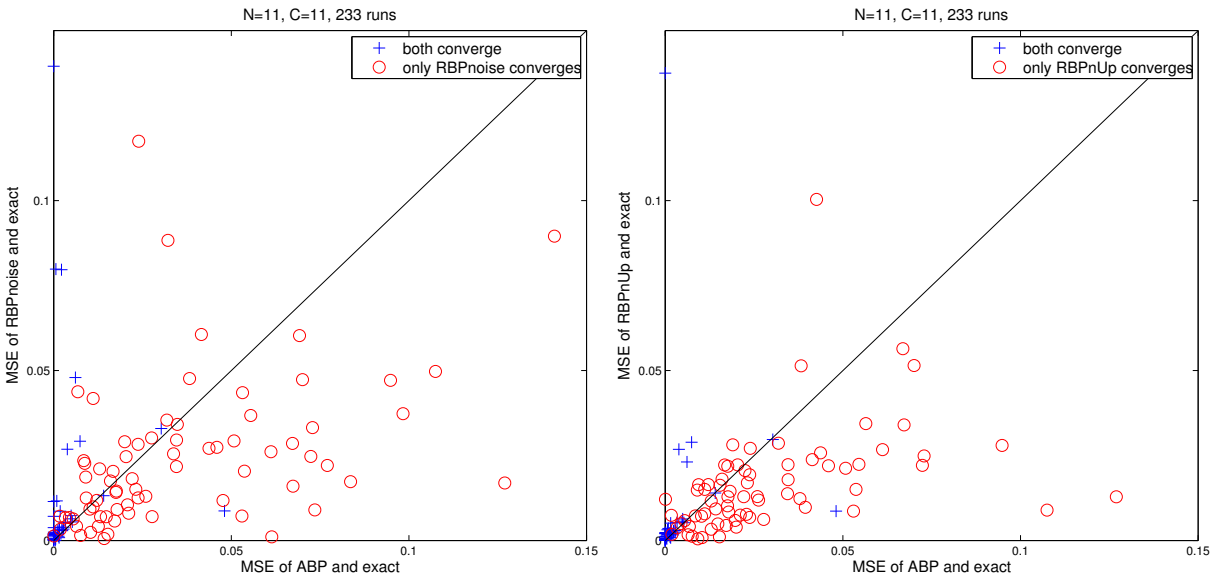


Figure 4.14: MSE of ABP against RBPnoise (left) and RBPnUp (right) for each run.

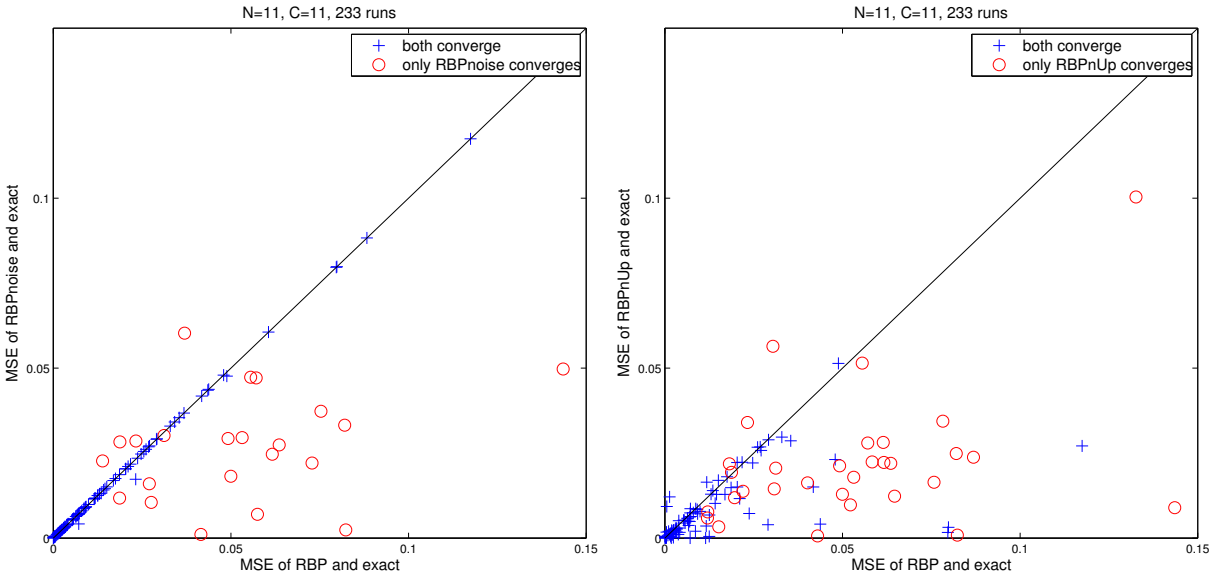


Figure 4.15: MSE of RBP against RBPnoise (left) and RBPnUp (right) for each run.

Kullback-Leibler divergence

The second considered measure is the *Kullback–Leibler divergence* (KL), which is a measure from information theory, that indicates the loss of information when an estimated distribution is used to approximate the actual distribution. This can be seen as measure for the difference between two probability distributions. For a marginal distribution over binary variables, as they are used in the experiments, the KL is defined by

$$D_{KL}(P(X_i) \,||\, \hat{P}(X_i)) = P(X_i = 0) \log \left(\frac{P(X_i = 0)}{\hat{P}(X_i = 0)} \right) + P(X_i = 1) \log \left(\frac{P(X_i = 1)}{\hat{P}(X_i = 1)} \right) \tag{4.4}$$

To compare the methods, $\hat{P}(X_i)$ will consistently denote the probability of the estimated marginals. That way the one-directional KL is sufficient to do an evaluation of the marginals.
In the same fashion as with the MSE, plots have been created to compare ABP with the RBP methods, which are depicted in Fig. 4.16.
Compared to the MSE plots, the results are mixed. Especially for RBPnoise, there are many runs where ABP resulted in a lower KL without even converging. However, for most runs the points are close to the median, so there is a marginal difference in terms of KL. Nonetheless, the improved RBP methods greatly help to reach convergence. It is notable that RBPnUp is better in helping runs that already have a smaller error to reach convergence, while RBPnoise manages to lead more difficult graphs with higher errors into convergence, visible by a wider spread of the markers. Overall, the graphs indicate that RBPnUp is the superior method.
Additional MSE and KL comparison plots for different grid sizes of this kind, can be found in Appendix C.

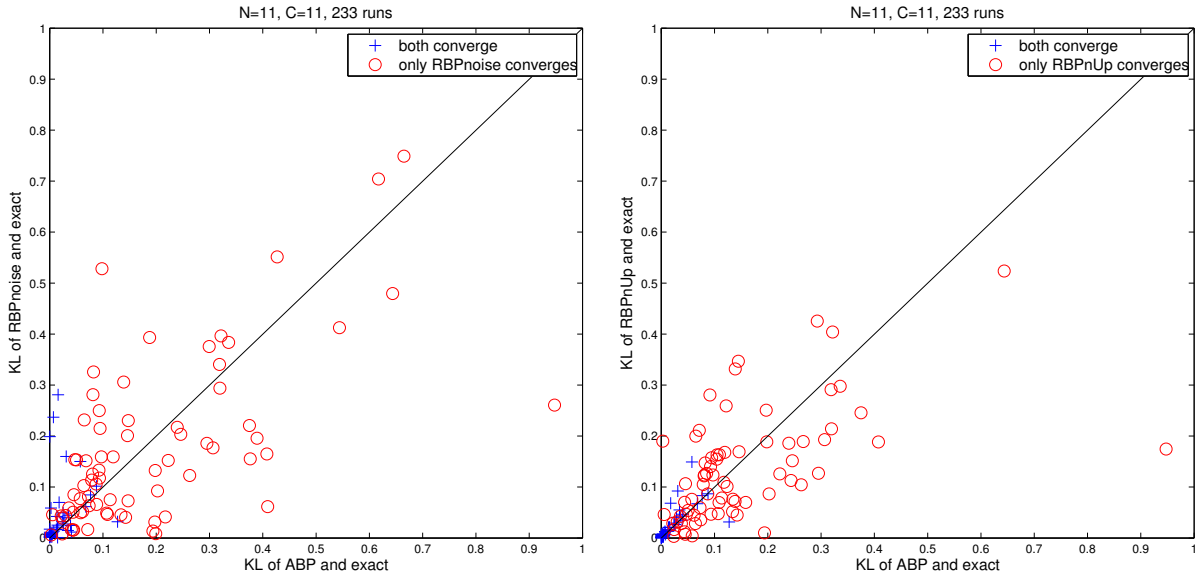


Figure 4.16: KL of ABP against RBPnoise (left) and RBPnUp (right) for each run.

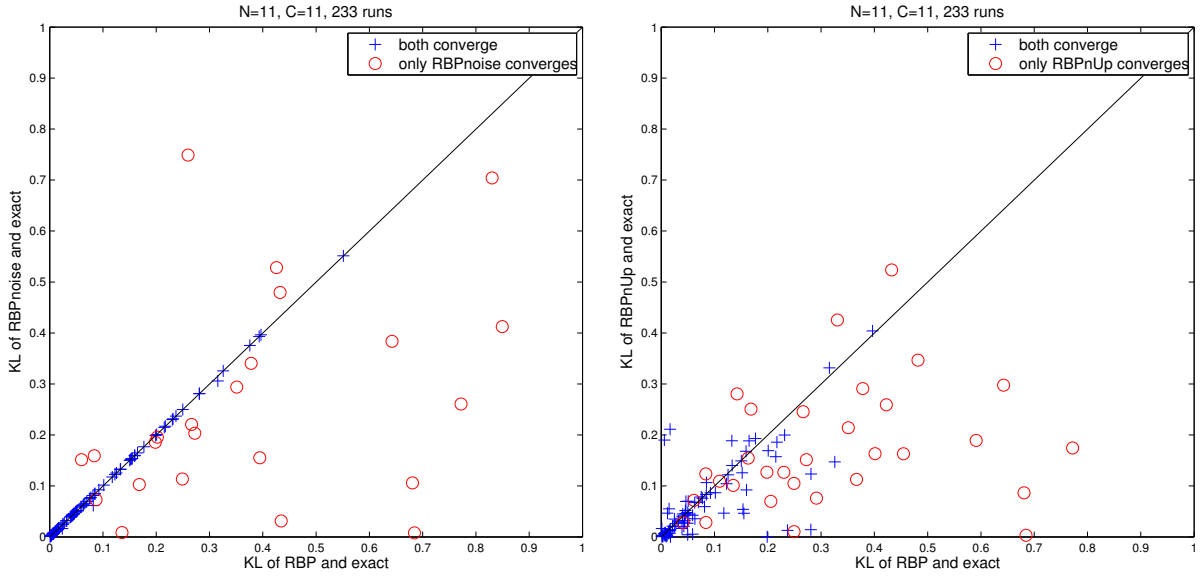


Figure 4.17: KL of RBP against RBPnoise (left) and RBPnUp (right) for each run.

4.3.4 Summary and discussion

All quality criteria for all methods and different grid sizes are listed in tables followed up by a discussion. First the quality criteria are described.

average MSE / KL (all): For each individual run the MSE / KL between the obtained marginals and the actual marginals was computed. Then these errors were averaged over all runs.

average MSE / KL (conv.): The same, but only for runs where the respective method converges. This is computed in order to rate the obtained results in case of convergence.

average MSE / KL (ABP conv.): Avg. MSE / KL again, but only for graphs where ABP converged. This is computed to compare the results of all RBP methods to common BP.

convergence rate While in previous sections, the evolution of the convergence was considered, here one looks solely on how many of the runs did actually converge.

In Tables 4.1-4.4 these measures are shown for square grids of $N = \{7, 9, 11, 13\}$ respectively.

N = 7	**ABP**	**RBP**	**RBPnoise**	**RBPweight**
average MSE (all)	0.0257	0.0205	0.0191	0.0165
average MSE (conv.)	0.0082	0.0104	0.0109	0.0101
average MSE (ABP conv.)	0.0082	0.0091	0.0075	0.0065
average KL (all)	0.1029	0.0982	0.0926	0.0797
average KL (conv.)	0.0348	0.0525	0.0573	0.0519
average KL (ABP conv.)	0.0348	0.04	0.0334	0.0296
convergence rate	61.8%	83.26%	85.41%	86.7%

Table 4.1: Quality criteria for 7×7 *Ising grids.*

N = 9	**ABP**	**RBP**	**RBPnoise**	**RBPweight**
average MSE (all)	0.0353	0.0311	0.0269	0.0243
average MSE (conv.)	0.0039	0.0128	0.013	0.0115
average MSE (ABP conv.)	0.0039	0.0095	0.0072	0.0056
average KL (all)	0.1684	0.1825	0.1593	0.1393
average KL (conv.)	0.0234	0.0838	0.0846	0.0721
average KL (ABP conv.)	0.0234	0.0593	0.0412	0.032
convergence rate	38.63%	62.66%	70.39%	64.38%

Table 4.2: Quality criteria for 9×9 *Ising grids.*

N = 11	**ABP**	**RBP**	**RBPnoise**	**RBPweight**
average MSE (all)	0.0415	0.0457	0.0375	0.0309
average MSE (conv.)	0.0053	0.0170	0.0193	0.0129
average MSE (ABP conv.)	0.0053	0.0131	0.0134	0.0076
average KL (all)	0.2272	0.292	0.2607	0.2257
average KL (conv.)	0.0313	0.1030	0.1318	0.1011
average KL (ABP conv.)	0.0313	0.0716	0.0806	0.0493
convergence rate	20.6%	41.63%	51.5%	51.07%

Table 4.3: Quality criteria for 11×11 *Ising grids.*

N = 13	**ABP**	**RBP**	**RBPnoise**	**RBPweight**
average MSE (all)	0.0563	0.0637	0.0551	0.0420
average MSE (conv.)	0.0143	0.0321	0.0316	0.0157
average MSE (ABP conv.)	0.0143	0.0373	0.0295	0.0141
average KL (all)	0.3291	0.4273	0.3889	0.3111
average KL (conv.)	0.0652	0.2089	0.2319	0.1189
average KL (ABP conv.)	0.0652	0.2851	0.2277	0.0941
convergence rate	8.58%	28.76%	42.92%	34.33%

Table 4.4: Quality criteria for 13×13 *Ising grids.*

Looking at these Tables, one can see again how ABP leads to a low convergence rate and hence marginals of bad quality in terms of MSE, averaged over all runs. Interestingly, for larger grids (Tables 4.3 and 4.4) the overall MSE of ABP is better than that of RBP, even though it reaches a very low convergence rate. Also, since the convergence rate is that low, the average MSE of the converged runs is the lowest among all methods. This is to be expected since ABP only converges in "easy" cases for graphs with fortunate factors. As indicated in Section 4.3.3, ABP gives better results on average for KL, where it achieves the lowest values, except for the smallest considered grid-size, since the other methods have very high convergence rates in this case.

While RBP significantly boosts the convergence rate by about 20% throughout all grid sizes, it results in the overall worst marginals for both MSE and KL. We see some exceptions for smaller grid sizes of $N = 7$ and $N = 9$ where at least a better average MSE than ABP can be achieved. Concerning only the runs where ABP converged, RBP gives also the worst results compared to RBPnoise and RBPnUp.

The usage of RBPnoise leads again to an increased convergence rate. The increase is higher the larger and thus more difficult the grids get. It leads to the highest convergence rate among all methods. Also better average MSE values can be obtained than with RBP, even better than ABP. The average KL can also be improved over RBP, however, as discussed in Section 4.3.3, overall, ABP achieves better KL results for the most difficult graphs. The biggest problem with RBPnoise comes when looking at the quality measures only for converged runs, where it reached the worst results. This is probably due to its noisy nature, that involves the most randomness in finding a solution. This could be seen as "forcing" the solution.

Using RBPnUp has a slightly lower convergence rate than RBPnoise, but by far the best marginals in terms of average MSE. Even when looking only at the converged runs, it comes very close to ABP, which is impressive, since the convergence rate is much higher. The same is true when only looking at runs where ABP converged. In terms of KL it results in the biggest improvements, achieving even better values than ABP for all grid sizes. The reason for this is probably that the update weighting results in a more evenly distributed update schedule, while still focusing on the messages that changed the most. Even though it did, as shown in Section 4.3.1, work worse than RBP in some runs, from all improved methods it yields the most reasonable results, giving the best marginals while still improving the convergence rate.

5 Conclusions and Future Work

In this thesis, the task of probabilistic inference in graphical models using methods of belief propagation (BP) was described. In particular, message passing algorithms for graphs with loops in the style of [1] were presented. Message scheduling methods that describe the order in which messages are updated were derived and dynamic schedules introduced by [18] were presented. This method called *residual belief propagation* (RBP) looks for each message on how much its value has changed, thereby defining the *message residual* and chooses the one with the highest residual. Our intention was to improve upon this method in terms of convergence rate. The occurring issue of message oscillation, where a set of messages repeatedly gets chosen for update because they assume the same residuals, was described and two methods to tackle this problem were introduced. One of which was the injection of noise into the message update called *RBPnoise*, an invasive method to suppress the oscillations. The second introduced method tried to only influence the scheduling by weighting each message residual with the number of updates it has received, where a higher number of updates reduces the importance of the message. We called this method *RBPnUp*.
Complete graphs with uniform Ising factors as well as square grid graphs with random Ising factors were used to show how RBP and the proposed adaptations thereof improved the convergence rate over generic BP while still giving good marginals.
In conclusion, it was shown that the improved methods further increase the convergence rate, while obtaining even better marginals, at the cost of run time. Thereby, RBPnoise improves convergence rate the most, while RBPnUp gives the best marginals.

5.1 Summary

In the following, a compact summary of the conclusion of this thesis is provided.

- For loopy graphs, using dynamic schedules with RBP, improves convergence rate while still giving reasonable results.
- Applied on difficult graphs, RBP tends to oscillate in the message update schedule. The proposed RBP methods were introduced to tackle this weaknesses.
- The invasive method of RBPnoise improves convergence rate the most, while giving the worst marginals.

- RBPnUp affects solely the schedule and results in a lower convergence rate than RBPnoise. However it still has a higher convergence rate than standard RBP and it gives the best marginals.
- Both new methods have a higher run-time than common BP and standard RBP. Standard RBP is the fastest of all BP methods.

5.2 Future Work

While the introduced methods, as well as standard RBP by [18] improve upon BP as stated, little to no research has been done to explain the origin of the improvements. Many publications relate the fixed points of BP to extrema of free energy functions such as Bethe free energy [13–15]. In that regard, there remains the question how the new found fixed points of dynamic message scheduling methods relate to the extrema of those free energy functions.
The idea of detecting oscillating messages as used in the RBPnoise method could be used in alternative ways. We had ideas such as locking the oscillating messages or using a random update schedule, that we considered but ultimately did not follow.
Since the proposed methods have a high computation time they could be further improved by applying the message approximations presented by Sutton and McCallum in [19] that was able to speed up RBP.
Lastly, an open question is the application of BP methods onto continuous variable models, as has been discussed by Sudderth et al [29].

Bibliography

[1] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, 2001.

[2] H.-A. Loeliger, "An introduction to factor graphs," *Signal Processing Magazine, IEEE*, vol. 21, no. 1, pp. 28–41, 2004.

[3] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[4] B. J. Frey and D. J. MacKay, "A revolution: Belief propagation in graphs with cycles," *Advances in neural information processing systems*, pp. 479–485, 1998.

[5] Y. Weiss, "Belief propagation and revision in networks with loops," 1997.

[6] ——, "Correctness of local probability propagation in graphical models with loops," *Neural computation*, vol. 12, no. 1, pp. 1–41, 2000.

[7] J. M. Mooij and H. J. Kappen, "Sufficient conditions for convergence of the sum–product algorithm," *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4422–4437, 2007.

[8] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[9] A. T. Ihler, J. Iii, and A. S. Willsky, "Loopy belief propagation: Convergence and effects of message errors," in *Journal of Machine Learning Research*, 2005, pp. 905–936.

[10] J. M. Mooij, B. Wemmenhove, B. Kappen, and T. Rizzo, "Loop corrected belief propagation," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 331–338.

[11] J. M. Mooij and H. J. Kappen, "Loop corrections for approximate inference on factor graphs," *The Journal of Machine Learning Research*, vol. 8, pp. 1113–1143, 2007.

[12] V. Gómez, J. M. Mooij, and H. J. Kappen, "Truncating the loop series expansion for belief propagation," *arXiv preprint cs/0612109*, 2006.

[13] T. Heskes, "On the uniqueness of loopy belief propagation fixed points," *Neural Computation*, vol. 16, no. 11, pp. 2379–2413, 2004.

[14] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring artificial intelligence in the new millennium*, vol. 8, pp. 236–239, 2003.

[15] ——, "Bethe free energy, kikuchi approximations, and belief propagation algorithms," *Advances in neural information processing systems*, vol. 13, 2001.

[16] J. S. Yedidia, W. T. Freeman, Y. Weiss *et al.*, "Generalized belief propagation," in *NIPS*, vol. 13, 2000, pp. 689–695.

[17] A. L. Yuille, "Cccp algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation," *Neural computation*, vol. 14, no. 7, pp. 1691–1722, 2002.

[18] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," *arXiv preprint arXiv:1206.6837*, 2012.

[19] C. Sutton and A. McCallum, "Improved dynamic schedules for belief propagation," *arXiv preprint arXiv:1206.5291*, 2012.

[20] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[21] F. Pernkopf, R. Peharz, and S. Tschiatschek, "Introduction to probabilistic graphical models," 2014.

[22] G. Shafer and V. Vovk, "The origins and legacy of kolmogorov's grundbegriffe," *The game-theoretic probability and finance project working paper*, vol. 4, p. 8, 2005.

[23] C. M. Bishop *et al.*, *Pattern recognition and machine learning*. springer New York, 2006, vol. 4, no. 4.

[24] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *International Journal of Approximate Reasoning*, vol. 15, no. 3, pp. 225–263, 1996.

[25] N. Taga and S. Mase, *On the convergence of belief propagation algorithm for stochastic networks with loops*. Citeseer, 2004.

[26] M. Mezard and A. Montanari, *Information, physics, and computation*. Oxford University Press, 2009.

[27] H. Nishimori, *Statistical physics of spin glasses and information processing*. Oxford University Press Oxford, 2001, vol. 187.

[28] J. M. Mooij, "libdai: A free and open source c++ library for discrete approximate inference in graphical models," *The Journal of Machine Learning Research*, vol. 11, pp. 2169–2173, 2010.

[29] E. B. Sudderth, A. T. Ihler, M. Isard, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation," *Communications of the ACM*, vol. 53, no. 10, pp. 95–103, 2010.

A
RBP noise algorithm

Algorithm 6: RBPnoise

input : Number of messages M
Set of factors $\{F_1, \ldots, F_D\}$
output: Converged stacked message vector $\mathbf{m}^{old}$

$\mathbf{m}^{old} \leftarrow$ uniform message array of size M;
$\mathbf{r} \leftarrow$ array of zeros with size M;
for i $\leftarrow$ 1 **to** M **do**
    $\mathbf{m}_i^{new} \leftarrow$ `computeUpdate`($\mathbf{m}_i^{old}$, $\mathbf{F}_{m_i}$);
    $r_i \leftarrow ||\mathbf{m}_i^{old} - \mathbf{m}_i^{new}||_\infty$
k $\leftarrow$ 1;
while k $<$ k_{max} **and** `convergenceCriterion`($\mathbf{m}^{old}$, $\mathbf{m}^{new}$) $=$ *false* **do**
    u $\leftarrow$ `index_max`(**r**);
    if `isOscillating`($\mathbf{m}_u^{old}$) **then**
        $\mathbf{m}_u^{old} \leftarrow \mathbf{m}_u^{new}$ + `RandomNormal`(μ, σ);
    else
        $\mathbf{m}_u^{old} \leftarrow \mathbf{m}_u^{new}$;
    $r_u \leftarrow 0$;
    for *all* j *where* $\mathbf{m}_u^{old} \in \mathcal{D}_j$ **do**
        $\mathbf{m}_j^{new} \leftarrow$ `computeUpdate`($\mathbf{m}_j^{old}$, $\mathbf{F}_{m_j}$);
        $r_j \leftarrow ||\mathbf{m}_j^{old} - \mathbf{m}_j^{new}||_\infty$
    k $\leftarrow$ k + 1

B

Additional marginal evolution plots

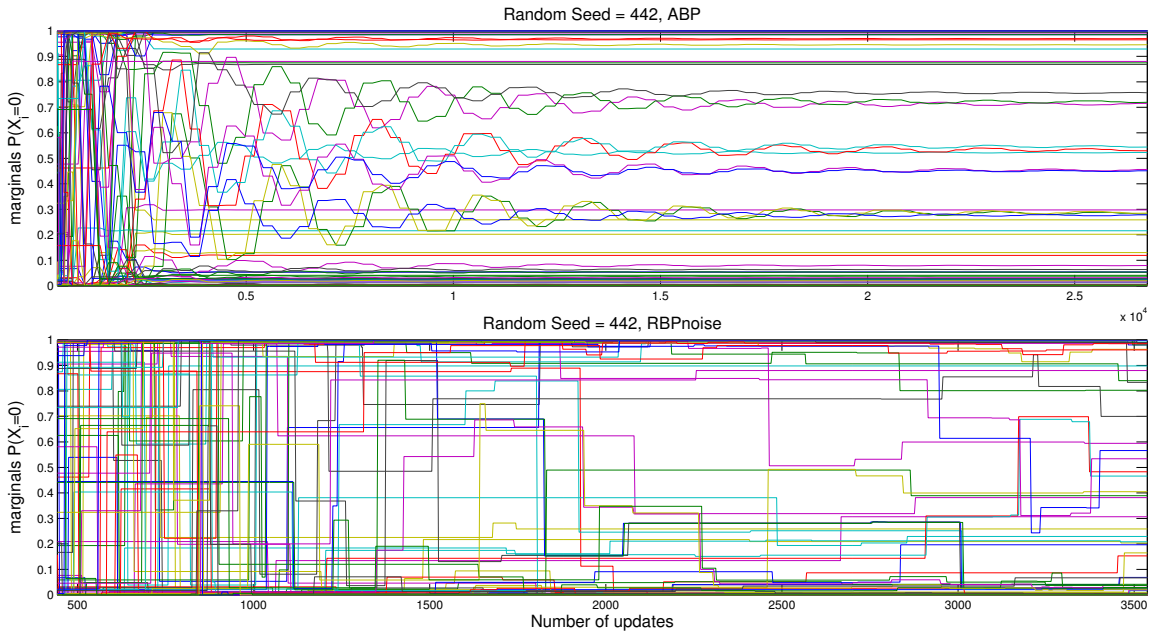


Figure B.1: RBP reaches convergence significantly faster than ABP.

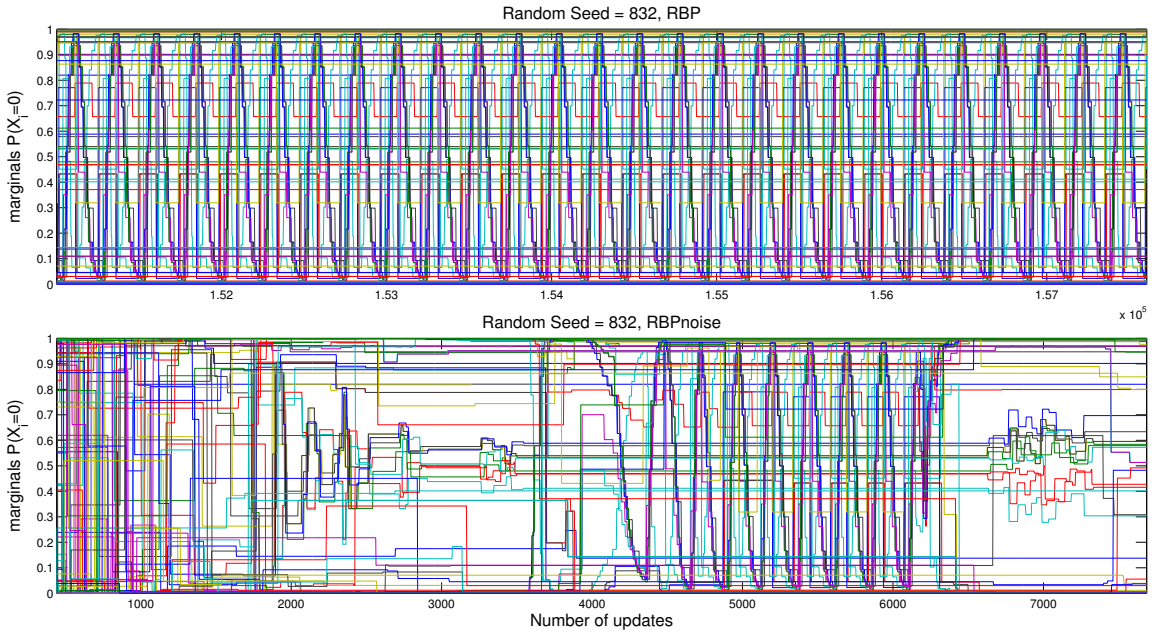


Figure B.2: Noise helps to reach convergence quickly.

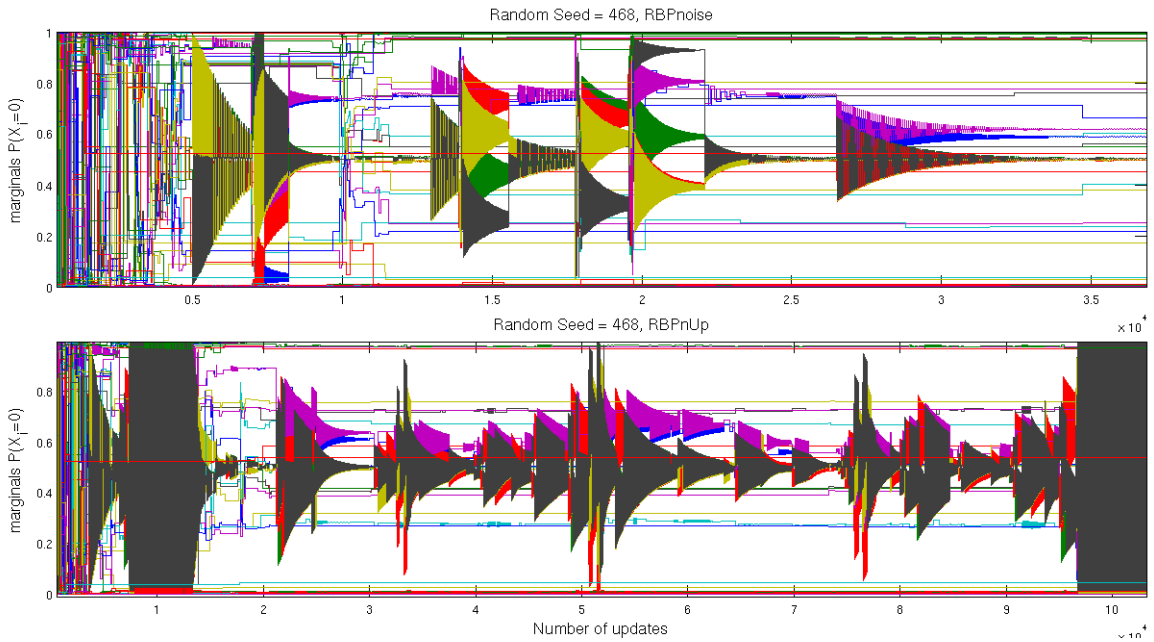


Figure B.3: Adding noise at specific times leads to convergence while weighting ultimately results in an unstable state.

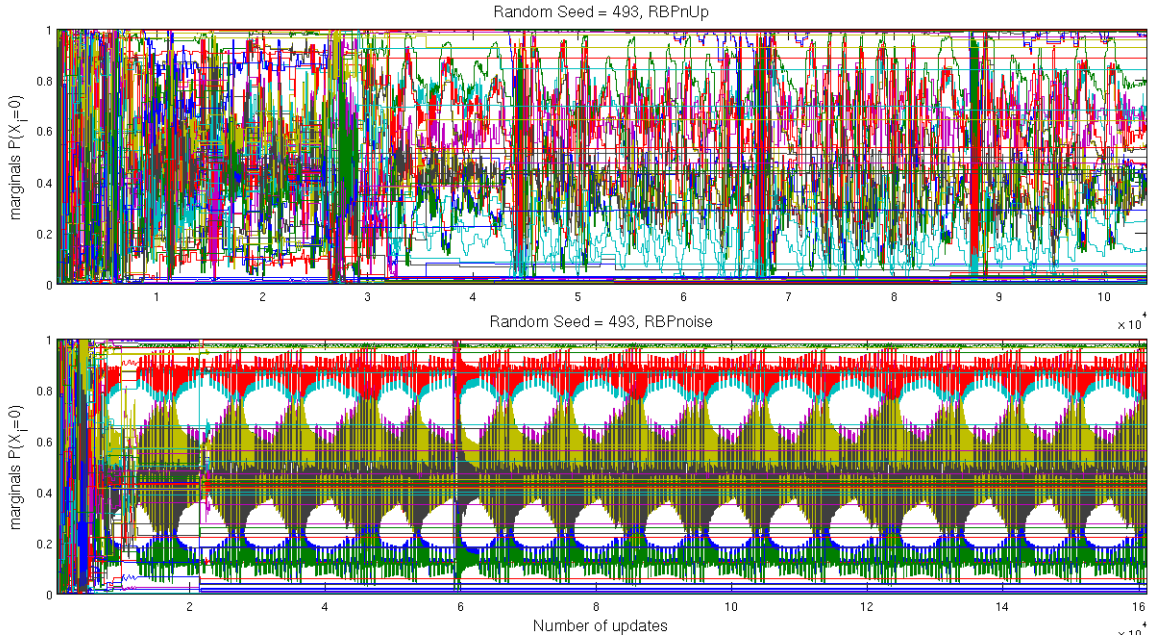


Figure B.4: Added noise leads to long term oscillations while weighting leads to convergence.

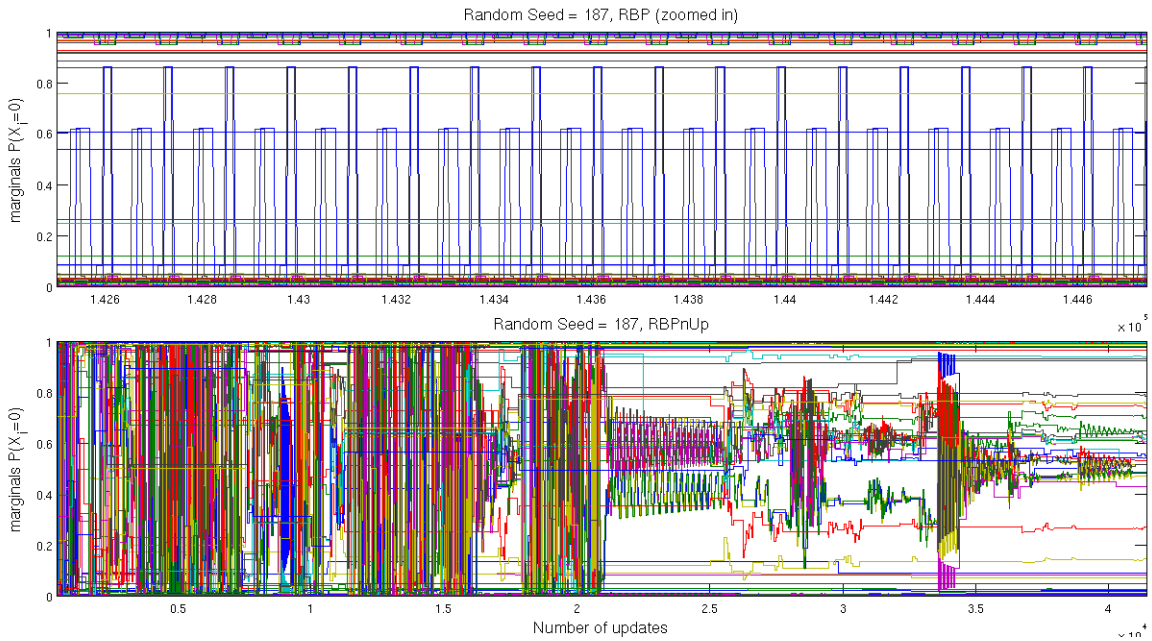


Figure B.5: Weighting results in a gradual approach of the converging state, while RBP is quickly trapped in oscillations.

C

Quality comparison plots for various grid sizes

C.1 MSE

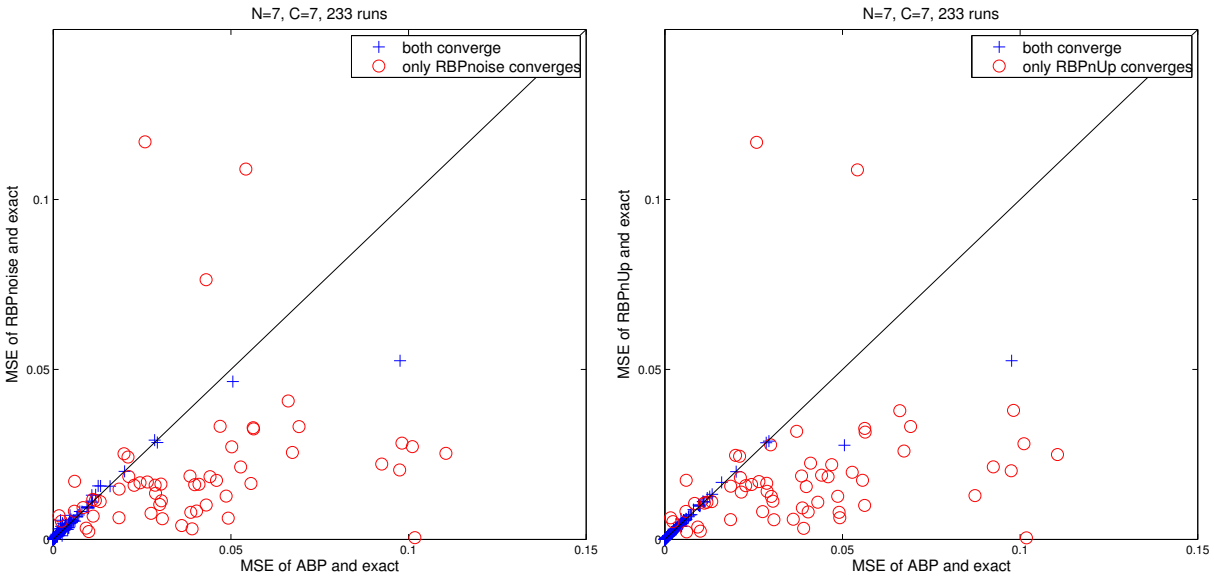


Figure C.1: MSE of ABP against RBP methods for each run.

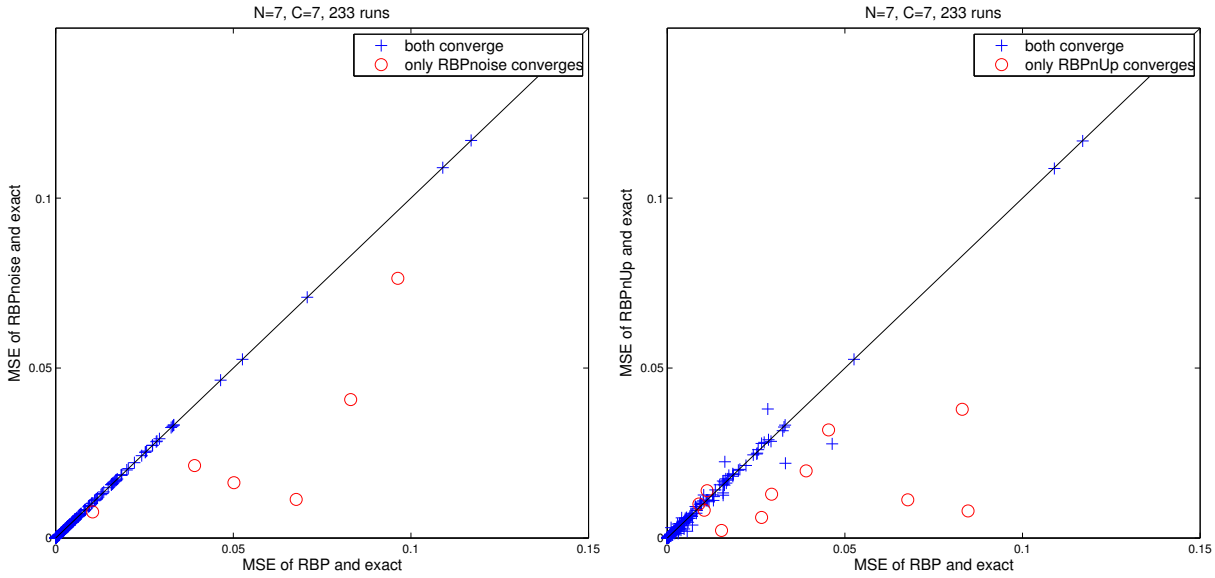


Figure C.2: MSE of RBP against improved methods for each run.

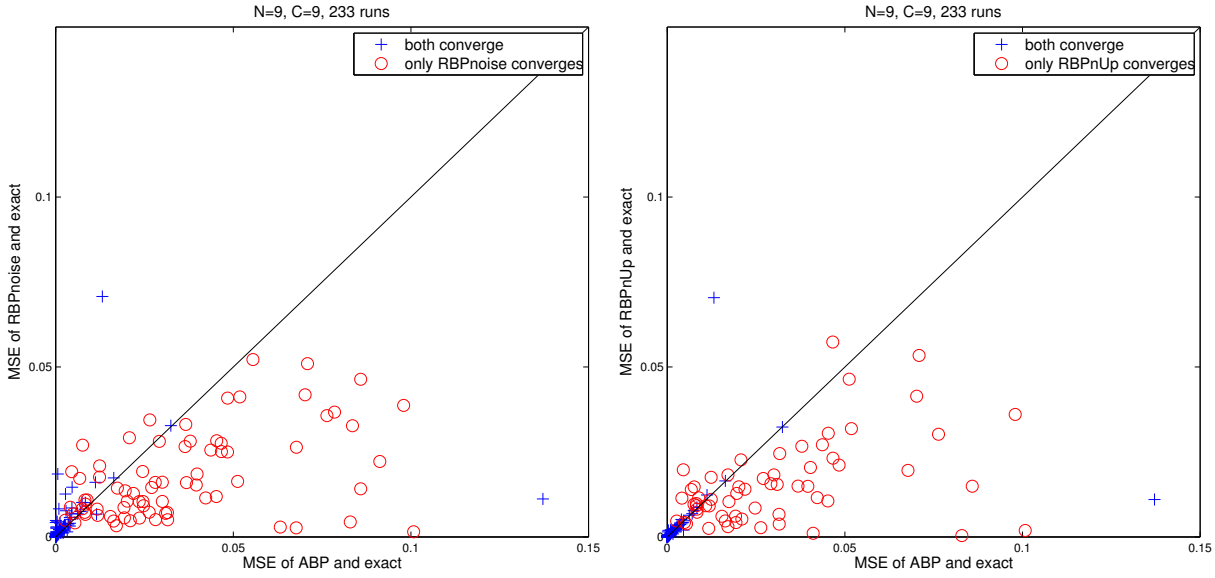


Figure C.3: MSE of ABP against RBP methods for each run.

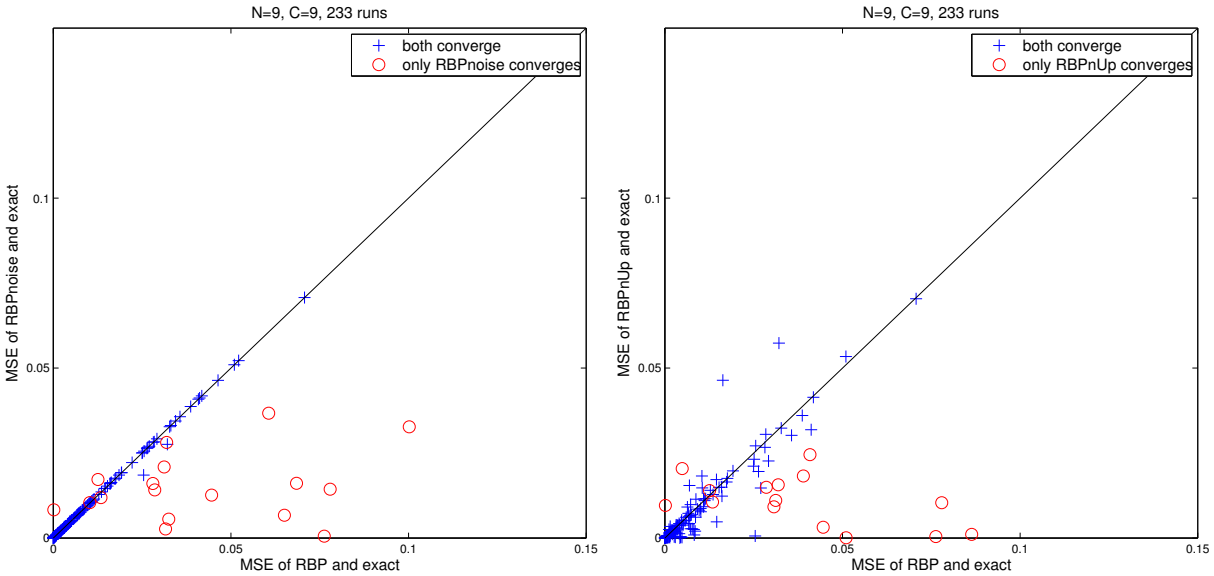


Figure C.4: MSE of RBP against improved methods for each run.

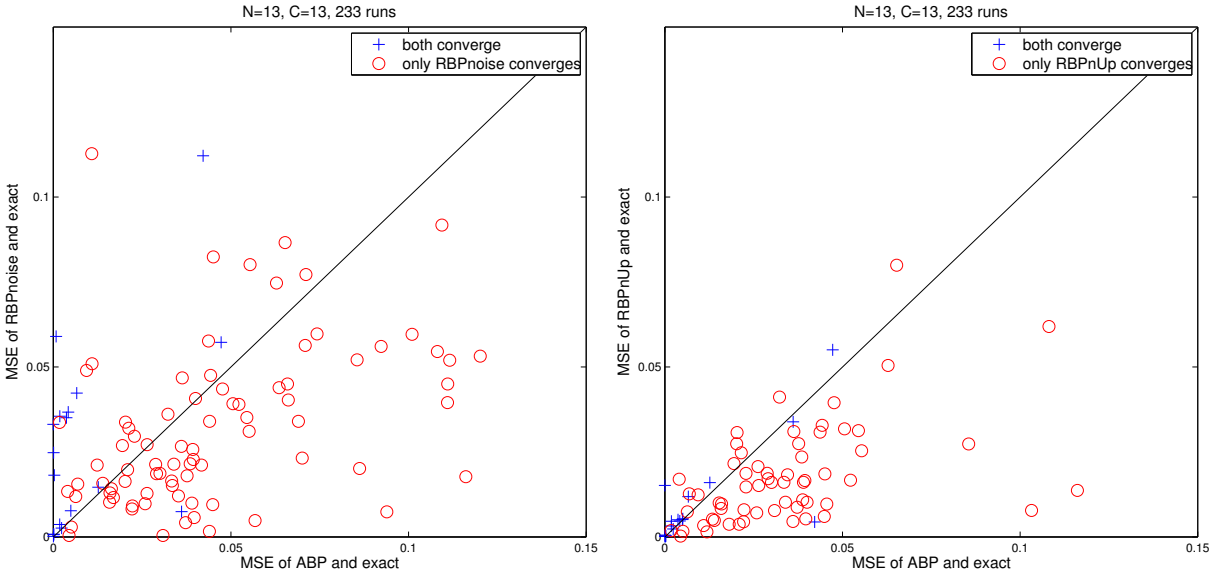


Figure C.5: MSE of ABP against RBP methods for each run.

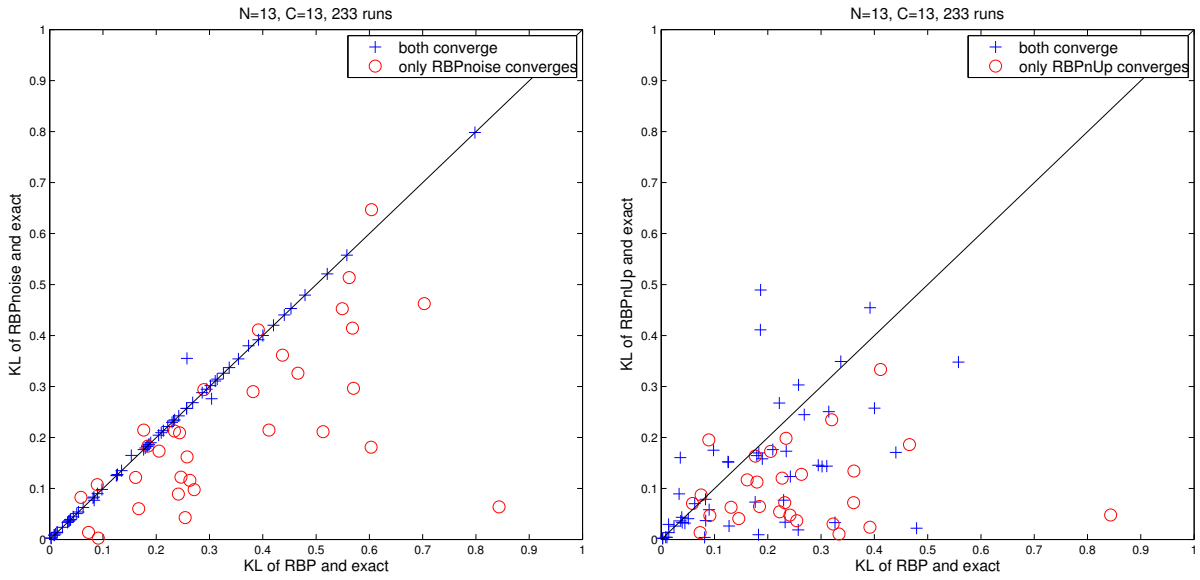


Figure C.6: MSE of RBP against improved methods for each run.

C.2 KL-divergence

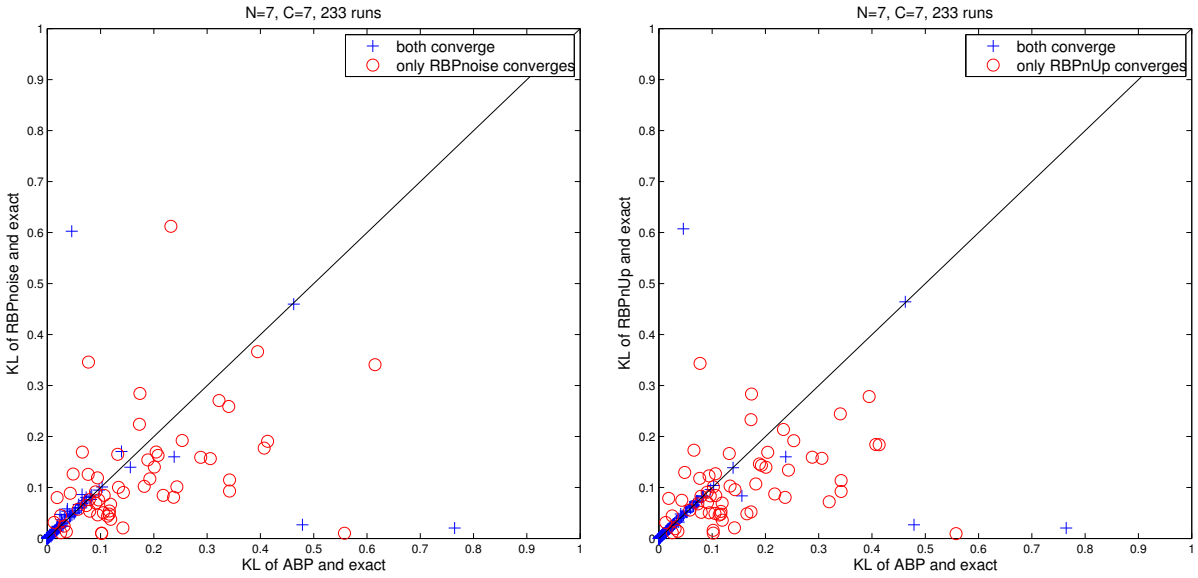


Figure C.7: KL-divergence of ABP against RBP methods for each run.

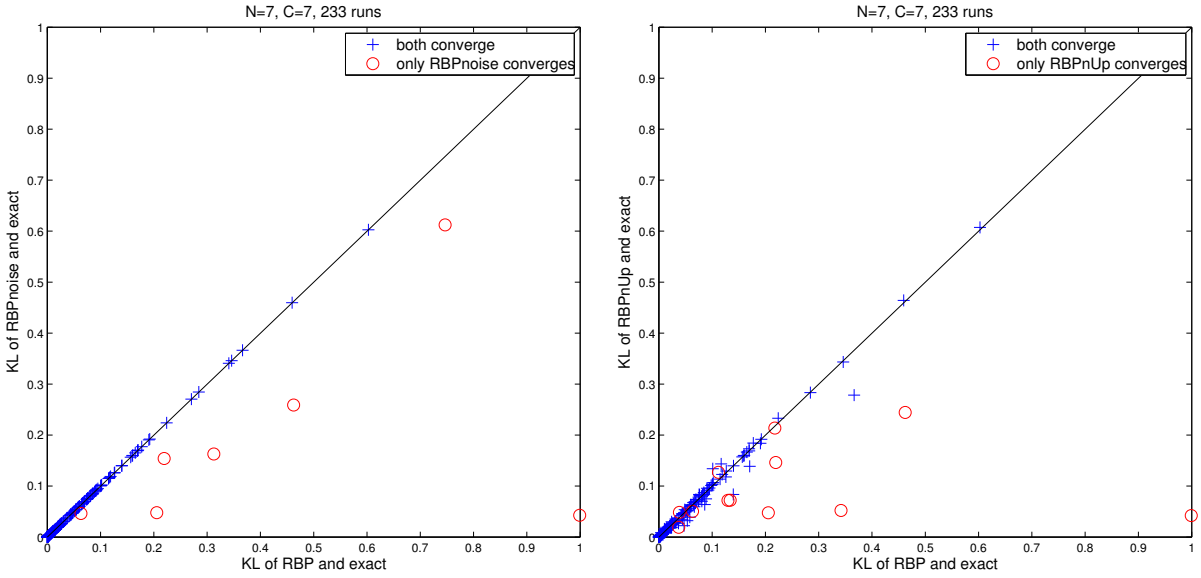


Figure C.8: KL-divergence of RBP against improved methods for each run.

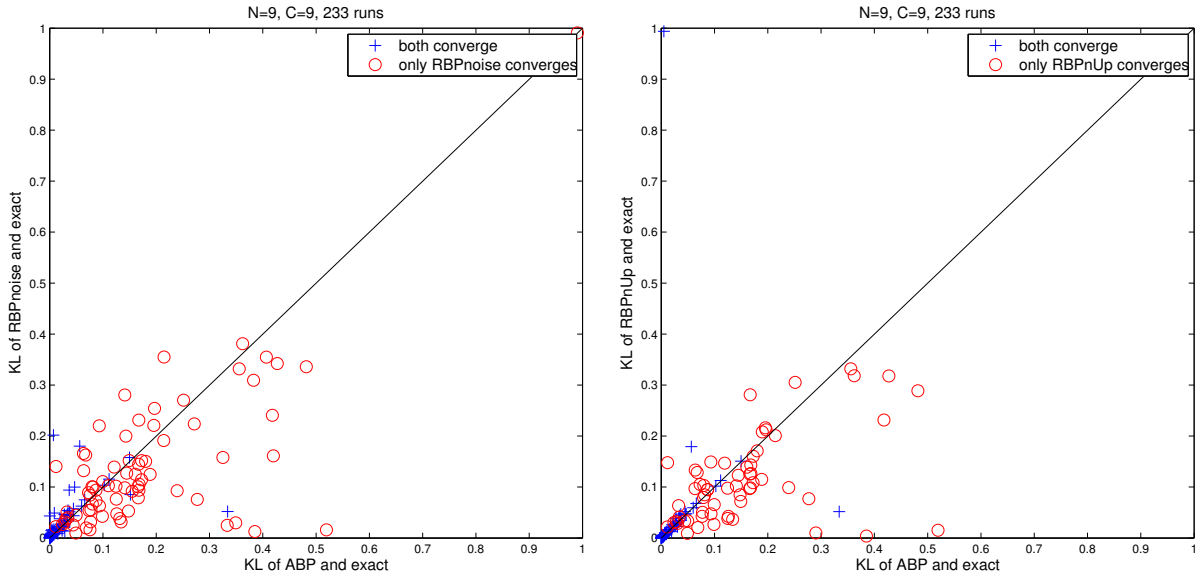


Figure C.9: KL-divergence of ABP against RBP methods for each run.

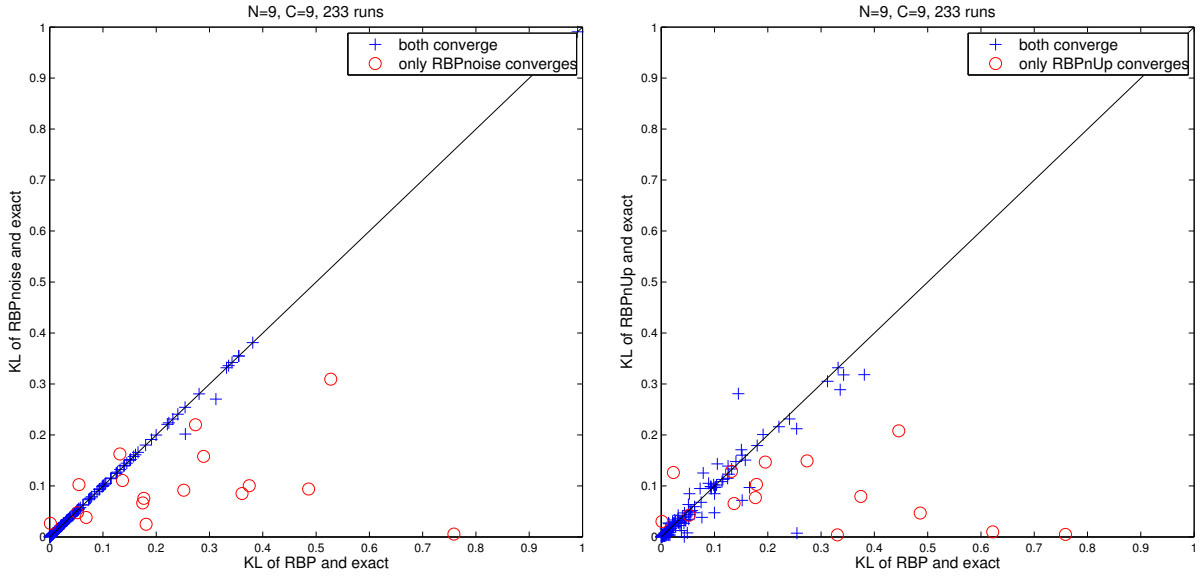


Figure C.10: KL-divergence of RBP against improved methods for each run.

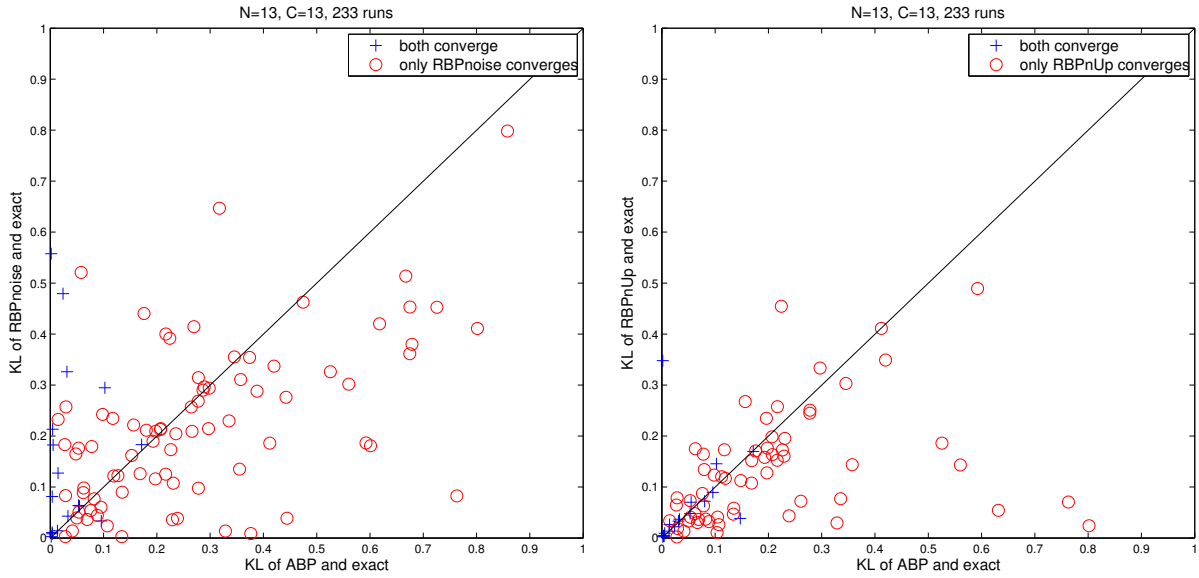


Figure C.11: KL-divergence of ABP against RBP methods for each run.

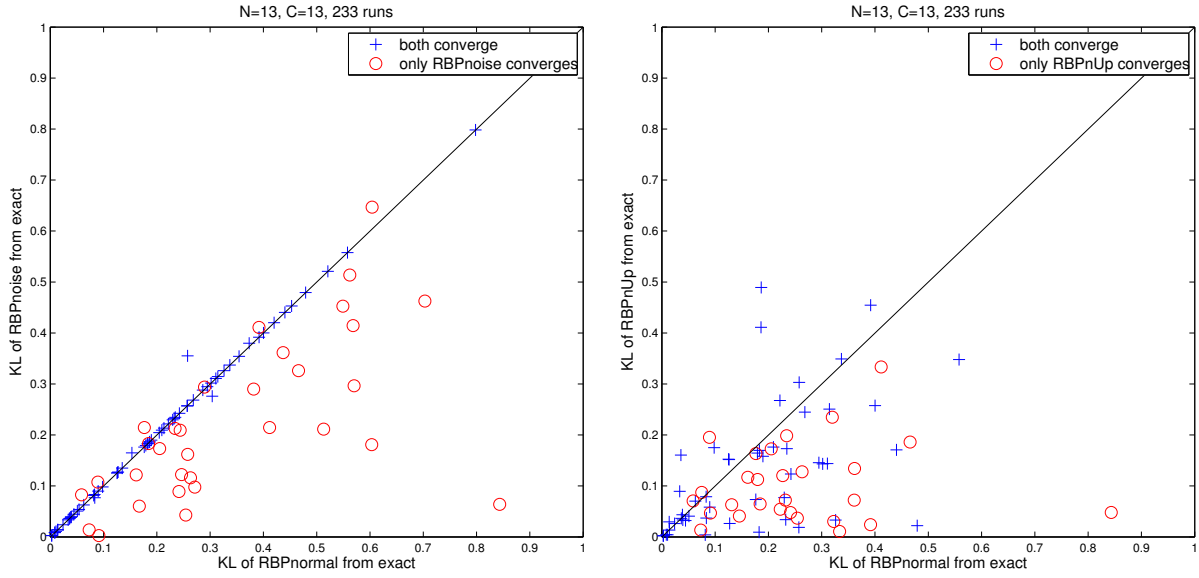


Figure C.12: KL-divergence of RBP against improved methods for each run.