New Message Passing Methods for Min-Max and Sum-Product Inference

by

Christopher Srinivasa

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto

Abstract

New Message Passing Methods for Min-Max and Sum-Product Inference

Christopher Srinivasa

Doctor of Philosophy

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering

University of Toronto

2018

Message passing algorithms powered by the distributive law of mathematics are efficient in finding approximate solutions to NP-hard inference problems. This thesis proposes new message passing algorithms to enable and improve performance in two important modes of inference. First is min-max inference which seeks the assignment that minimizes the worse-case outcome. Examples include well known NP-hard combinatorial problems such as min-max clustering, the asymmetric K-center problem, K-packing, the bottleneck traveling salesman problem, and makespan minimization. Within min-max inference two algorithms are proposed. The first is Min-Max Propagation (MMP), a min-max version of Belief Propagation (BP) obtained by noting that min and max operators satisfy the distributive law. With MMP, it is shown that for any high-order function which can be minimized in $\mathcal{O}(\omega)$, the message update can be computed using an efficient $\mathcal{O}(K(\omega + \log(K))$ procedure, where K is the number of variables. The second algorithm reduces min-max inference to a sequence of Constraint Satisfaction Problems (CSPs). Both approaches perform well on NP-hard combinatorial problems. The second mode of interest is sum-product inference. Here, loopy BP performs poorly when the underlying mass contains multiple disjoint modes, capturing only one mode when it reaches a fixed point. As such, a message passing procedure that attempts to model all the fixed points of BP by

defining a distribution over BP messages is considered; namely Survey Propagation (SP). Unfortunately SP is intractable beyond CSPs because, to perform general SP updates, one has to operate on distributions over a continuous domain. To efficiently extend the application of SP to marginalization in binary pairwise graphical models an approximation scheme is proposed. This scheme has $\mathcal{O}(DKlog(DK)\tau)$ complexity per iteration, where τ is the complexity of BP per iteration, D is the maximum node degree and K is a resolution constant controlling the approximations fidelity. Experiments show that this method can track many BP fixed points, achieving a high marginalization accuracy within a few iterations, in difficult settings where BP is often non-convergent and inaccurate.

To my parents, for their never ending love and support in my pursuit of knowledge.

Acknowledgements

To my advisor, Brendan. I still remember the min-max question on your exam which lead to so many of the developments in this thesis. As I look back on this journey, there are two components in particular that come to mind: confidence and freedom. It was you who gave me to the confidence to believe in myself and freedom to dig deep to find the answers when times were tough. I can't express enough how much these two ingredients were crucial in my success. Thank you. I'd also like to extend special mentions to Inmar and Siamak. Inmar, you were there in the beginning, when I started looking through those min-max factors, I remember us sitting down in various coffee shops and studying the problem. You were also a great inspiration as you were the last to work on the topic of factor graphs right before I joined the lab. Siamak, where do I start. I suppose your presentation and time spent in the lab one summer would be a good place. You're the one who brought me into the world of survey propagation and your guidance and insights since then on the many problems we tackled together, I will never forget. Thank you to both of you.

To my lab mates, Alice, Alireza, Andrew, Babak, Boyko, Hannes, Hui, Jeroen, Jimmy, Michael, Oren, and Rahul. As I write your names down, each and every one of you bring out a distinct memory whether research related or not where you helped me grow as a person. Thank you all so much.

To my friends, whom there are simply too many to mention. Specifically, I'd like to point to the many friendships made during my time at Knox College. My time and the people there truly shaped me as a person. I still stay in touch with many of you, you know who you are and you truly are friends for life.

To my partner, Lori. I still remember the early days at Knox. We've come so far since then. You believed in me in the hardest of times and have always been my

number one fan. Now as this journey reaches the end I can't wait to see what journey of life has in store for us.

To my parents, Louise and Rao. When I decided to pursue this degree, you were there like you always have been. Little did we know what we were getting ourselves into. It certainly was a marathon and not a sprint. But just like a marathon where there are water stations to refuel, you were there at every twist and turn of the winding road to help me refuel. This one's for you!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis addresses the issue of computational tractability. In today's world, the cardinality and complexity of mathematical problems continues to grow at a rapid pace. To solve these large problems, great advances in computational power have also followed. In this thesis we aim to show that fundamental mathematics still lie at the core of transforming a complex problem into one that is feasible to solve.

In the fields of mathematics and engineering, fundamental laws and equivalences are the cornerstone which enable efficient computation. This thesis is centered around one of these, namely the distributive law of mathematics. This law is pervasive in our daily lives. It is often introduced to us early in our educational ecosystems in its simplest form as follows, the key observation here being that the left-hand side of the equation requires three operations (two multiplications and one addition) while the right-hand side only requires two (one multiplication and one addition).

$$ab + ac = a(b + c) \tag{1.1}$$

In all chapters of this thesis we will constantly showcase how all contributions con-

stantly tie back into this law in one of its generalized forms.

From an algorithmic standpoint, the other aspect of this thesis is centered around NP-hard problems. NP-hard problems are those for which the best possible answer cannot be found within polynomial time. In such cases, many computational methods trade in this best possible solution for an approximate one which can be instead obtained in reasonable time. The distributive law in combination with iterative methods forms one of the most powerful mechanisms at minimizing this trade off: jointly minimizing the computational time and maximizing the quality of the solution.

In this thesis, we are interested in investigating the benefits of the distributive law in two areas of importance: min-max inference and sum-product inference. We leave the technical details as to how the distributive law of mathematics can be formulated as an iterative method to later chapters. Instead, the following sections motivate both these areas of research before showcasing the remaining structure and contributions of this thesis.

1.1 Min-max inference

When solving an engineering problem, a popular approach to any undertaking is one which minimizes the worst possible outcome for that venture. By adopting this safe strategy, one thereby guarantees that all other possible outcomes can only be better. In computational methods, min-max inference aims precisely to achieve this goal by finding the solution which minimizes the maximum cost described by a cost function which represents some problem.

One example is makespan minimization. As shown in Figure 1.1, the goal is to schedule a set of given tasks, each with a processing time, on machines which operate in parallel such that the operating time for the machine which operates the

longest, given the schedule, is minimized. This problem has wide applicability such as in the energy sector where the machines represent turbines and the tasks represent electrical power demands. Finding the optimal schedule to minimize the makespan is computationally intractable and thus many approximation algorithms exist to find suboptimal schedules.

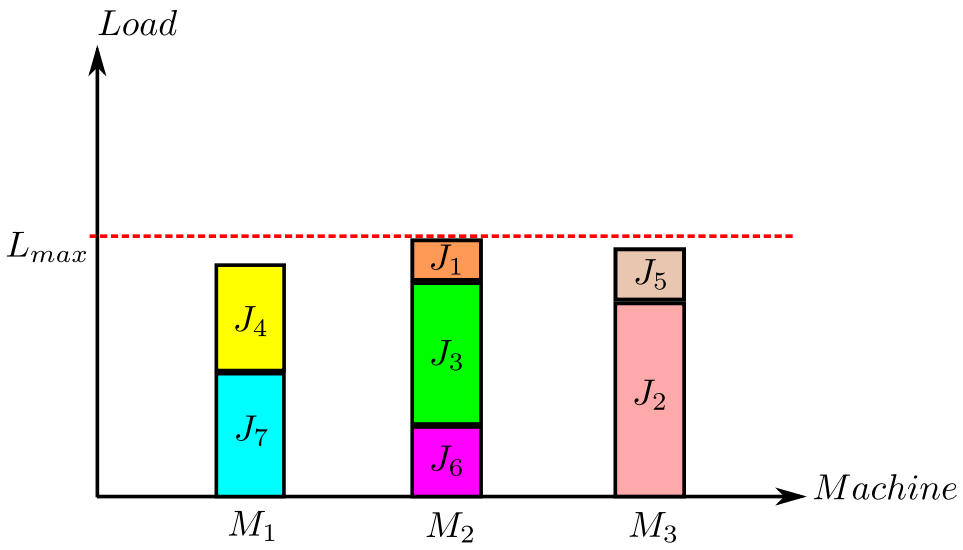


Figure 1.1: Makespan example

Other areas where min-max inference is valued include signal processing, molecular biology, statistical physics, economics, and business.

In this thesis, we aim to show that using iterative methods powered by the distributive law on min-max problems consistently yields solutions with lower costs than other min-max inference methods. Our line of thought in doing so is to cast the min-max problem in a framework that can be powered by the distributive law.

Our first objective is to investigate if much like the sum and product operators, min and max operators satisfy the distributive law. If so, the idea would be that any algorithm with sum and product message passing rules could instead be implemented with min and max operators.

Another avenue we wish to explore involves converting the min-max problem into a satisfiability problem and asking whether or not a particular bound can be satisfied. The reason for promise here is that satisfiability problems can be solved using logical OR and AND operators which are themselves subsets of the sum and product operators, as we will show in later chapters. Since we know that sum and product operators already obey the distributive law, then min-max problems could be efficiently solved as satisfiability problems. Thus, the question of whether or not min-max problems can be efficiently cast as a constraint satisfaction problem is of great interest in our quest of solving min-max problems on factors graph.

1.2 Sum-product inference

Perhaps the most prominent way in which the sum and product operators presented in the distributive law earlier in this chapter are used is in probabilistic inference. Here the product operator merges subsets of a joint mass and the sum operator then marginalizes over this joint mass.

Sum-product inference is often an NP-hard problem, thus involving iterative methods. When these methods arrive at a final answer, this result represents having summed over a non-zero mass. However, a problem arises when this mass for which we are trying to account is disjoint and multimodal. An example of this is shown in Figure 1.2. Here, the goal is to marginalize out x_i (or x_j) by summing over the three masses, ultimately projecting them onto the x_j (or x_i) axis, respectively. An iterative method involving the sum and product operators should achieve this goal. However, each mass in the figure corresponds to the quantity summed over by one iterative method to obtain its final result. Thus, the method, only being able to capture one mass at a time would miss out on two of the three masses, making the projection

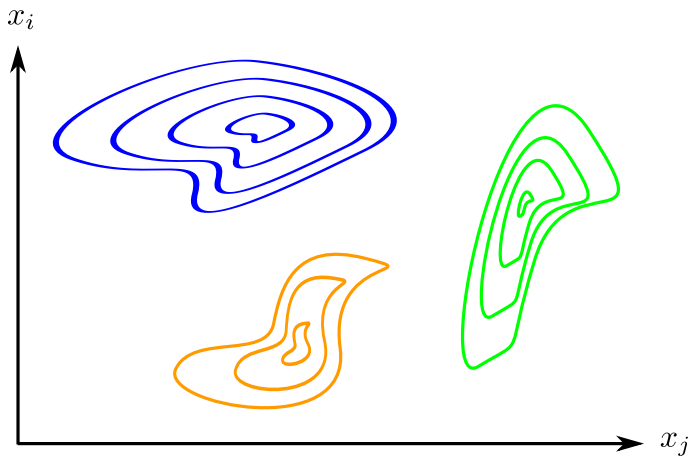


Figure 1.2: Joint probability of two variables containing multiple modes. Each colored mass corresponds to a mass over which an iterative method performed summation.

incorrect.

To address this issue, we suggest a probabilistic treatment of the modes themselves (*i.e.*, across the masses). This has much promise since, just like a probabilistic treatment within each mass, any probabilistic treatment of a quantity (*i.e.*, in this case of the modes) involves the sum and product operators, which we already showed satisfy the distributive law of mathematics.

1.3 Thesis outline

The remaining chapters of this thesis are structured as follows. We first review the required background material in chapter 2. Chapters 3 and 4 then showcase novel advances in the areas of min-max inference and sum-product inference. Specifically, the contributions to the field are:

- Chapter 3: A formulation of min-max problems as constraint satisfaction problems on factor graphs by noting that the min-max solution is always contained in one of the factors and leveraging that fact to build a tractable candidate

solution set, with results on random graphs and applications such as the bottleneck traveling salesman problem, min-max clustering, sphere packing, and the K-center problem. This work can be found in:

  - Min-Max Problems on Factor Graphs. Siamak Ravanbakhsh, Christopher Srinivasa, Brendan Frey, Russell Greiner. Proceedings of The 31st International Conference on Machine Learning, pp. 1035-1043, 2014.

  A derivation of mix-max message passing over factor graphs with polynomial complexity results for high order factors, a head-to-head comparison to the above mentioned approach, and results on makespan minimization. This work was published in:

  - Min-Max Propagation. Christopher Srinivasa, Inmar Givoni, Siamak Ravanbakhsh, Brendan Frey. Neural Information Processing Systems (NIPS), 2017

- Chapter 4: An efficient implementation of message passing using convolutional message updates for sum-product inference where the underlying mass has multiple modes, with results on random graphs. This work was published in:

  - Survey Propagation beyond Constraint Satisfaction Problems. Christopher Srinivasa, Siamak Ravanbakhsh, Brendan Frey. Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 286-295, 2016

We also propose other directions for future work related to both topics and conclude in chapter 5.

Chapter 2

Background

In this chapter we introduce the background material pertaining to the contributions in the thesis. For the advanced reader, some of these topics may be familiar. Below is a summary of each section such that they may choose at their discretion.

Section 2.1. We show how variable elimination can be done efficiently if a function factors as product of local functions which can be visualized as a factor graph. The procedure, belief propagation, is described as message passing on the factor graph. We review the sum-product belief propagation rules, and the resulting beliefs.For a loopy graph, loopy belief propagation is reviewed along with alternatives.

Section 2.2. We study loopy belief propagation with operators other than sum-product. We look at loopy max-sum (equivalently max-product) belief propagation and, as an example, its successes on clustering problems. We also study loopy logical or-and belief propagation to solve constraint satisfaction problems such as K-SAT.

Section 2.3. We review survey propagation, a more comprehensive version of belief propagation. We review how this method, developed for constraint satisfaction problems, addresses a phase transition where the landscape of the solution space fractions into clusters as the problem gets more difficult.

2.1 Message passing and factor graphs

Given a function $F(x)$ where $x = \{x_1, ..., x_N\}$ where, without loss of generality, x_i is discrete with $|x_i| = M$ and $x \in \mathcal{X} \triangleq \mathcal{X}_1 \times \mathcal{X}_i \times \mathcal{X}_N$, one often seeks to know the behaviour of F with regards to a subset of its variables β by summing over the remaining ones [10]:

$$m_\beta(x_\beta) = \sum_{x_{\setminus\beta}} F(x) \tag{2.1}$$

where $\setminus\beta$ denotes the set of variables excluding β. If the function represents a probability distribution, this corresponds to computing a marginal probability distribution.

This operation can be very computationally expensive. For example, if we wish to sum over all but one variable x_i (*i.e.*, $\beta = i$), the summation requires $\mathcal{O}\left(M^{N-1}\right)$ steps.

2.1.1 Factorized problem representation

If $F(x)$ factorizes as a combination of local functions $f_I(x_I)$, it can be visualized using a factor graph [49]. This graph is bipartite where each square node represents a local function f_I (also known as a factor) and each circle node represents a variable x_i. Each variable is connected via an edge to each factor in which it appears, as shown in the example of Figure 2.1.

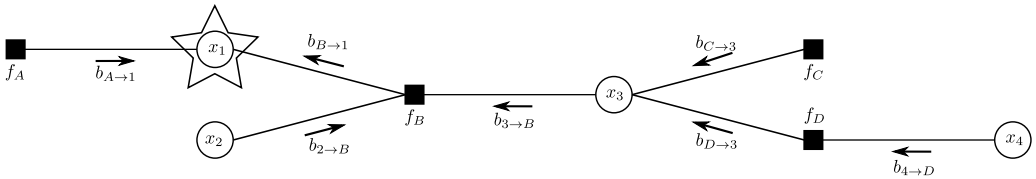


Figure 2.1: Variable elimination example. The factor graph represents the function $F(x_1, x_2, x_3, x_4) = f_A(x_1)f_B(x_1, x_2, x_3)f_C(x_3)f_D(x_3, x_4)$.

2.1.2 Variable elimination

More specifically, if $F(x)$ factorizes as a product of local functions (*i.e.*, $F(x) = \prod_I f_I(x_I)$), then the original summation problem can be solved efficiently by noting that the sum and product operations satisfy the distributive law of mathematics [2]. Using this property, we can first apply summation to the factors f_I (rather than the entire function $F(x)$) and then combine the results via the product operation. This process can be viewed as passing messages along the edges of a factor graph where the sum-product result of each subsection of the graph is interpreted as a message.

In Figure 2.1,we can obtain $m_1(x_1) = \sum_{x_2,x_3,x_4} F(x_1, x_2, x_3, x_4)$ efficiently by first noting that x_4 only occurs in f_D and applying its sum locally (where $b_{4\to D}(x_d) = 1$):

$$b_{D\to 3}(x_3) = \sum_{x_4} f_D(x_3, x_4) b_{4\to D}(x_4) \tag{2.2}$$

Note that the result $b_{D\to 3}(x_3)$ is a function of only x_3.Thus, we can combine all remaining quantities which are functions of only x_3 (where $b_{C\to 3}(x_3) = f_C(x_3)$):

$$b_{3\to B}(x_3) = b_{C\to 3}(x_3) b_{D\to 3}(x_3) \tag{2.3}$$

Since x_2 and x_3 only occur in f_B, we can now sum over both variables by applying the sum locally to f_B together with $b_{2\to B}(x_2)$ and $b_{3\to B}(x_3)$ (where $b_{2\to B}(x_2) = 1$):

$$b_{B\to 1}(x_1) = \sum_{x_2,x_3} f_B(x_1, x_2, x_3) b_{2\to B}(x_2) b_{3\to B}(x_3) \tag{2.4}$$

Combining quantities which are functions of only x_1 yields (where $b_{A\to 1}(x_1) = f_A(x_1)$):

$$m_1(x_1) = b_{A\to 1}(x_1) b_{B\to 1}(x_1) \tag{2.5}$$

2.1.3 Belief propagation

Suppose that in the example of Figure 2.1, we now want to sum over all variables except x_3. We would follow a similar procedure. However, it is not hard to see from Figure 2.1, that solving for x_3 would reuse messages $b_{D\to 3}(x_3)$ and $b_{C\to 3}(x_3)$ in its computations. Rather than re-computing those messages again, one efficient strategy would be to reuse the ones already computed when solving for x_1. This strategy leads us to the sum-product Belief Propagation (BP) algorithm. In BP, we compute the messages in both directions along each edge of the graph once, store them, and then use them accordingly when solving for the desired variables.

From the example of Figure 2.1 we can see that in general, for any factor graph where ∂i denotes all factors neighbouring variable i, and ∂I denotes all variables neighbouring factor I, two kinds of BP message updates exist.

Variable to factor message

Messages from variables to factors $b_{i\to I}$ are computed as:

$$b_{i\to I}(x_i) = \prod_{J\in\partial i\setminus I} b_{J\to i}(x_i) \tag{2.6}$$

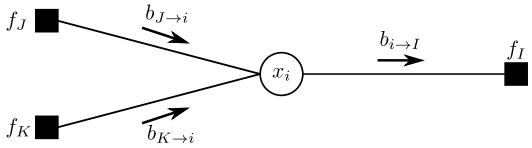


Figure 2.2: Variable-to-factor message.

Factor to variable message

Messages from factors to variable $b_{I\to i}$ are computed as:

$$b_{I\to i}(x_i) = \sum_{x_{\partial I\setminus i}} f_I(x_I) \prod_{j\in\partial I\setminus i} b_{j\to I}(x_j) \tag{2.7}$$

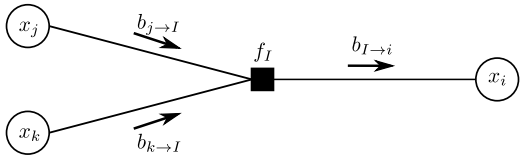


Figure 2.3: Factor-to-variable message.

One important question is what kind of messages should be sent out from leaf nodes in the graph. To understand this, we look at both message passing equations.

Initialization

From the variable to factor message equation, the initial message sent from a variable that is a leaf involves taking the product of zero incoming messages and so the product identity is needed. In other words, we need the identity I of $\prod$, s.t. $I \times a = a$ which is $I = 1$. Thus, we can initialize the messages sent from leaf variable nodes as:

$$b_{i\to I}(x_i) = 1 \tag{2.8}$$

Likewise, from the factor to variable message equation, we see that the message sent from a factor that is a leaf will involve summing over the empty set of incoming

messages. Thus, we can initialize the messages sent from leaf factor node as:

$$b_{I\to i}(x_i) = f_I(x_i) \tag{2.9}$$

General procedure

Note from these message updates that computing one outgoing message from any node in the graph requires all other incoming messages to that node. When the graph is a tree, this is done by choosing an arbitrary root node, passing messages to it from the leaves, and then back down to the leaves. From a graph with E edges, $2E$ messages need to be computed.

Once all messages have been passed on the graph, we can use them to compute quantities of interest. There are generally three quantities of main interest.

Variable beliefs

The results at each variable are known as beliefs and as we saw in the example, can be computed as:

$$b_i(x_i) = \prod_{I\in\partial i} b_{I\to i}(x_i) \tag{2.10}$$

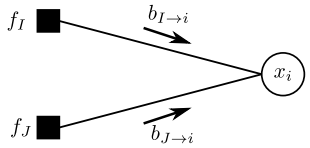


Figure 2.4: Message fusion at variable node.

Factor beliefs

We can also use the messages to compute beliefs at factor nodes as:

$$b_I(x_I) = f_I(x_I) \prod_{i \in \partial I} b_{i \to I}(x_i) \tag{2.11}$$

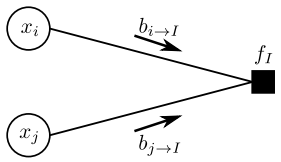


Figure 2.5: Message fusion at factor node.

When the graph is a tree, all beliefs are exactly equal to the marginal quantities of interest (*i.e.*, $b_\beta(x_\beta) = m_\beta(x_\beta)$) [10].

The last quantity of interest, the partition function, is discussed in section 2.3.

2.1.4 Loopy belief propagation

When the graph is not a tree, one way by which we can still perform message passing is by initializing all messages in the graph, sending them around the graph until they stop changing, and then using them to compute the beliefs [82]. This process is called Loopy BP (LBP) and the final stable set of messages b is called a fixed point of LBP. It is possible that more than one fixed point exists [38]. While LBP is known to give good results [25], it does not have any guarantees [63] and sometimes does not converge at all or simply converges to a fixed point which represents an incorrect answer [60] by itself. Four crucial aspects arise in LBP: Initialization, Normalization, Message Passing Schedules, and Damping.

- Initialization: Traditionally all messages in a loopy graph are initialized to be

uniform (*i.e.*, having the same value for both settings of the variable). However, since the set of messages at any iteration in LBP is computed from the set at the previous iteration [78], LBP is sensitive to the set of initial messages and different initializations can yield different answers. As such, some implementations consider multiple runs where the messages in each run are initialized randomly.

- Normalization: Since messages are passed around the graph iteratively, they become numerically unstable. To prevent this, the messages are normalized after being computed at each iteration. In the sum-product case, this is done by ensuring that each message sums to a specific constant (usually chosen to be one).

- Message Passing Schedules: The three most popular kinds of message passing schedules are: flooding, sequential linear, and sequential random. In a flooding schedule, all messages are sent in parallel from all factor-to-variable nodes, using a cached version of the variable-to-factor messages. These factor-to-variable messages are then cached themselves and used to compute all variable-to-factor messages in parallel, and so on. In a sequential schedule, only the messages leaving a chosen variable node are computed at a time and then used immediately for the following variables. If the schedule is linear, the variables are chosen in a linear order which remains the same throughout all iterations. If it is random, the variables are chosen in an order that is randomly permuted at each iteration.

- Damping: A popular method to help convergence of the messages on a loopy graph is damping. In damping, a newly computed message is averaged using a damping factor with the message on the same edge at the previous iteration.

2.1.5 Other variants of belief propagation

When a graph contains loops, we note that many other variants of BP exist which either solve for the marginals exactly or approximately.

- Exact methods: One exact method is the junction tree algorithm [50]. Here, the loopy graph is converted into a tree by grouping variables involved in a loop into new variable nodes. Regular BP is then run on the tree. The complexity of this method is highly dependent on the structure of the graph since highly loopy graphs will have new variable nodes with very high cardinality.

- Approximate cluster based methods: As we move towards approximate methods, in [3], [64], and [81] BP was generalized to consider clusters of variables and pass messages between those clusters to obtain better solutions at the cost of computational complexity. These algorithms can exactly account for short loops within these clusters. However, the cost of inference grows exponentially with the size of the largest cluster and therefore these techniques fail to account for large and influential loops in the factor graph.

- Loop correction methods: Then, the authors in [39], [47], [76], [79], and [84], introduced a class of BP algorithms which improve convergence, here at the cost of the quality of the solution. In practice, one often observes that when LBP converges, the quality of its results are better than such convergent alternatives. Lastly, we note that other methods based on loop correction also exist such as the ones in [57] and [59]. These come at an increased computational cost and still are not capable of capturing more than one BP fixed point at a time.

Other approximate BP methods include the loop series methods of [15, 32] and the cut-set conditioning techniques of [63, 19].

2.1.6 Sampling

The main alternative to message passing methods for inference are sampling algorithms. Specifically, when dealing with a function of many variables, one can compute marginals for the variables by first producing many samples for the joint variables and then looking at the sample count for each variable individually. The method for producing samples of the joint variables is called Gibbs sampling [30]. Here, one begins with a random setting for the variables and a sampling ordering for them. At each iteration a new sample for all variables is then produced. To do so, we travel through the variables in the defined order. At each stop, we obtain a marginal distribution for the variable by plugging into the function the values to which all other variables are currently set. Using this distribution, we draw a new value for the variable and immediately replace its previous value by this new one. Once all variables have been visited, the current setting of variables is considered to be a joint sample for the iteration and we can proceed to the next iteration where the process is repeated.

The main advantage of sampling is that it avoids any form of summation, which as we mentioned earlier, is expensive if computed jointly over many variables (*e.g.*, for the purpose of computing marginals). However, if the function has a complex landscape (*e.g.*, exhibits multiple modes of different height), one may need to produce many samples before getting a good idea of what the function looks like. This can be time consuming.

When running Gibbs sampling, three parameters must be set:

- Number of burn-in iterations: This refers to the number of iterations at the beginning of the sampling process which are completely ignored. This is because the sampling process begins with a random sample for the variables and takes a few iterations before producing samples which are completely influenced by

the function which we are trying to model.

- Sampling interval: This refers to the number of sampling iterations waited before keeping a sample. This is done to avoid keeping samples which are correlated in time which occurs if successive samples, or samples that are not spaced enough in time are kept.

- Number of iterations: Knowing that the total number of samples is given by the total number of iterations minus the burn in period and then divided by the sampling interval, the number of iterations is set to produce the desired number of samples.

2.2 Belief propagation with alternate operators

All inference concepts described so far in this chapter have involved a function which factorizes as *product* of local functions and a marginalization task which involves *summation* over its variables. We now aim to generalize these concepts. That is, we will now denote $\bigotimes$ as the expansion operator under which a function factorizes and $\bigoplus$ as the marginalization operator which operates over its variables. Staying true with the underlying concept of message passing algorithms, the marginalization and expansion operators must obey the distributive law of mathematics, that is $\bigoplus$ must distribute over $\bigotimes$. With these two new terms, an inference problem can be fully defined by specifying:

1. A set of variables w

2. A function $E(w)$

3. A marginalization operator $\bigoplus$ which operates over the variables w

4. An expansion operator $\bigotimes$ under which the function $E(w)$ factorizes as a set of local functions

For instance, the example of Figure 2.1 can now be seen as a specific instance of this new framework where:

1. The variable set $w = x = \{x_1, x_2, x_3, x_4\}$
2. The function $E(w) = F(x) = f_A(x_1) f_B(x_1, x_2, x_3) f_C(x_3) f_D(x_3, x_4)$
3. The marginalization operator $\bigoplus = \sum$
4. The expansion operator $\bigotimes = \prod$

With this mapping, we can also define *operator-generic* belief propagation equations by replacing $\sum$ with $\bigoplus$ and $\prod$ with $\bigotimes$ in Equations 2.6 to 2.11 to yield the operator-generic variable-to-factor, factor-to-variable, and beliefs respectively as shown below.

$$\nu_{i \to I}(x_i) = \bigotimes_{J \in \partial i \setminus I} \nu_{J \to i}(x_i) \tag{2.12}$$

$$\nu_{I \to i}(x_i) = \bigoplus_{x_{\partial I \setminus i}} f_I(x_I) \bigotimes_{j \in \partial I \setminus i} \nu_{j \to I}(x_j) \tag{2.13}$$

$$\nu_i(x_i) = \bigotimes_{I \in \partial i} \nu_{I \to i}(x_i) \tag{2.14}$$

$$\nu_I(x_I) = f_I(x_I) \bigotimes_{i \in \partial I} \nu_{i \to I}(x_i) \tag{2.15}$$

Note that we have used different symbols (*i.e.*, ν) to denote these operator-generic quantities. While these quantities are indeed equivalent to b when $\bigoplus = \sum$ and $\bigotimes = \prod$, our reasons for using distinct notation will become clear in chapter 4. Until that point, we suggest the reader to simply view b as a specific instance of ν when $\bigoplus = \sum$ and $\bigotimes = \prod$. In fact, the remainder of this thesis is structured in the same manner: we look at the operator-generic belief propagation equations and see how we can solve problems for different instantiations (*i.e.*, choices) of w, $E(w)$, $\bigoplus$, and $\bigotimes$ We start by using the rest of this section to highlight such successful instances in the existing literature.

Lastly, we note that, when dealing with a loopy graph, our choices for $\bigoplus$ and $\bigotimes$ may affect message passing initialization, normalization, and schedule when using LBP. We will address these aspects throughout the thesis every time new choices for the operators are made.

2.2.1 Optimization

BP is often used for discrete optimization because the min (or max) and sum (or product) operators also satisfy the distributive law of mathematics. For problems of this nature the goal is to find a joint setting of variables which minimizes (or maximizes) a given function. One example, is the problem of clustering data points to achieve concise data representations and thus minimize resource usage. Here, one aims to find the subset of data points, dubbed exemplars, which best concisely represents all other data points in the data set by minimizing the sum of all assignment costs when the remaining points are each assigned to an exemplar. Finding the optimal solution to this problem has been shown to be computationally intractable. Thus, many clustering methods are approximation algorithms which find suboptimal

solutions in reasonable computational time. The authors in [24] showed that message passing gives substantially better clusters than these algorithms and achieves this in much less computational time. To do this, the authors defined the factor graph shown in figure 2.6. In this factor graph, each binary variable x_{nm} indicates whether or not data point n has been assigned to cluster m.

To fully define the clustering problem, the factor graph has the following factors:

$$h_{nm}(x_{nm}) = -D_{nm}x_{nm} \tag{2.16}$$

where D_{nm} is the precomputed distance between data point n and m (when $n = m$ it is common to use the average of the distances to all other points),

$$f_m(x_{1m}, ..., x_{Nm}) = -\infty \mathbb{I}\left(x_{mm} == 0 \wedge \sum_{n \setminus m} x_{nm} > 0\right) \tag{2.17}$$

and,

$$g_n(x_{n1}, ..., x_{nM}) = -\infty \left(1 - \mathbb{I}\left(\sum_{m=1}^{N} x_{nm} = 1\right)\right) \tag{2.18}$$

where $\mathbb{I}(.)$ is the indicator function (*i.e.*, $\mathbb{I}(True) = 1$ and $\mathbb{I}(False) = 0$) and $\wedge$ is the boolean logical AND operator. Also by convention $\infty\ 0 \triangleq 0$.

In these functions, Equation 2.16 represent the assignment cost of point n to cluster m. For its part, Equation 2.17 enforces that a data point can only be assigned to a cluster if the data point acting as the exemplar for that cluster assigns itself to the cluster. Lastly, Equation 2.18 enforces that each data point is assigned to exactly one cluster.

For the factor graph in Figure 2.6, we can thus combine all factors to obtain:

$$J(x_{11},...,x_{NM}) = \sum_{n=1}^{N}\sum_{m=1}^{M} h_{nm}(x_{nm}) + \sum_{m=1}^{M} f_m(x_{1m},...,x_{Nm}) + \sum_{n=1}^{N} g_n(x_{n1},...,x_{nM}) \tag{2.19}$$

for which we seek to find the optimal values of the variables by maximizing it (i.e. $\max_{\mathcal{X}} J(x_{11},...,x_{NM})$).

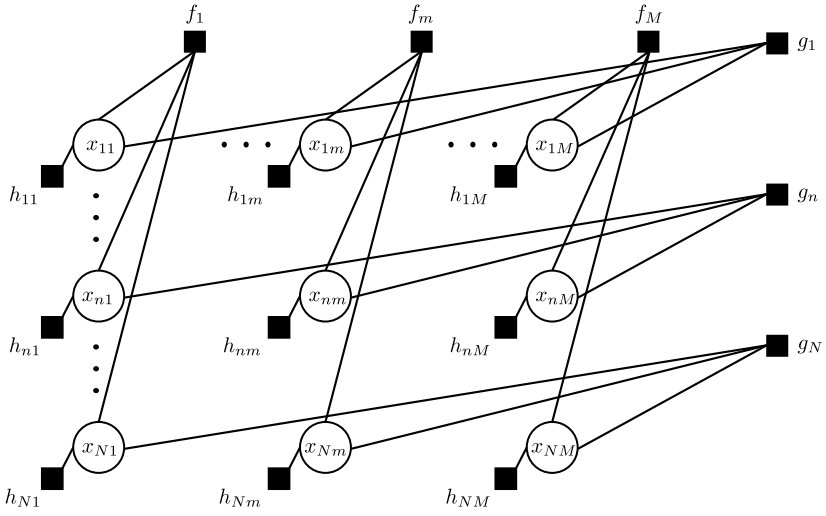


Figure 2.6: Factor graph for affinity propagation problem.

Within our operator-generic BP framework, this problem can be seen as a specific instance where:

1. The variable set $w = x = \{x_{11},...,x_{NM}\}$

2. The function $E(w) = J(x) = J(x_{11},...,x_{NM}) =$
   $\sum_{n=1}^{N}\sum_{m=1}^{M} h_{nm}(x_{nm}) + \sum_{m=1}^{M} f_m(x_{1m},...,x_{Nm}) + \sum_{n=1}^{N} g_n(x_{n1},...,x_{nM})$

3. The marginalization operator $\bigoplus = \max$

4. The expansion operator $\bigotimes = \sum$

It can thus be solved using LBP by making the appropriate operator substitutions in the operator-generic BP equations. It should be noted that when using this choice of operators, uniform message initialization corresponds to setting all values in the message to zeros since it is the identity of the sum operator (*i.e.*, $I + a = a$ when $I = 0$) and normalization corresponds to subtracting a constant from all values in the message. With regards to message passing schedules, any of the schedules mentioned in section can be used. It should noted that in addition to these, the authors considered unique schedules where messages are sent in parallel first up to vertical factors, then back down to the variables, then across to the horizontal factors and finally, back from those to the variables.

Other examples where BP has been used for optimization include maximum weight matching in graphs [6] and the Steiner tree problem [9].

In a probabilistic setting, we note that the use of max and product operators corresponds to performing Maximum a Posteriori (MAP) inference on a function which represents a joint probability distribution. We note that when we use max-product operators with LBP, the initialization, normalization, and schedules defined for sum-product operators apply as is.

Lastly, we wish to note that this kind of problem can also be solved as a Linear Program (LP). The authors in [80] do this where they treat the messages themselves as variables in the LP and use an off the shelf LP solver to perform the optimization. Their results, in which they compare this approach to actually passing the messages on the graph, show that while LP solvers are more general than message passing, the latter is able to exploit the structure of the problem (in this case the structure of the graph) to gain a speed up when performing the optimization and scales much better to larger problems.

2.2.2 Satisfiability

Factor graphs have also been used in the world of Constraint Satisfaction Problems (CSP) (*e.g.*, [56] and [85]) because logical OR (*i.e.*, $\vee$) and AND (*i.e.*, $\wedge$) operators are known to satisfy the distributive law. In such problems, the goal is to find a joint setting of the variable where the function evaluates to true (*i.e.*, outputs a value of one). A particular example of this is the K-SAT problem, as shown in Figure 2.7 for K=3. This factor graph contains 5 binary variables and 7 factors defined as follows:

$$c_1(x_1, x_3, x_4) = \neg x_1 \vee x_3 \vee x_4 \tag{2.20}$$

$$c_2(x_1, x_2, x_3) = x_1 \vee \neg x_2 \vee x_3 \tag{2.21}$$

$$c_3(x_2, x_3, x_4) = x_2 \vee x_3 \vee x_4 \tag{2.22}$$

$$c_4(x_2, x_3, x_5) = \neg x_2 \vee \neg x_3 \vee x_5 \tag{2.23}$$

$$c_5(x_1, x_3, x_5) = x_1 \vee x_3 \vee \neg x_5 \tag{2.24}$$

$$c_6(x_3, x_4, x_5) = x_3 \vee x_4 \vee x_5 \tag{2.25}$$

$$c_7(x_1, x_4, x_5) = x_1 \vee \neg x_4 \vee \neg x_5 \tag{2.26}$$

to form the following cost function

$$\begin{aligned} J(x_1, ..., x_7) =& c_1(x_1, x_3, x_4) \wedge c_2(x_1, x_2, x_3) \wedge c_3(x_2, x_3, x_4) \wedge c_4(x_2, x_3, x_5) \\ & \wedge c_5(x_1, x_3, x_5) \wedge c_6(x_3, x_4, x_5) \wedge c_7(x_1, x_4, x_5) \end{aligned} \tag{2.27}$$

for which we seek at least one satisfiable assignment of the variables by logically ORing over it (i.e. $\vee_{\mathcal{X}} J(x_1, ..., x_7)$).

Here each factor represents a clause which evaluates to true (*i.e.*, one) if one or

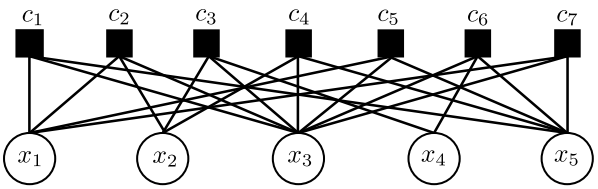


Figure 2.7: Factor graph for 3-SAT problem.

more of its (possibly negated, *i.e.*, $\neg x_i$) variables is true. Since, K=3 each clause contains exactly 3 variables. The clauses are then ANDed together meaning that they must all be true for the variable assignment to be satisfactory.

Now going back to our operator-generic BP framework, this problem can be seen as a specific instance where:

1. The variable set $w = x = \{x_1, ..., x_7\}$
2. The function $E(w) = J(x) = J(x_1, ..., x_7) = c_1(x_1, x_3, x_4) \wedge c_2(x_1, x_2, x_3) \wedge c_3(x_2, x_3, x_4) \wedge c_4(x_2, x_3, x_5) \wedge c_5(x_1, x_3, x_5) \wedge c_6(x_3, x_4, x_5) \wedge c_7(x_1, x_4, x_5)$
3. The marginalization operator $\bigoplus = \vee$
4. The expansion operator $\bigotimes = \wedge$

Once again, LBP, with the appropriate operator substitutions, can used to solve it. With this choice of operators, uniform message initialization corresponds to setting all values in the message to ones since it is the identity of the AND operator (*i.e.*, $I \wedge A = A$ when $I = 1$). We also note that no normalization is needed. This is because every clause always outputs either one or zero and that every message is initially all ones. Thus, logically ANDing and ORing these values together will always result in an output of zero or one. With regards to message passing schedules, any of the schedules mentioned in section can be used.

As a side note we mention that, the OR and AND operators are restricted cases of the sum and product operators when values are restricted to being zero or one, with the summation being capped at one.

2.3 Partition function and survey propagation

In section 2.1.3, we showed how we can compute beliefs at variable nodes and factor nodes for sum-product BP. If we further add up the values within any of these beliefs, the resulting quantity represents having summed over all the variables of the original function. Using our sum-product BP messages b from section 2.1.3 and defining functions for the beliefs as:

$$s_i(\hat{b}_{\partial i \to i})(x_i) = \prod_{I \in \partial i} \hat{b}_{I \to i}(x_i) \tag{2.28}$$

$$s_I(\hat{b}_{\partial I \to I})(x_I) = f_I(x_I) \prod_{i \in \partial I} \hat{b}_{i \to I}(x_i) \tag{2.29}$$

and

$$s_{i \leftrightarrow I}(\hat{b}_{i \to I}, \hat{b}_{I \to i})(x_i) = \hat{b}_{i \to I}(x_i) \hat{b}_{I \to i}(x_i) \tag{2.30}$$

,where $\hat{b}$ is a normalized version of b which sums to one, appendix A[1] shows that this quantity can also be computed as (where *e.g.*, $s_I(\hat{b}_{\partial I \to I})(\emptyset) = \sum_{x_I} s_i(\hat{b}_{\partial I \to I})(x_I)$):

$$G(\hat{b})(\emptyset) = \prod_I s_I(\hat{b}_{\partial I \to I})(\emptyset) \prod_i s_i(\hat{b}_{\partial i \to i})(\emptyset) \left(\prod_{i, I \in \partial i} s_{i \leftrightarrow I}(\hat{b}_{i \to I}, \hat{b}_{I \to i})(\emptyset) \right)^{-1} \tag{2.31}$$

When the graph is a tree, Equation 2.31 is exactly equivalent to the total mass represented by the function. This total mass is known as the partition function. When the graph contains loops, Equation 2.31 is an approximation to the partition function. If we run LBP until we reach a fixed point, we can plug the resulting set of messages $\hat{b}$ into Equation 2.31 and treat its value as a weight for the corresponding fixed point.

Notice that each of the terms in Equation 2.31 already contains a summation over the BP variable space. Thus, to capture all fixed points, a marginalization problem can now be viewed as marginalizing once over the variables of interest for a particular fixed point, and then a second time over all BP fixed points (i.e. $\sum_{\hat{b}} G(\hat{b})(\emptyset)$).

The algorithm capable of efficiently performing the second summation across all BP fixed points is known as Survey Propagation (SP). To obtain the SP updates, we first note that Equation 2.31 itself factors as a product of local functions, implying that it can itself be represented by a factor graph as shown in [14], [48], [53], [55], and [68]. More specifically, we see that it can be expressed as the product of three types of factors:

Variable-type factors. These contain the component of G for each variable node in the original factor graph $s_i(\hat{b}_{\partial i \to i})(\emptyset)$.

Factor-type factors. These contain the component of G for each factor node

[1]Please note that the material in appendix A is obtained from [68]. It is reproduced in appendix A in order to give completion to the derivations performed in this chapter.

in the original factor graph $s_I(\hat{b}_{\partial I \to I})(\emptyset)$.

Edge-type factors. These contain the component of G for each edge in the original factor graph $s_{i \leftrightarrow I}(\hat{b}_{i \to I}, \hat{b}_{I \to i})(\emptyset)$.

Figure 2.8 shows the resulting alternate SP factor graph for a portion of an original BP factor graph.

Furthermore, we wish to emphasize that we can cast the above formulation with our operator-generic BP framework where:

1. The variable set $w = \hat{b} = \{\hat{b}_{i \to I}, \hat{b}_{I \to i}\} \qquad \forall i, I$
2. The function $E(w) = G(\hat{b})(\emptyset) = \prod_I s_I(\hat{b}_{\partial I \to I})(\emptyset) \prod_i s_i(\hat{b}_{\partial i \to i})(\emptyset) \left(\prod_{i, I \in \partial i} s_{i \leftrightarrow I}(\hat{b}_{i \to I}, \hat{b}_{I \to i})(\emptyset) \right)^{-1}$
3. The marginalization operator $\bigoplus = \sum$
4. The expansion operator $\bigotimes = \prod$

Note here that with the above substitutions the original BP messages are now the variables for the SP framework.

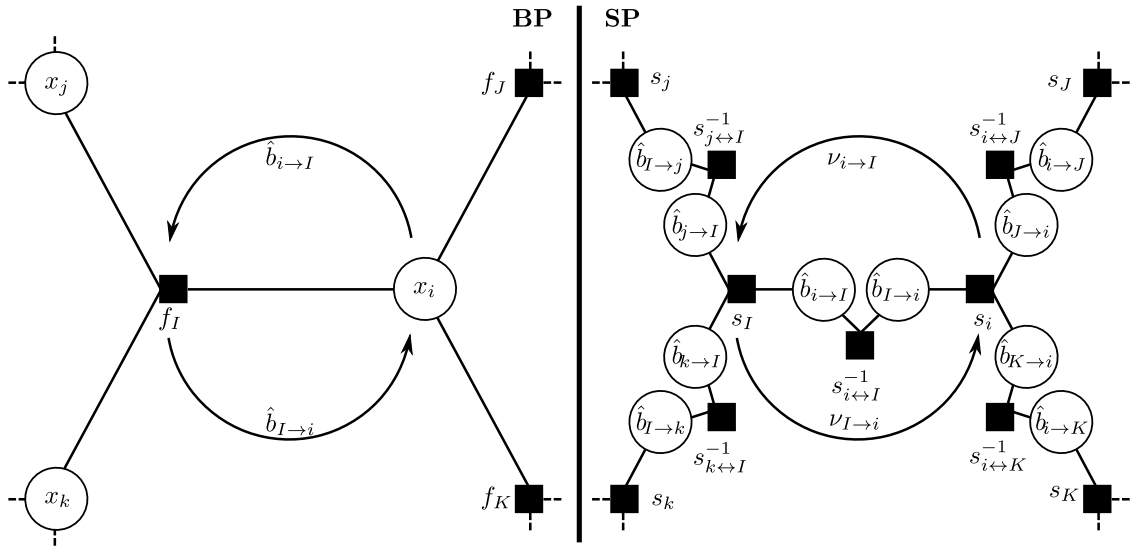


Figure 2.8: BP vs. SP factor graph comparison [68].

With the above correspondences established, appendix A shows the full set of SP messages. While the full set of SP messages involves three types of factors (as shown in appendix A), the appendix also shows that they can be reduced to two types of messages from variable-type factors to factor-type factors:

$$\nu_{i\to I}(\hat{b}_{i\to I}) \propto \sum_{\hat{b}_{\partial i\setminus I\to i}} \left(\mathbb{I}\left(\hat{b}_{i\to I} = s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})\right) s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(\emptyset) \prod_{J\in\partial i\setminus I} \nu_{J\to i}\ (\hat{b}_{J\to i}) \right) \tag{2.32}$$

and from factor-type factors to variable-type factors, given by:

$$\nu_{I\to i}(\hat{b}_{I\to i}) \propto \sum_{\hat{b}_{\partial I\setminus i\to I}} \left(\mathbb{I}\left(\hat{b}_{I\to i} = s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})\right) s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(\emptyset) \prod_{j\in\partial I\setminus i} \nu_{j\to I}\ (\hat{b}_{j\to I}) \right) \tag{2.33}$$

where:

$$s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(x_i) = \prod_{J\in\partial i\setminus I} \hat{b}_{J\to i}(x_i) \tag{2.34}$$

$$s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(x_i) = \sum_{x_{\partial I\setminus i}} f_I(x_I) \prod_{j\in\partial I\setminus i} \hat{b}_{j\to I}(x_j) \tag{2.35}$$

Lastly, the SP marginal at a variable-type factor is given by:

$$\nu_i(\hat{b}_i) \propto \sum_{\hat{b}_{\partial i\to i}} \left(\mathbb{I}\left(\hat{b}_i = s_i(\hat{b}_{\partial i\to i})\right) s_i(\hat{b}_{\partial i\to i})(\emptyset) \prod_{I\in\partial i} \nu_{I\to i}\ (\hat{b}_{I\to i}) \right) \tag{2.36}$$

and at a factor-type factor by:

$$\nu_I(\hat{b}_I) \propto \sum_{\hat{b}_{\partial I \to I}} \left(\mathbb{I}\left(\hat{b}_I = s_I(\hat{b}_{\partial I \to I})\right) s_I(\hat{b}_{\partial I \to I})(\emptyset) \prod_{i \in \partial I} \nu_{i \to I}\ (\hat{b}_{i \to I}) \right) \tag{2.37}$$

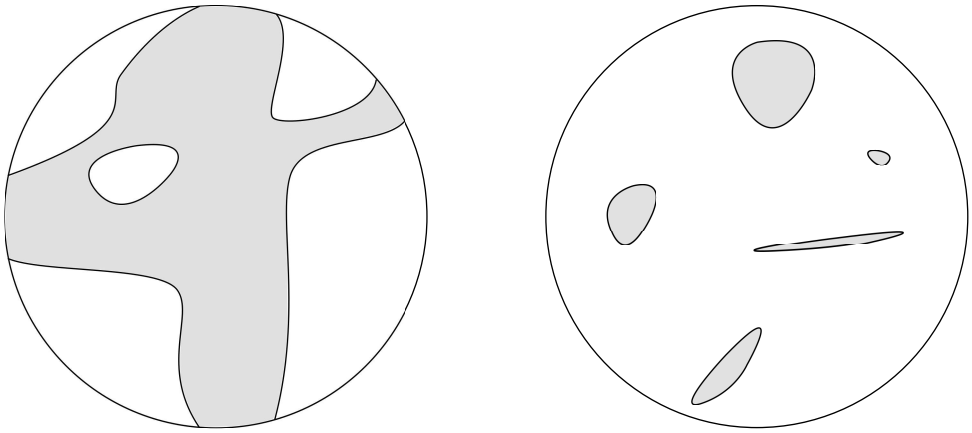
In the above equations, note that the indicator functions guarantee that the original BP updates are respected. Furthermore, each of these SP beliefs is a distribution over all possible values for a BP belief at either a variable or factor node of the original factor graph, respectively. Looking at these expressions, we see that the probability of each possible value is given by the number of BP fixed points in which it takes part along with the weight (*i.e.*, G) of each of these fixed points.

2.3.1 History of survey propagation

When SP was first introduced in [12] and [56], it was used to solve K-SAT problems where the variables are binary. In section 2.2.2 we looked at solving a SAT problem using LBP with logical OR and AND operators. When using these operators, the values of the messages and beliefs are then restricted to being either zero or one. More specifically, a message will have a 1 on one of its states if there exists a satisfiable solution for all other variables by setting its destination variable to that state and a 0 otherwise. It is with this limited BP message space that SP was originally designed: $b \in \{[1,0],[0,1],[1,1]\}$ for a message constraining a variable to zero, one, or leaving it unconstrained respectively.

For K-SAT problems SP had much success in finding solutions at high clause to variable ratios where a phase transition is known to exist. Specifically, it has been shown that when the ratio of clauses to the number of variables α is low, the set of satisfiable configurations for the variables is connected as shown in Figure 2.9a [55]. A connected set means that it is possible to move from one satisfiable configuration

of the variables to another by changing the states on a very small subset of the variables. However, it has been shown that there exists a critical threshold where this set becomes disjoint as shown in Figure 2.9b [55].



(a) Before clustering. (b) After clustering.

Figure 2.9: Phase transitions of solution space.

Running BP within this disjoint space would lead to only searching within one of the clusters of the satisfiable configurations. However, SP would return beliefs that capture, not only a satisfiable solution within a cluster as done with BP, but rather across all clusters. This is important as the ratio increases and the clusters fraction more and more becoming smaller as well. By tracking the BP messages and thus all fixed points, we can think of SP as essentially reconnecting the disjoint clusters together, even though they are actually disjoint and reside far apart in the original space.

For 3-SAT, the ratio at which the solution space fractions into clusters occurs at $\alpha \approx 3.9$ and BP begins to fail at finding solutions around this value [12]. In contrast, SP is able to find solutions without failing almost exactly up to $\alpha \approx 4.267$ at which point no more SAT solutions are believed to exist [12].

SP was extended in [77] and [83] to other CSPs with binary variables, and then

in [13],[11] and [74] to CSPs with non-binary variable alphabets such as the graph coloring problem. In all these problems, all the factors in the graph represent hard constraints. For that reason, the BP messages assume only a finite number of values and the SP messages that are essentially distributions over BP messages remain tractable [55]. This property is lost when the factors are no longer hard constraints. In chapter 4 we develop SP for applications beyond CSPs by addressing the new algorithmic issues which arise in these settings.

Chapter 3

Min-max problems and factor graphs

The min-max objective appears in various fields, particularly in building robust models under uncertain and adversarial settings. In the context of probabilistic graphical models, several different min-max objectives have been previously studied ([45] and [42]). In combinatorial optimization, min-max may refer to the relation between maximization and minimization in dual combinatorial objectives and their corresponding linear programs ([71]), or it may refer to min-max settings due to uncertainty in the problem specification ([5] and [1].

Our setting is closely related to a third class of min-max combinatorial problems that are known as bottleneck problems. Instances of these problems include the bottleneck Traveling Salesman Problem (TSP) [62], min-max clustering [33], the k-centers problem ([20] and [46]) and the bottleneck assignment problem [35].

The authors in [22] introduce a bottleneck framework with a duality theorem that relates the min-max objective in one problem instance to a max-min objective in the corresponding dual problem. An intuitive example is the duality between the min-

max cut separating nodes i and j – minimum of the maximum weight in a cut – and min-max path between i and j– minimum of maximum weight in a path [26]. The authors in [41] leverage the triangle inequality in metric spaces to find constant factor approximations to several NP-hard min-max problems under a unified framework.

In this chapter we show three ways by which a min-max problem can be solved on a factor graph:

- Using a min-max variant of BP with min and max operators themselves directly inside the messages

- As a satisfiability problem with a variant of sum-product BP as the satisfiability solver

- As a min-sum BP problem where the factor graph is transformed such that the min-sum objective produces the same optimal result as the min-max objective on the original factor graph

Although the third method is theoretically appealing, we show that it suffers from numerical problems. We then compare the first two methods on hard instances of random graphs and also show results on the bottleneck problems mentioned above.

Since the idea of running BP with min-max operators is also in itself novel, we show for the first method an efficient manner to pass min-max BP messages through high order factors. This enables us to use the first method to solve the problem of makespan minimization as presented in chapter 1 - a problem which we explain cannot be solved via the satisfiability method.

3.1 Min-max propagation

A question of interest is whether min and max operators satisfy the distributive law of mathematics as presented in chapter 1. As shown in Table 3.1, it turns out that indeed,

$$\min(\max(a,b),\max(a,c)) = \max(a,\min(b,c)). \tag{3.1}$$

Case	LHS	RHS
$a > b > c$	a	a
$a > c > b$	a	a
$b > a > c$	a	a
$b > c > a$	c	c
$c > a > b$	a	a
$c > b > a$	b	b

Table 3.1: Min-max distributive law (left hand side vs. right hand side of Equation 3.1).

From this observation we can derive a min-max version of BP.We name this new algorithm Min-Max Propagation (MMP). Ignoring for now the choice of 1) a variable set and 2) a function, this operator substitution can be viewed as an instance of our operator-generic BP framework from chapter 2 where:

3. The marginalization operator $\bigoplus = \min$

4. The expansion operator $\bigotimes = \max$

Due to the novelty of using and min and max operators for messages in a factor graph context, we make these substitutions explicit to obtain the following MMP update rules and beliefs.

3.1.1 Variable-to-factor messages

The message sent from variable x_i to factor f_I is computed as follows:

$$\nu_{i \to I}(x_i) = \max_{J \in \partial i \setminus I} \nu_{J \to i}(x_i) \tag{3.2}$$

3.1.2 Factor-to-variable messages

The message sent from factor f_I to variable x_i is computed using:

$$\nu_{I \to i}(x_i) = \min_{x_{\partial I \setminus i}} \left(f_I(x_I), \max_{j \in \partial I \setminus i} \nu_{j \to I}(x_j) \right) \tag{3.3}$$

3.1.3 Beliefs

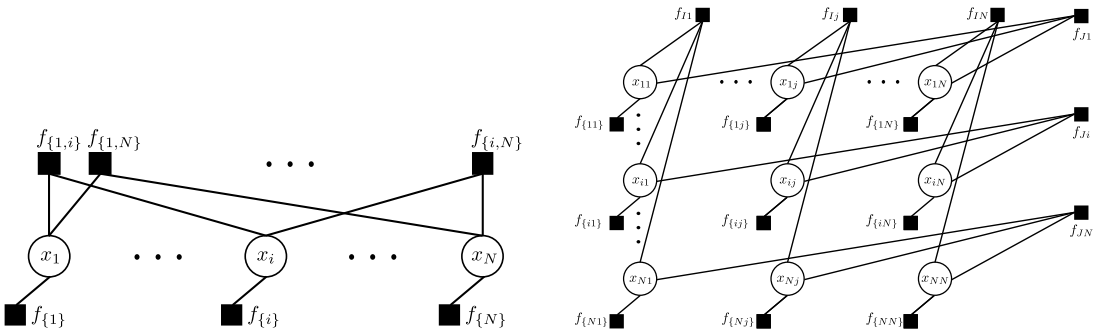
The beliefs at variable and factor nodes are computed as follows:

$$\nu_i(x_i) = \max_{I \in \partial i} \nu_{I \to i}(x_i) \tag{3.4}$$

$$\nu_I(x_I) = f_I(x_I) \max_{i \in \partial I} \nu_{i \to I}(x_i) \tag{3.5}$$

3.1.4 Initialization

When using min-max operators, uniform message initialization corresponds to setting all values in the message to $-\infty$ since it is the identity of the max operator (*i.e.*, $\max(I, a) = a$ when $I = -\infty$), and normalization corresponds to subtracting a constant from all values in the message.



(a) (b) K-packing (binary)

Figure 3.1: Factor graphs for the bottleneck assignment problem.

3.2 Min-Max as a satisfiability problem

The common theme in a majority of heuristics for min-max or bottleneck problems is the relation of the min-max objective to a CSP ([41] and [61]). We establish a similar relation within the context of factor graphs, by reducing the min-max inference problem on the original factor graph to that of sampling by solving a CSP on the reduced factor graph.

3.2.1 Factor graph conversion and bisection search

Given $\mathcal{F}$ to be the set of factors and x^* to be the configuration of the variables to obtain the optimal min-max solution, let $\mathcal{Y} = \bigcup_I \mathcal{Y}_I$ denote the union over the range of all factors. The min-max value belongs to this set $\max_{I \in \mathcal{F}} f_I(x_I^*) \in \mathcal{Y}$. In fact for any assignment x, $\max_{I \in \mathcal{F}} f_I(x_I) \in \mathcal{Y}$.

Example *Bottleneck Assignment Problem:* given a matrix $D \in \Re^{N \times N}$, select a subset of entries of size N to minimize the maximum entry, such that in each row and each column exactly a single number is selected. As an application assume the entry $D_{i,j}$ is the time required by worker i to finish task j. The min-max assignment minimizes the maximum time required by any worker to finish his assignment. This

problem is also known as bottleneck bipartite matching and belongs to class $\mathcal{P}$ [29].

Here we show two factor-graph representations of this problem.

Categorical variable representation: Consider a factor-graph with $x = \{x_1, \ldots, x_N\}$, where each variable $x_i \in \{1, \ldots, N\}$ indicates the column of the selected entry in row i of D. For example, $x_1 = 5$ indicates that the fifth column of the first row is selected (see Figure 3.1a). Define the following factors: (a) local factors $f_{\{i\}}(x_i) = D_{i,x_i}$ and (b) pairwise factors $f_{\{i,j\}}(x_{\{i,j\}}) = \infty\mathbb{I}(x_i = x_j) - \infty\mathbb{I}(x_i \neq x_j)$ that enforce the constraint $x_i \neq x_j$. Note that if $x_i = x_j$, $f_{\{i,j\}}(x_{\{i,j\}}) = \infty$, making this choice unsuitable in the min-max solution. On the other hand, if $x_i \neq x_j$, $f_{\{i,j\}}(x_{\{i,j\}}) = -\infty$ and this factor has no impact on the min-max value.

Binary variable representation: Consider a factor-graph where $x = [x_{1-1}, \ldots, x_{1-N}, x_{2-1} \ldots, x_{2-N}, \ldots, x_{N-N}] \in \{0,1\}^{N\times N}$, indicates whether each entry is selected $x_{i-j} = 1$ or not $x_{i-j} = 0$ (see Figure 3.1b). Here the factor $f_I(x_I) = -\infty\mathbb{I}(\sum_{i\in\partial I} x_i = 1) + \infty\mathbb{I}(\sum_{i\in\partial I} x_i \neq 1)$ ensures that only one variable in each row and column is selected and local factors $f_{i-j}(x_{i-j}) = x_{i-j}D_{i,j} - \infty(1 - x_{i-j})$ have any effect only if $x_{i-j} = 1$.

The min-max assignment in both of these factor-graphs gives a solution to the bottleneck assignment problem.

For any $y \in \mathcal{Y}$ in the range of factor values, we *reduce* the original min-max problem to a CSP using the following reduction. For any $y \in \mathcal{Y}$, the μ_y**-reduction** of the min-max problem is given by

$$\mu_y(x) \quad \triangleq \quad \frac{1}{Z_y}\prod_I \mathbb{I}(f_I(x_I) \leq y) \tag{3.6}$$

where $Z_y \triangleq \sum_{\mathcal{X}}\prod_I \mathbb{I}(f_I(x_I) \leq y)$ is the normalizing constant and $\mathbb{I}(.)$ is the indicator

function. [1] This distribution defines a CSP over $\mathcal{X}$, where $\mu_y(x) > 0$ *iff* x is a satisfying assignment. Moreover, Z_y gives the number of satisfying assignments.

We will use $f_I^y(x_I) \triangleq \mathbb{I}(f_I(x_I) \leq y)$ to refer to reduced factors. The following theorem is the basis of our approach to solving min-max problems.

Theorem 3.2.1 [2] *Let x^* denote the min-max solution and y^* be its corresponding value – $y^* = \max_I \ f_I(x_I^*)$. Then $\mu_y(x)$ is satisfiable for all $y \geq y^*$ (in particular $\mu_y(x^*) > 0$) and unsatisfiable for all $y < y^*$.*

This theorem enables us to find a min-max assignment by solving a sequence of CSPs. Let $y^{(1)} \leq \ldots \leq y^{(N)}$ be an ordering of $y \in \mathcal{Y}$. Starting from $y = y^{(\lceil N/2 \rceil)}$, if μ_y is satisfiable then $y^* \leq y$. On the other hand, if μ_y is not satisfiable, $y^* > y$. Using a bisection search, we need to solve $\log(|\mathcal{Y}|)$ CSPs to find the min-max solution. Moreover, at any time step during the search we have both upper and lower bounds on the optimal solution. That is $\underline{y} < y^* \leq \overline{y}$, where $\mu_{\underline{y}}$ is the latest unsatisfiable and $\mu_{\overline{y}}$ is the latest satisfiable reduction.

3.2.2 Satisfiability solvers

So far, we have assumed that we are using an exact CSP solver. Previously, in solving CSP reductions, we assumed an ideal CSP solver. Finding an assignment x such that $\mu_y(x) > 0$, or otherwise showing that no such assignment exists, is in general NP-hard [18]. However, message passing methods have been successfully used to provide state of the art results in solving difficult CSPs.

We use Perturbed Belief Propagation (PBP) [69] for this purpose. By using an incomplete solver [44], we lose the upper-bound $\overline{y}$ on the optimal min-max solution,

[1] To always have a well-defined probability, we define $\frac{0}{0} \triangleq 0$.
[2] All proofs appear in appendix B.

as BP or PBP may not find a solution even if the instance is satisfiable. However, the following proposition states that, as we increase y from the min-max value y^*, the number of satisfying assignments to μ_y-reduction increases, making it potentially easier to solve.

Proposition 3.2.2

$$y_1 < y_2 \Rightarrow Z_{y_1} \leq Z_{y_2} \qquad \forall y_1, y_2 \in \Re$$

This means that the sub-optimality of our solution is related to our ability to solve CSP-reductions – that is, as the gap $y - y^*$ increases, the μ_y-reduction potentially becomes easier to solve.

PBP is a message passing method that interpolates between sum-product BP and Gibbs sampling. At each iteration, PBP sends a message from variables to factors ($\nu_{i \to I}$) and vice versa ($\nu_{I \to i}$). The factor to variable message is the same as BP. However, the variable to factor message for PBP is slightly different from BP; it is a linear combination of BP message update and an indicator function, defined based on a sample from the current estimate of marginal $\widehat{\mu}(x_i)$:

$$\nu_{i \to I}(x_i) \;\propto\; (1-\gamma) \prod_{J \in \partial i \setminus I} \nu_{J \to i}(x_i) \;+\; \gamma\, \mathbb{I}(\widehat{x}_i = x_i) \tag{3.7}$$

$$\text{where } \widehat{x}_i \sim \widehat{\mu}(x_i) \;\propto\; \prod_{J \in \partial i} \nu_{J \to i}(x_i) \tag{3.8}$$

where for $\gamma = 0$ we have BP updates and for $\gamma = 1$, we have Gibbs sampling. PBP starts at $\gamma = 0$ and linearly increases γ at each iteration, ending at $\gamma = 1$ at its final iteration. At any iteration PBP may encounter a contradiction where the product of incoming messages to node i is zero for all $x_i \in \mathcal{X}_i$. This could mean that either

the problem is unsatisfiable or PBP is not able to find a solution. However, if it reaches the final iteration, PBP produces a sample from $\mu_y(x)$, which is a solution to the corresponding CSP. The number of iterations T is the only parameter of PBP and increasing T, increases the chance of finding a solution (Only downside is time complexity; nb., no chance of a false positive.)

Computational complexity

PBP's time complexity per iteration is identical to that of BP:

$$\mathcal{O}(\sum_I (|\partial I| \; |\mathcal{X}_I|) + \sum_i (|\partial i| \; |\mathcal{X}_i|)) \tag{3.9}$$

where the first summation accounts for all factor-to-variable messages [3] and the second one accounts for all variable-to-factor messages.

To perform binary search over $\mathcal{Y}$ we need to sort $\mathcal{Y}$, which requires $\mathcal{O}(|\mathcal{Y}| \log(|Y|))$. However, since $|\mathcal{Y}_i| \leq |\mathcal{X}_i|$ and $|\mathcal{Y}| \leq \sum_I |\mathcal{Y}_i|$, the cost of sorting is already contained in the first term of Equation 3.9, and may be ignored in asymptotic complexity.

The only remaining factor is that of binary search itself, which is $\mathcal{O}(\log(|\mathcal{Y}|)) = \mathcal{O}(\log(\sum_I (|\mathcal{X}_I|)))$ – *i.e.*, at most logarithmic in the cost of PBP's iteration (*i.e.*, first term in Equation 3.9). Also note that the factors added as constraints only take two values of $\pm\infty$, and have no effect in the cost of binary search.

As this analysis suggests, the dominant cost is that of sending factor-to-variable messages, where a factor may depend on a large number of variables. The next section shows that many factors of interest are sparse, which allows efficient calculation of messages.

[3] The $|\partial I|$ indicates the number of messages that are sent from each factor. However if the messages are calculated synchronously this factor disappears. Although more expensive, in practice, asynchronous message updates performs better.

High order factors

The factor-graph formulation of many interesting min-max problems involves sparse high-order factors. In all such factors, we are able to significantly reduce the $\mathcal{O}(|\mathcal{X}_I|)$ time complexity of calculating factor-to-variable messages. Efficient message passing over such factors is studied by several works in the context of sum-product and max-product inference ([66], [37]. [72], and [73]). The simplest form of sparse factor in our formulation is the so-called Potts factor, $f^y_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i = x_j)\phi(x_i)$. This factor assumes the same domain for all the variables ($\mathcal{X}_i = \mathcal{X}_j \; \forall i, j$) and its tabular form is non-zero only across the diagonal. It is easy to see that this allows the computation of the factor-to-variable message to be performed in $\mathcal{O}(|\mathcal{X}_i|)$ rather than $\mathcal{O}(|\mathcal{X}_i| \; |\mathcal{X}_j|)$. Another factor of similar form is the inverse Potts factor, $f^y_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i \neq x_j)$, which ensures $x_i \neq x_j$. In fact, any pairwise factor that is a constant plus a band-limited matrix allows $\mathcal{O}(|\mathcal{X}_i|)$ inference (*e.g.*, see section 3.5.4).

In section 3.5, we use cardinality factors, where $\mathcal{X}_i = \{0, 1\}$ and the factor is defined based on the number of non-zero values – *i.e.*, $f^y_{\mathcal{K}}(x_{\mathcal{K}}) = f^y_{\mathcal{K}}(\sum_{i\in\mathcal{K}} x_i)$. The μ_y-reduction of the factors we use in the binary representation of the bottleneck assignment problem is in this category. The work of [27] proposes a simple $\mathcal{O}(|\partial I| \; K)$ method for $f^y_{\mathcal{K}}(x_{\mathcal{K}}) = \mathbb{I}(\sum_{i\in\mathcal{K}} x_i = K)$. We refer to this factor as K-of-N factor and use similar algorithms for at-least-K-of-N $f^y_{\mathcal{K}}(x_{\mathcal{K}}) = \mathbb{I}(\sum_{i\in\mathcal{K}} x_i \leq K)$ and at most-K-of-N $f^y_{\mathcal{K}}(x_{\mathcal{K}}) = \mathbb{I}(\sum_{i\in\mathcal{K}} x_i \geq K)$ factors (see Appendix B). An alternative for more general forms of high order factors is the clique potential of [66]. For large K, more efficient methods evaluate the sum of pairs of variables using auxiliary variables forming a binary tree and use Fast Fourier Transform (FFT) to reduce the complexity of K-of-N factors to $\mathcal{O}(N \log(N)^2)$ (see [73] and references in there).

3.3 Reduction to min-sum

The authors of [43] introduce a simple method to transform instances of the bottleneck TSP to TSP. Here we show how this result extends to min-max problems over factor graphs.

Lemma 3.3.1 *Any two sets of factors,* $\{f_I\}_{I\in\mathcal{F}}$ *and* $\{f'_I\}_{I\in\mathcal{F}}$*, have identical min-max solution*

$$\arg_x \min \max_I f_I(x_I) = \arg_x \min \max_I f'_I(x_I)$$

if $\forall I, J \in \mathcal{F}, x_I \in \mathcal{X}_I, x_J \in \mathcal{X}_J$

$$f_I(x_I) < f_J(x_J) \Leftrightarrow f'_I(x_I) < f'_J(x_J)$$

This lemma states that what matters in the min-max solution is the *relative ordering* in the factor values.

Let $y^{(1)} \leq \ldots \leq y^{(N)}$ be an ordering of elements in $\mathcal{Y}$, and let $r(f_I(x_I))$ denote the rank of $y_I = f_I(x_I)$ in this ordering. Defining the min-sum reduction of $\{f_I\}_{I\in\mathcal{F}}$ as

$$f'_I(x_I) = 2^{r(f_I(x_I))} \quad \forall I \in \mathcal{F}$$

we have,

Theorem 3.3.2

$$\arg_x \min \sum_I f'_I(x_I) \quad = \quad \arg_x \min \max_I f_I(x_I)$$

where $\{f'_I\}_I$ *is the min-sum reduction of* $\{f_I\}_I$*.*

$$\begin{bmatrix} D & D-1 & D-2 & \dots & 1 \\ D-1 & D-1 & D-2 & \dots & 1 \\ D-2 & D-2 & D-2 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Figure 3.2: A pairwise factor before random shuffling.

Although this allows us to use min-sum message passing to solve min-max problems, the values in the range of factors grow exponentially fast, resulting in numerical problems.

3.4 Algorithm comparison

Like all BP propagation algorithms, MMP is exact when the factor graph is a tree. However, our first point of interest is how MMP performs on loopy graphs. For this, we compare its performance against the sum-product (or CSP) reduction.

3.4.1 Experimental setup

Combined with the observation made in section 3.3 that only the *ordering* of the factor values affects the min-max assignment, our setup is based on the following:

Observation Only the factor(s) which output(s) the max value, *i.e.*, **max factor**(s), matter. For all other factors, the variables involved can be set in any way as long as the value of the factor(s) remains smaller or equal to that of the max factor.

This means that variables that do not appear in the max factor(s), which we call **free variables**, could potentially assume an exponentially large number of values without affecting the min-max value. *This phenomenon is unique to min-max inference and does not appear in min-sum and sum-product counterparts.*

We rely on this observation in designing benchmark random min-max inference problems: i) we use integers as the range of factor values; ii) by selecting all factor values in the same range, we can use the number of factors as a *control parameter* for the difficulty of the inference problem.

For N variables $x_1, \ldots, x_N$, where each $x_i \in \{1, \ldots, D\}$, we draw Erdos-Renyi graphs with edge probability $p \in (0, 1]$ and treat each edge as a pairwise factor. Consider the factor $f_a(x_i, x_j) = \min(\pi(x_i), \pi'(x_j))$, where π, π' are permutations of $\{1, \ldots, D\}$. With $D = 2$, this definition of factor f_a reduces to 2-SAT factor. This setup for random min-max instances therefore generalizes different K-SAT settings, where the min-max solution of $min_x \max_a f_a(x_{\partial a}) = 1$ for $D = 2$, corresponds to a satisfying assignment. *The same argument with $K > 2$ establishes the "NP-hardness" of min-max inference in factor-graphs.*

We test our setup on graphs with $N \in \{10, 100\}$ variables and cardinality $D \in \{4, 6, 8\}$. For each choice of D and N, we run min-max propagation and sum-product reduction for various connectivity in the Erdos-Renyi graph. Both methods use random sequential update. For $N = 10$ we also report the exact min-max solutions.

MMP is run for a maximum $T = 1000$ iterations or until convergence, whichever comes first. The PBP used in the sum-product reduction requires a fixed T; we report the results for T 1) determined by worse case min-max convergence iterations and 2) $T = 1000$ iterations. Each setting is repeated 10 times for a random graph of a fixed connectivity value $p \in (0, 1]$.

Decimation. To obtain a final min-max assignment we need to fix the free variables. For this we use a decimation scheme similar to what is used with min-sum inference or in finding a satisfying CSP assignment in sum-product. We consider three different decimation procedures:

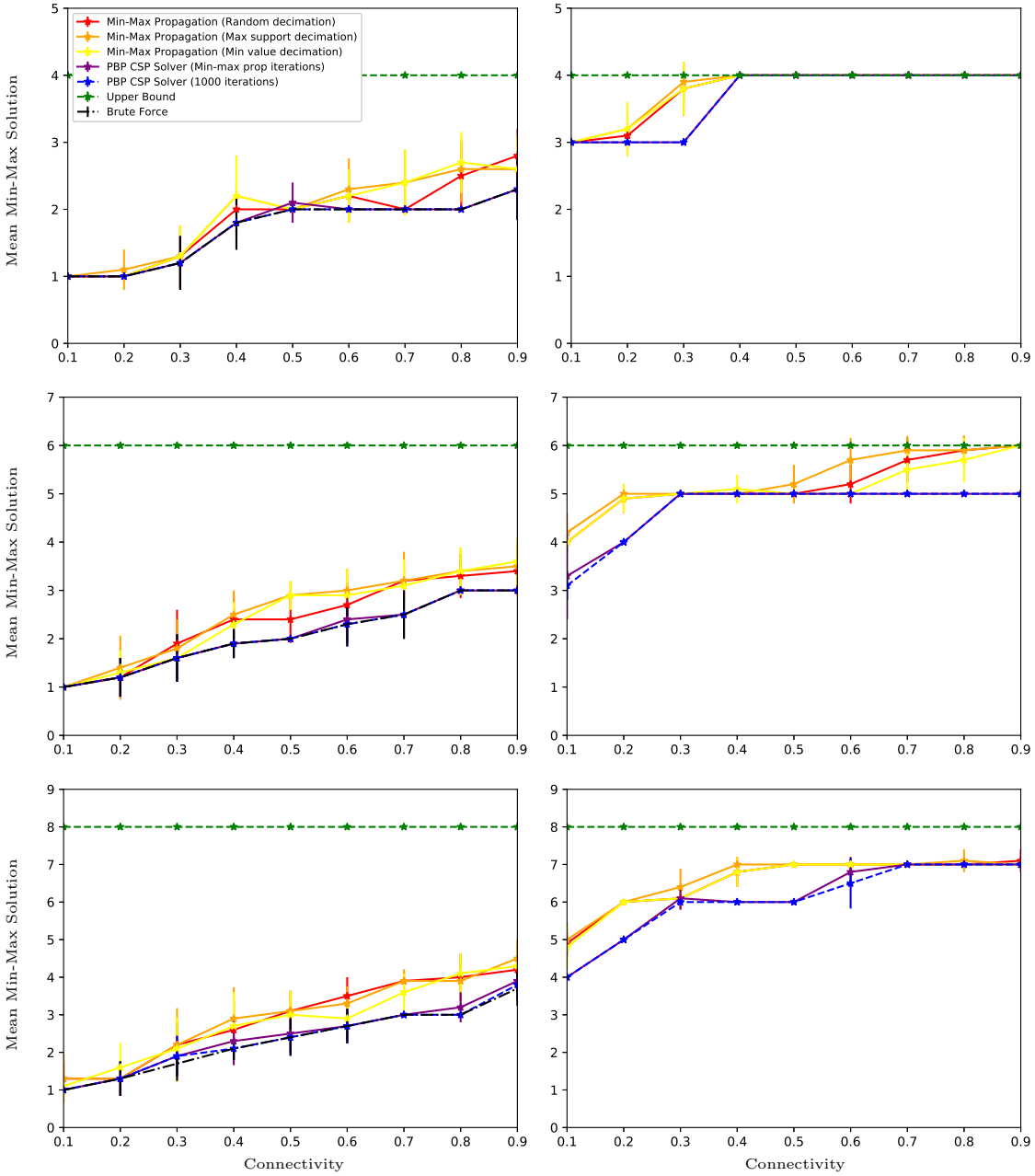


Figure 3.3: Min-max performance of different methods on Erdos-Renyi random graphs. Left: N=10, Right: N=100, Top: K=4, Middle: K=6, Bottom: K=8

Random: Randomly choose a variable, set it to the state with minimum min-max marginal value.

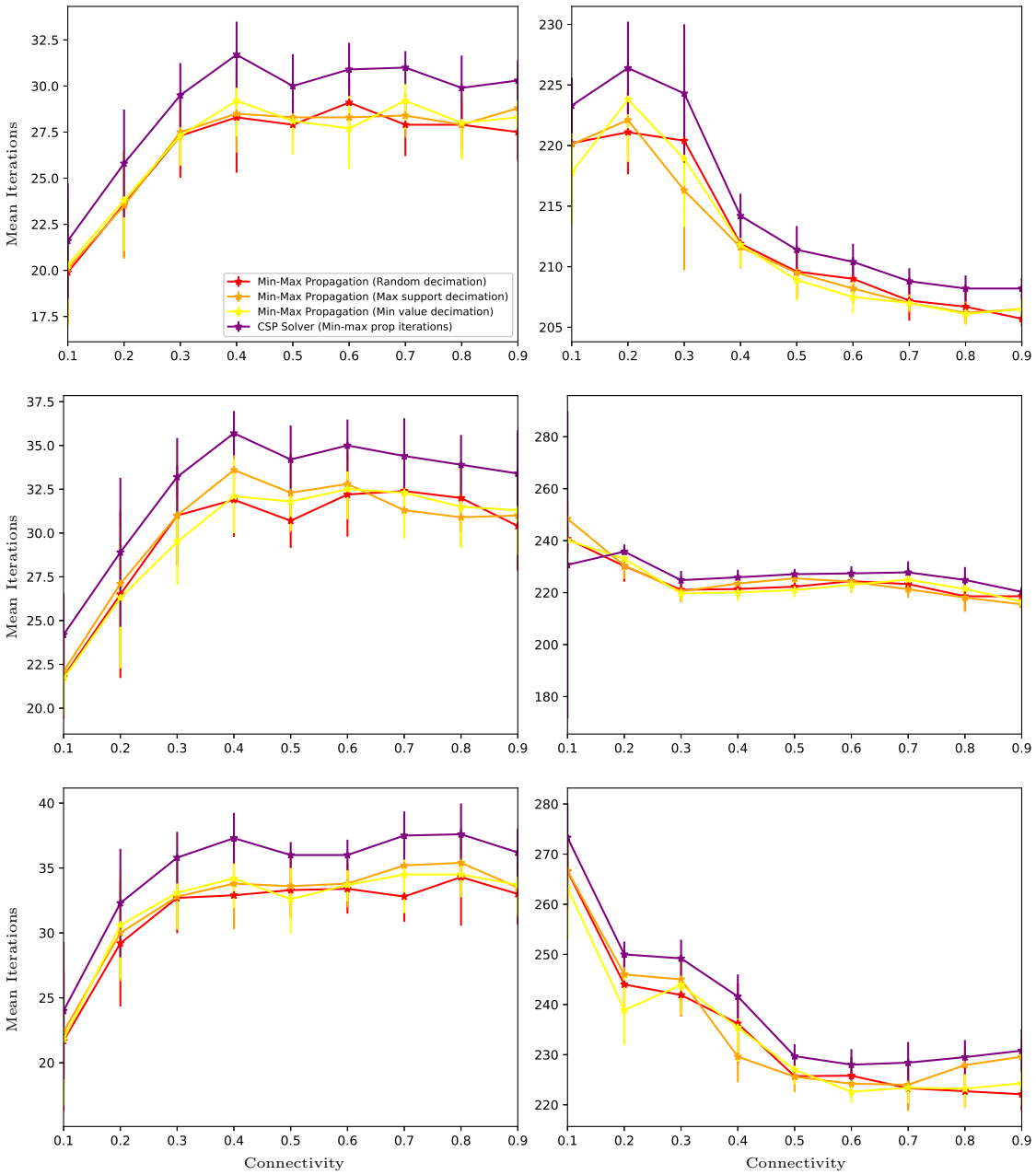


Figure 3.4: Number of iterations for different methods on Erdos-Renyi graphs. Left: N=10, Right: N=100, Top: K=4, Middle: K=6, Bottom: K=8

Min-value: Fix the variable with the minimum min-max marginal value.

Max-support: Choose the variable for which its min value occurs most frequently.

min-max problem	μ_y-reduction	msg-passing cost	α
min-max clustering	clique cover problem	$\mathcal{O}(N^2K\log(N))$	2^* ([33])
K-packing	max-clique problem	$\mathcal{O}(N^2K\log(N))$	N/A
(weighted) K-center problem	dominating set problem	$\mathcal{O}(N^3\log(N))$ or $\mathcal{O}(NR^2\log(N))$	$\min(3, 1+\frac{\max_i w_i}{\min_i w_i})$ ([20])
asymmetric K-center problem	set cover problem	$\mathcal{O}(N^3\log(N))$ or $\mathcal{O}(NR^2\log(N))$	$\log(N)^*$ ([61];[17])
bottleneck TSP	Hamiltonian cycle problem	$\mathcal{O}(N^3\log(N))$	2^* ([62])
bottleneck Asymmetric TSP	directed Hamiltonian cycle	$\mathcal{O}(N^3\log(N))$	$\log(n)/\log(\log(n))$ ([4])

Table 3.2: Min-max combinatorial problems and the corresponding CSP-reductions. The last column reports the best α-approximations when triangle inequality holds. $*$ indicates best possible approximation.

3.4.2 Results

Figure 3.3 compares the performance of sum-product reduction that relies on PBP against that of MMP. For its part Figure 3.4 shows the number of iterations actually taken by all methods. Each row uses a different variable cardinality D while each column varies in the number of variables N. In the left column (where the problems are small enough), we also report the exact min-max value obtained via brute force. While each row changes the range D of values in the factors, we observe a similar trend in the performance of different methods.

As expected, by increasing the number of factors (connectivity), the min-max value increases. Overall, the sum-product reduction (although asymptotically more expensive) produces slightly better results. Also, different decimation schemes do not significantly affect the results.

Since the sum-product reduction yields better results, we choose it as our de-facto min-max solver for the applications considered in section 3.5. Only in section 3.6 do we use MMP to tackle the makespan minimization example because, as we show, the sum-product reduction of that application cannot be solved in polynomial time.

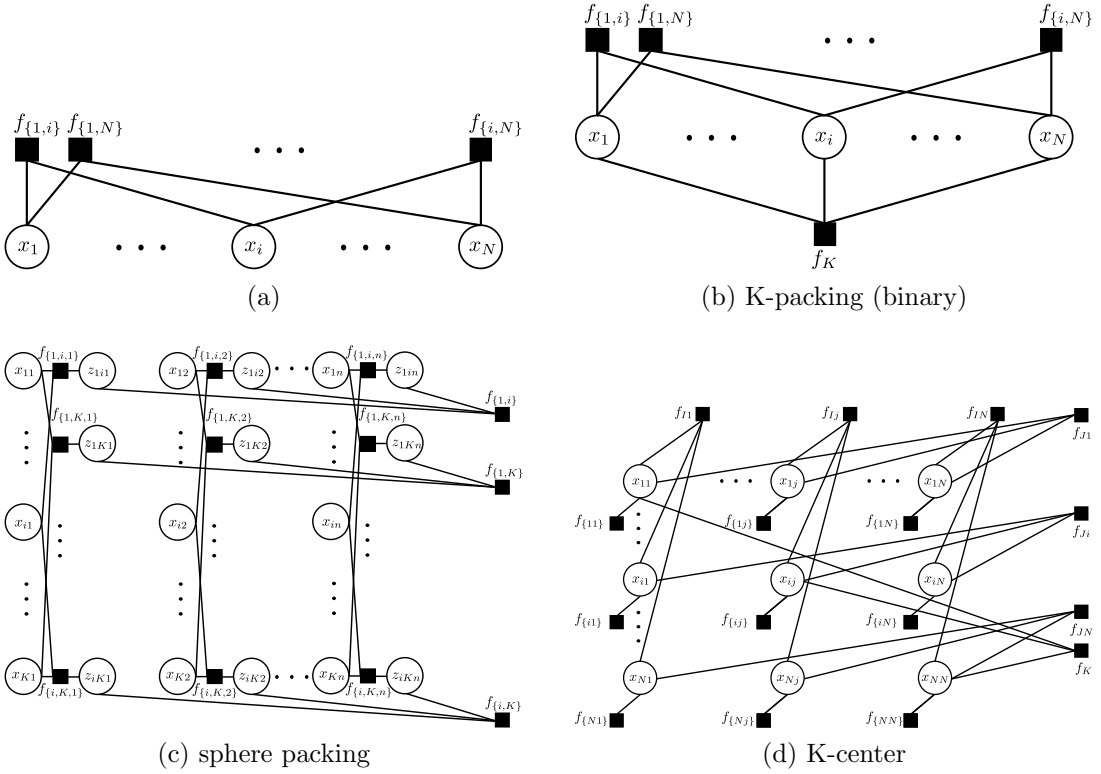


Figure 3.5: Factor graphs for different applications. Factor graph (a) is common between min-max clustering, Bottleneck TSP and K-packing (categorical). However, the definition of factors is different in each case.

3.5 Applications using the CSP solver

Here, we introduce the factor graphs for several NP-hard min-max problems. The CSP-reduction for each is an important NP-hard problem. Table 3.2 shows the relationship between the min-max and the corresponding CSP and the factor α in the constant factor approximation available for each case - *e.g.*, $\alpha = 2$ means the results reported by some algorithm is guaranteed to be within factor 2 of the optimal min-max value $\widehat{y^*}$ when the distances satisfy triangle inequality. Comparing against methods that provide a guarantee gives us a reference point to demonstrate near-optimality of our approach. Table 3.2 shows the complexity of the procedure (with asynchronous message updates) to find the solution. See Appendix B for details.

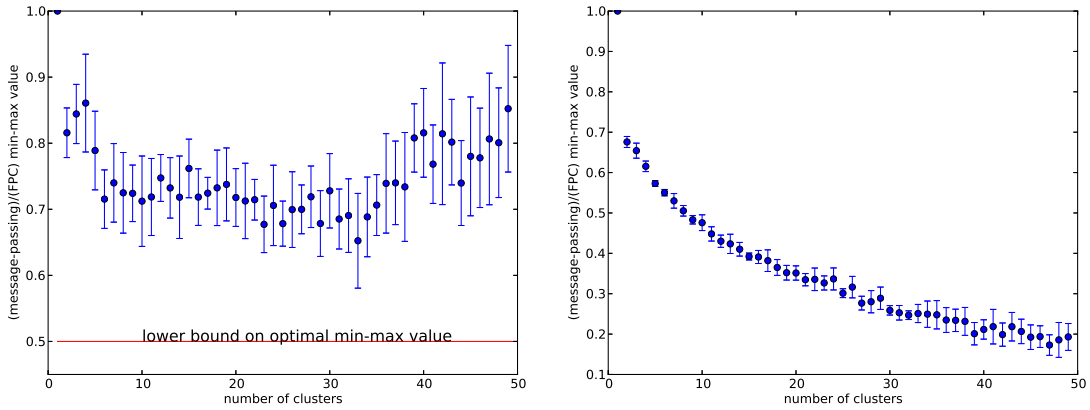


Figure 3.6: Min-max clustering of 100 points with varying number of clusters (x-axis). Each point is an average over 10 random instances. The y-axis is the ratio of the min-max value obtained by message passing ($T = 50$ iterations for PBP) over the min-max value of FPC. **(left)** Clustering of random points in 2D Euclidean space. The horizontal line is the lower bound on the optimal result based on the worst case guarantee of FPC. **(right)** Using symmetric random distance matrix where $D_{i,j} = D_{j,i} \sim U(0, 1)$.

3.5.1 Min-max clustering

Given a symmetric matrix of pairwise distances $D \in \Re^{N \times N}$ between N data-points and a number of clusters K, min-max clustering seeks a partitioning of data-points that minimizes the maximum distance between all the pairs in the same partition.

Let $x = \{x_1, \ldots, x_N\}$ with $x_i \in \mathcal{X}_i = \{1, \ldots, K\}$ be the set of variables, when $x_i = k$ means, point i belongs to cluster k. The Potts factor $f_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i = x_j)D_{i,j} - \infty\mathbb{I}(x_i \neq x_j)$ between any two points is equal to $D_{i,j}$ if points i and j belong to the same cluster and $-\infty$ otherwise (see Figure 3.5a). When applied to this factor graph, the min-max solution defines the clustering of points with minimum of maximum inter-cluster distances.

Now we investigate properties of μ_y-reduction for this factor graph.

y-neighborhood graph for distance matrix $D \in \Re^{N \times N}$ is defined as graph $\mathcal{G}(D, y) = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ and $\mathcal{E} = \{(i, j) |\ D_{i,j} \leq y\}$. Note than this definition is also valid for an asymmetric adjacency matrix D. In such cases the y-neighborhood

graph is a directed-graph.

The *K-clique-cover* $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a partitioning of $\mathcal{V}$ to at most K partitions such that $\forall i, j, k \;\; i, j \in \mathcal{C}_k \Rightarrow (i, j) \in \mathcal{E}$.

Proposition 3.5.1 *The μ_y-reduction of the min-max clustering factor graph defines a uniform distribution over the K-clique-covers of $\mathcal{G}(D, y)$.*

Figure 3.6 compares the performance of min-max clustering using message passing to that of Furthest Point Clustering (FPC) of [33] that is 2-optimal when the triangle inequality holds. Note that message passing solutions are superior even when using Euclidean distance.

3.5.2 K-packing

Given a symmetric distance matrix $D \in \Re^{N \times N}$ between N data-points and a number of code-words K, the K-packing problem is to choose a subset of K points such that the minimum distance $D_{i,j}$ between any two code-words is maximized. Here we introduce two different factor graph formulations for this problem.

First formulation: binary variables

Let binary variables $x = \{x_1, \ldots, x_N\} \in \{0, 1\}^N$, indicate a subset of variables of size K that are selected as code-words (see Figure 3.5b). Use the factor $f_{\mathcal{K}}(x) = \infty\mathbb{I}(\sum_i x_i \neq K) - \infty\mathbb{I}(\sum_i x_i = K)$ (here $\mathcal{K} = \{1, \ldots, N\}$) to ensure this constraint. The μ_y-reduction of this factor for any $-\infty < y < +\infty$ is a K-of-N factor as defined in Section 3.2.2. Furthermore, for any two variables x_i and x_j, define the factor $f_{\{x_i, x_j\}}(x_{i,j}) = -D_{i,j} x_i x_j - \infty(1 - x_i x_j)$. This factor is effective only if both points are selected as code-words. We use $-D_{i,j}$ to convert the initial max-min objective to min-max.

Second formulation: categorical variables

Define the K-packing factor graph as follows: let $x = \{x_1, \ldots, x_K\}$ be the set of variables where $x_i \in \mathcal{X}_i = \{1, \ldots, N\}$ (see Figure 3.5a). For every two distinct points $1 \leq i < j \leq K$, define the factor $f_{\{i,j\}}(x_i, x_j) = -D_{x_i,x_j}\mathbb{I}(x_i \neq x_j) + \infty\mathbb{I}(x_i = x_j)$. Here each variable represents a code-word and the last term of each factor ensures that code-words are distinct.

Proposition 3.5.2 *The μ_y-reduction of the K-packing factor graph for the distance matrix $D \in \Re^{N\times N}$ defines a uniform distribution over the cliques of $\mathcal{G}(-D, -y)$ that are larger than K.*

The μ_y-reduction of our second formulation is similar to the factor graph used by [67] to find non-linear binary codes. The authors consider the Hamming distance between all binary vectors of length n (*i.e.*, $N = 2^n$) to obtain binary codes with known minimum distance y. As we saw, this method is $\mathcal{O}(N^2K^2) = \mathcal{O}(2^{2n}K^2)$ - *i.e.*, grows exponentially in the number of bits n. In the following section, we introduce a factor graph formulation specific to categorical variables with Hamming distance that have message passing complexity $\mathcal{O}(n^2K^2y)$ –*i.e.*, not exponential in n. Using this formulation, we find optimal binary codes where both n and y are large.

Sphere packing with Hamming distance

Our factor graph defines a distribution over the K binary vectors of length n such that the distance between every pair of binary vectors is at least y. [4] Finding so called "nonlinear binary codes" is a fundamental problem in information theory and a variety of methods have been used to find better codes where either the number

[4] For convenience we restrict this construction to the case of binary vectors. Similar procedure may be used to find maximally distanced ternary and q-ary vectors, for arbitrary q.

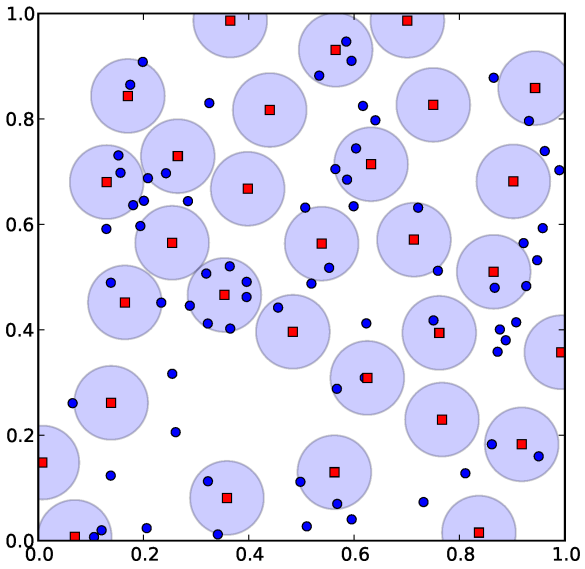


Figure 3.7: Using message passing to choose $K = 30$ out of $N = 100$ random points in the Euclidean plane to maximize the minimum pairwise distance (with $T = 500$ iterations for PBP). Touching circles show the minimum distance.

of keywords K or their minimum distance y is maximized (*e.g.*, see [52], and the references therein).

Let $x = \{x_{1-1}, \ldots, x_{1-n}, x_{2-1}, \ldots, x_{2-n}, \ldots, x_{K-n}\}$ be the set of our binary variables, where $x_i = \{x_{i-1}, \ldots, x_{i-n}\}$ represents the i^{th} binary vector or code-word. Additionally, for each $1 \leq i < j \leq K$, define an auxiliary binary vector $z_{i,j}$ of length n: $z_{i,j} = \{z_{i,j,1}, \ldots, z_{i,j,n}\}$ (see Figure 3.5c).

For each distinct pair of binary vectors x_i and x_j, and a particular bit $1 \leq k \leq n$, the auxiliary variable $z_{i,j,k} = 1$ *iff* $x_{i-k} \neq x_{j-k}$. Then we define an at-least-y-of-n factor over $z_{i,j}$ for every pair of code-words, to ensure that they differ in at least y bits.

In more details, define the following factors on x and z:

(a) *z-factors:* For every $1 \leq i < j \leq K$ and $1 \leq k \leq n$, define a factor to ensure that

2	1	2	1	0	1	1	2	2	1	2	1	2	1	0	2
1	1	1	2	1	0	0	2	2	1	1	1	0	2	1	0
0	0	1	2	0	1	2	0	2	1	2	2	1	0	2	0
0	0	0	2	1	0	1	2	1	0	1	0	2	1	0	1
2	2	0	2	2	2	1	1	2	0	2	2	1	2	1	2
0	2	0	0	0	2	0	2	0	0	2	1	0	0	0	0
1	1	2	0	0	1	2	2	0	0	1	0	1	2	2	1
2	0	2	0	2	1	0	1	1	2	0	1	1	1	2	0
0	2	1	1	1	1	1	1	0	1	0	0	0	0	1	2
1	0	0	1	0	2	0	0	1	2	0	0	2	2	1	2
1	2	2	1	1	0	2	1	1	2	2	2	2	0	2	1
1	0	1	0	2	2	2	1	0	1	1	2	2	1	0	2

Figure 3.8: Example of an optimal ternary code ($n = 16, y = 11, K = 12$), found using the K-packing factor graph of Section 3.5.2. Here each of $K = 12$ lines contains one code-word of length 16, and each two code-words are different in at least $y = 11$ digits.

$z_{i,j,k} = 1$ *iff* $x_{i-k} \neq x_{j-k}$:

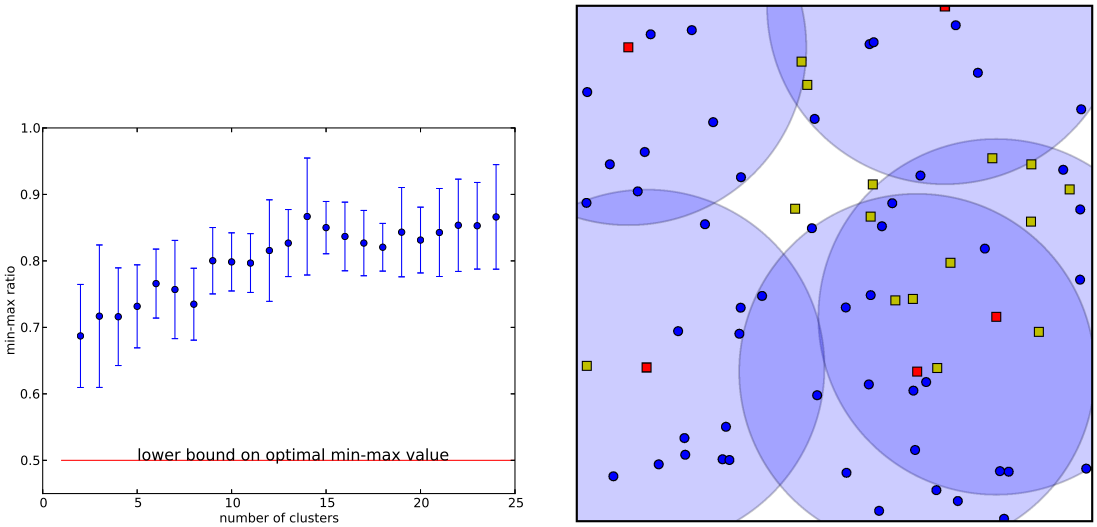
$$f(x_{i-k}, x_{j-k}, z_{i,j,k}) = \mathbb{I}(x_{i-k} \neq x_{j-k})\mathbb{I}(z_{i,j,k} = 1) + \mathbb{I}(x_{i-k} = x_{j-k})\mathbb{I}(z_{i,j,k} = 0).$$

This factor depends on three binary variables, therefore we can explicitly define its value for each of $2^3 = 8$ possible inputs.

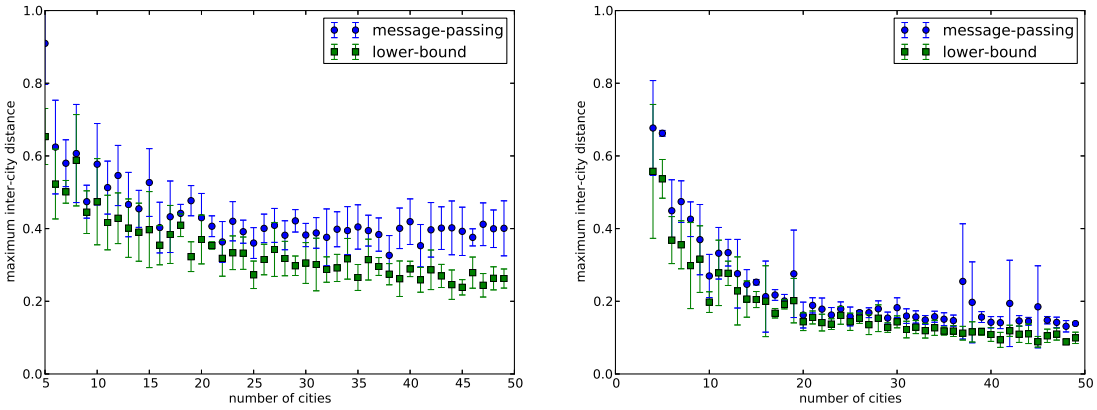
(b) *distance-factors:* For each $z_{i,j}$ define at-least-y-of-n factor:

$$f_{\mathcal{K}}(z_{i,j}) = \mathbb{I}(\sum_{1 \leq k \leq n} z_{i,j,k} \geq y)$$

Table 3.3 reports some optimal codes including codes with large number of bits n, recovered using this factor graph. Here PBP has used $T = 1000$ iterations.



(a) K-center (Euclidean) (b) Facility Location



(c) BTSP (Euclidean) (d) BTSP (Random)

Figure 3.9: Results: **(a)** K-center clustering of 50 random points in 2D plane with varying number of clusters (x-axis). The y-axis is the ratio of the min-max value obtained by message passing ($T = 500$ for PBP) over the min-max value of 2-approximation of [20]. **(b)** Min-max K-facility location formulated as an asymmetric K-center problem and solved using message passing. Squares indicate 20 potential facility locations and circles indicate 50 customers. The task is to select 5 facilities (squares) to minimize the maximum distance from any customer to a facility. The radius of circles is the min-max value. **(c,d)** The min-max solution for Bottleneck TSP with different number of cities (x-axis) for 2D Euclidean space as well as asymmetric random distance matrices ($T = 5000$ for PBP). The error-bars in all figures show one standard deviation over 10 random instances.

n	K	y	n	K	y	n	K	y	n	K	y
8	4	5	11	4	7	14	4	9	16	6	9
17	4	11	19	6	11	20	8	11	20	4	13
23	6	13	24	8	13	23	4	15	26	6	15
27	8	15	28	10	15	28	5	16	26	4	17
29	6	17	29	4	19	33	6	19	34	8	19
36	12	19	32	4	21	36	6	21	38	8	21
39	10	21	35	4	23	39	6	23	41	8	23
39	4	25	43	6	23	46	10	25	47	12	25
41	4	27	46	6	27	48	8	27	50	10	27
44	4	29	49	6	29	52	8	29	53	10	29

Table 3.3: Some optimal binary codes from [52] recovered by K-packing factor graph in the order of increasing y. n is the length of the code, K is the number of code-words and y is the minimum distance between code-words.

3.5.3 (Asymmetric) K-center problem

Given a pairwise distance matrix $D \in \Re^{N\times N}$, the K-center problem seeks a partitioning of nodes, with one center per partition such that the maximum distance from any node to the center of its partition is minimized. This problem is known to be NP-hard, even for Euclidean distance matrices [54]. In [24], max-product message passing is used to solve the min-sum variation of this problem –a.k.a. K-median problem. A binary variable factor-graph for the same problem is introduced in [31]. Here we introduce a binary variable model for the asymmetric K-center problem. Let $x = \{x_{1-1}, \ldots, x_{1-N}, x_{2-1}, \ldots, x_{2-N}, \ldots, x_{N-N}\}$ denote N^2 binary variables. Here $x_{i-j} = 1$ indicates that point i participates in the partition that has j as its center. We define the factors as follows:

A. *local factors:* $\forall i \neq j \quad f_{\{i-j\}}(x_{i-j}) = D_{i,j}x_{i-j} - \infty(1 - x_{i-j})$.

B. *uniqueness factors:* Every point follows at exactly one center (which can be itself). For every i consider $I = \{i - j \mid 1 \leq j \leq N\}$ and define $f_I(x_I) = \infty\mathbb{I}(\sum_{i-j\in\partial I} x_{i-j} \neq 1) - \infty\mathbb{I}(\sum_{i-j\in\partial I} x_{i-j} = 1)$.

C. *consistency factors:* Ensures that when j is selected as a center by any node i, node j also recognizes itself as a center. $\forall j, i \neq j$ define $f(x_{\{j-j,i-j\}}) = \infty\mathbb{I}(x_{j-j} = 0 \wedge x_{i-j} = 1) - \infty\mathbb{I}(x_{j-j} = 1 \vee x_{i-j} = 0)$.

D. *K-of-N factor:* Ensures than only K nodes are selected as centers. Let $\mathcal{K} = \{i-i \mid 1 \leq i \leq N\}$, define $f_{\mathcal{K}}(x_{\mathcal{K}}) = \infty\mathbb{I}(\sum_{i-i\in\mathcal{K}} x_{i-i} \neq K) - \infty\mathbb{I}(\sum_{i-i\in\mathcal{K}} x_{i-i} = K)$.

For variants of this problem such as the capacitated K-center, additional constraints on the maximum/minimum points in each group may be added as the at-least/at-most K-of-N factors.

We can significantly reduce the number of variables and the complexity (which is $\mathcal{O}(N^3 \log(N))$) by bounding the distance to the center of the cluster $\overline{y}$. Given an upper bound $\overline{y}$, we may remove all the variables x_{i-j} for $D_{i,j} > \overline{y}$ from the factor graph. Assuming that at most R nodes are at distance $D_{i-j} \leq \overline{y}$ from every node j, the complexity of min-max inference drops to $\mathcal{O}(NR^2 \log(N))$.

Figure 3.9(a) compares the performance of message-passing and the 2-approximation of [20] when the triangle inequality holds. The min-max facility location problem can also be formulated as an asymmetric K-center problem where the distance *to* all customers is ∞ and the distance from a facility to another facility is $-\infty$ (see Figure 3.9(b)).

The following proposition establishes the relation between the K-center factor graph above and dominating set problem as its CSP reduction. *The K-dominating set* of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a $\mathcal{D} \subseteq \mathcal{V}$ of size K, such that any node in $\mathcal{V} \setminus \mathcal{D}$ is adjacent to at least one member of D – *i.e.*, $\forall i \in \mathcal{V} \setminus \mathcal{D} \quad \exists j \in \mathcal{D}$ *s.t.* $(i, j) \in \mathcal{E}$.

Proposition 3.5.3 *For symmetric distance matrix $D \in \Re^{N\times N}$, the μ_y-reduction of the K-center factor graph above, is non-zero ($\mu_y(x) > 0$) iff x defines a K-dominating set for $\mathcal{G}(D, y)$.*

Note that in this proposition (in contrast with Propositions 3.5.1 and 3.5.2) the relation between the assignments x and K-dominating sets of $\mathcal{G}(D, y)$ is not one-to-one as several assignments may correspond to the same dominating set. Here we establish a similar relation between asymmetric K-center factor graph and set-cover problem.

Given universe set $\mathcal{V}$ and a set $\mathcal{S} = \{\mathcal{V}_1, \ldots, \mathcal{V}_M\}$ *s.t.* $\mathcal{V}_m \subseteq \mathcal{V}$, we say $\mathcal{C} \subseteq \mathcal{S}$ *covers* $\mathcal{V}$ *iff* each member of $\mathcal{V}$ is present in at least one member of $\mathcal{C}$ –*i.e.*, $\bigcup_{\mathcal{V}_m \in \mathcal{C}} \mathcal{V}_m = \mathcal{V}$. Now we consider a natural set-cover problem induced by any directed-graph. Given a directed-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for each node $i \in \mathcal{V}$, define a subset $\mathcal{V}_i = \{j \in \mathcal{V} \mid (j, i) \in \mathcal{E}\}$ as the set of all nodes that are connected *to* i. Let $\mathcal{S} = \{\mathcal{V}_1, \ldots, \mathcal{V}_N\}$ denote all such subsets. An *induced K-set-cover* of $\mathcal{G}$ is a set $\mathcal{C} \subseteq \mathcal{S}$ of size K that covers $\mathcal{V}$.

Proposition 3.5.4 *For a given asymmetric distance matrix $D \in \Re^{N \times N}$, the μ_y-reduction of the K-center factor graph as defined above, is non-zero ($\mu_y(x) > 0$)* iff *x defines an induced K-set-cover for $\mathcal{G}(D, y)$.*

3.5.4 (Asymmetric) bottleneck traveling salesman problem

Given a distance matrix $D \in \Re^{N \times N}$, the task in the bottleneck TSP is to find a tour of all N points such that the maximum distance between two consecutive cities in the tour is minimized ([43]). Any constant-factor approximation for arbitrary instances of this problem is $\mathcal{NP}$-hard ([62]).

Let $x = \{x_1, \ldots, x_N\}$ denote the set of variables where $x_i \in \mathcal{X}_i = \{0, \ldots, N-1\}$ represents the time step at which node i is visited. Also, we assume modular arithmetic (module N) on members of $\mathcal{X}_i$ –*e.g.*, $N \equiv 0 \mod N$ and $1 - 2 \equiv N - 1$

$$\begin{bmatrix} \infty & D_{i,j} & -\infty & \cdots & -\infty & -\infty & D_{j,i} \\ D_{j,i} & \infty & D_{i,j} & \cdots & -\infty & -\infty & -\infty \\ -\infty & D_{j,i} & \infty & \cdots & -\infty & -\infty & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -\infty & -\infty & -\infty & \cdots & \infty & D_{i,j} & -\infty \\ -\infty & -\infty & -\infty & \cdots & D_{j,i} & \infty & D_{i,j} \\ D_{i,j} & -\infty & -\infty & \cdots & -\infty & D_{j,i} & \infty \end{bmatrix}$$

Figure 3.10: The tabular form of $f_{\{i,j\}}(x_i, x_j)$ used for the bottleneck TSP.

mod N. For each pair x_i and x_j of variables, define the factor (Figure 3.5a)

$$f_{\{i,j\}}(x_i, x_j) = \infty\mathbb{I}(x_i = x_j) - \infty\mathbb{I}(|x_i - x_j| > 1) \tag{3.10}$$

$$+D_{i,j}\mathbb{I}(x_i = x_j - 1) + D_{j,i}\mathbb{I}(x_i = x_j - 1) \tag{3.11}$$

where the first term ensures $x_i \neq x_j$ and the second term means this factor has no effect on the min-max value when node i and j are not consecutively visited in a path. The third and fourth terms express the distance between i and j depending on the order of visit. Figure 3.10 shows the tabular form of this factor. In Appendix B we show an $\mathcal{O}(N)$ procedure to marginalize this type of factor.

Here we relate the min-max factor-graph above to a uniform distribution over Hamiltonian cycles.

Proposition 3.5.5 *For any distance matrix $D \in \Re^{N\times N}$, the μ_y-reduction of the bottleneck TSP factor-graph, defines a uniform distribution over the (directed) Hamiltonian cycles of $\mathcal{G}(D, y)$.*

Figure 3.9(c,d) reports the performance of message passing (over 10 instances) as well as a lower-bound on the optimal min-max value for tours of different length (N). Here we report the results for random points in 2D Euclidean space as well as asymmetric random distance matrices. For the symmetric case, the lower-bound is the maximum over j of the distance of two closest neighbors to each node j. For asymmetric random distance matrices, the maximum is over all of the minimum

length incoming edges and minimum length outgoing edges for each node. [5]

3.6 Applications using min-max propagation

Since BP with min-max operators is novel, we now showcase how we can efficiently pass min-max messages for two types of factors. We then show how these methods can be used to efficiently solve an application called makespan minimization.

3.6.1 Efficient update for high-order factors

When passing messages from factors to variables, we are interested in efficiently evaluating Equation 3.3. In its original form, this computation is exponential in the number of neighbouring variables $|\partial I|$. Since many interesting problems require high-order factors in their factor-graph formulation, many have investigated efficient min-sum and sum-product message passing through special family of, often sparse, factors ([72] and [66]).

For the time being, consider the factors over binary variables $x_i \in \{0, 1\} \forall i \in \partial I$ and further assume that efficient minimization of the factor f_I is possible. The function $f_I : x_{\partial I} \to \Re$ can be minimized in time $\mathcal{O}(\omega)$ with any subset $\mathcal{B} \subset \partial I$ of its variables fixed. In the following we show how to calculate min-max factor-to-variable messages in $\mathcal{O}(K(\omega + \log(K)))$, where $K = |\partial I| - 1$. In comparison to the limited settings in which high-order factors allow efficient min-sum and sum-product inference, we believe this result to be quite general.

The idea is to break the problem in half, at each iteration. We show that for one of these halves, we can obtain the min-max value using a single evaluation of f_I. By

[5] If for one node the minimum length incoming and outgoing edges point to the same city, the second minimum length incoming and outgoing edges are also considered in calculating a tighter bound.

reducing the size of the original problem in this way, we only need to choose the final min-max message value from a set of candidates that is at most linear in $|\partial I|$.

Procedure. According to Equation 3.3, in calculating the factor-to-variable message $\nu_{I\to i}(x_i)$ for a fixed $x_i = c$, we are interested in efficiently solving the following optimization problem

$$\min_{x_{\partial I\setminus i}} \max \left(\nu_{1\to I}(x_1), \nu_{2\to I}(x_2), ..., \nu_{K\to I}(x_K), f_I(x_{\partial I\setminus i}, x_i = c_i)\right) \tag{3.12}$$

where, without loss of generality we are assuming $\partial I \setminus i = \{1, \ldots, K\}$, and for better readability, we drop the index I, in factor f_I, and the message destination (i.e. $\to I$ for the incoming messages and $\to i$ for the outgoing message).

There are 2^K configurations of $x_{\partial I\setminus i}$, one of which is the minimizing solution. We divide this set in half in each iteration and save the minimum in one of these halves in the min-max candidate list $\mathcal{C}$. The maximization part of the expression is equivalent to $\max\left(\max\left(\nu_1(x_1), \nu_2(x_2), ..., \nu_K(x_K)\right), f(x_{\partial I}, x_i = c_i)\right)$.

Let $\nu_{j_1}(c_{j_1})$ be the largest ν value that is obtained at some index j_1, for some value $c_{j_1} \in \{0, 1\}$. In other words, $\nu_{j_1}(c_{j_1}) = \max\left(\nu_1(0), \nu_1(1), ..., \nu_K(0), \nu_K(1)\right)$. For future use, let $j_2, \ldots, j_M$ be the index of the next largest message indices up to the K largest ones, and let $c_{j_2}, \ldots, c_{j_K}$ be their corresponding assignment. Note that the same message (*e.g.*, $\nu_3(0), \nu_3(1)$) could appear in this sorted list at different locations.

We then partition the set of all assignments to $x_{\partial I\setminus i}$ into two sets of size 2^{K-1} depending on the assignment to x_{j_1}: 1) $x_{j_1} = c_{j_1}$ or; 2) $x_{j_1} = 1 - c_{j_1}$. The minimization of Equation 3.12 can also be divided to two minimizations each having x_{j_1} set to a different value. For $x_{j_1} = c_{j_1}$, Equation 3.12 simplifies to

$$\nu^{(j_1)} = \max\left(\nu_{j_1}(c_{j_1}), \min_{x_{\partial I\setminus\{i,j_1\}}} \left(f(x_{\partial I\setminus\{i,j_1\}}, x_i = c_i, x_{j_1} = c_{j_1})\right)\right) \tag{3.13}$$

where we need to minimize f, subject to a fixed x_i, x_{j_1}. We repeat the procedure above at most K times, for $j_1, \ldots, j_k, \ldots j_K$, where at each iteration we obtain a candidate solution $\nu^{(j_m)}$ that we add to the candidate set $\mathcal{C} = \{\nu^{(j_1)}, \ldots, \nu^{(j_K)}\}$. The final solution is the smallest value in the candidate solution set, $\min \mathcal{C}$.

Early Termination. If $j_k = j_{k'}$ for $1 \leq k, k' \leq K$ it means that we have performed the minimization of Equation 3.12 for both $x_{j_k} = 0$ and $x_{j_k} = 1$. This means that we can terminate the iterations and report the minimum in the current candidate set. Adding the cost of sorting $\mathcal{O}(K \log(K))$ to the worst-case cost of minimization of $f()$ in Equation 3.13 gives a total cost of $\mathcal{O}(K(\log(K) + \omega))$.

Arbitrary Discrete Variables. This algorithm is not limited to binary variables. The main difference in dealing with cardinality $D > 2$, is that we run the procedure for at most $K(D-1)$ iterations, and in early termination, all variable values should appear in the top $K(D-1)$ incoming message values.

Functions f that are easy to minimize

Now, we need to address the issues of minimizing $f(x)$ subject to a given set of constraints, fixing values of certain x_is. There are several forms of f for which this evaluation is easy. We explain some examples.

Functions of the form $f(x) = \sum_i \alpha_i x_i$. In the simplest case, where $\alpha_i = 1$, we are counting the number of variables set to one. Therefore, the minimum is obtained by setting all unconstrained variables to zero. The value obtained is equal to the number of constrained variables already fixed to one. With positive coefficients, again, the minimum is obtained by setting all the unconstrained variables to zero, and the value obtained is the weighted sum of the constrained variables fixed to one. Thus, generally, unconstrained variables are fixed to zero if their coefficient is positive, and to one if it is negative. The value obtained is the weighted sum of all coefficients

whose variables are fixed to one, constrained and unconstrained.

Functions of the form $f(x) = g\left(\sum_i \alpha_i x_i\right)$. These can be solved if g is either monotonically increasing or monotonically decreasing. In the former case, all unconstrained variables with positive coefficients are set to zero. All unconstrained variables with negative coefficients are set to one. The value is found by calculating the corresponding g value. In the latter case, all unconstrained variables with positive coefficients are set to one, and all unconstrained variables with negative coefficients are set to zero.

For some factors, we can go further and calculate *all* factor-to-variable messages leaving f_I in a time linear in $|\partial I|$. The following section derives such update rule for a type of factor that we use in the make-span application of section 3.6.3.

3.6.2 Choose-one constraint

If $f_I(X_{\partial I})$ implements a constraint such that only a subset of configurations $X_{\mathcal{A}} \subset \mathcal{X}_{\partial I}$, of the possible configurations are allowed, then the message from function f_I to x_i simplifies to (where for simplification here again we remove $\rightarrow I$ for the incoming messages)

$$\nu_{I\rightarrow i}(x_i') = \min_{X_{\mathcal{A}}|x_i=x_i'} \max_{j\in\partial I\setminus i} \nu_{j\rightarrow I}(x_j) \tag{3.14}$$

In many applications, this can be further simplified by taking into account properties of the constraints. Here, we describe such a procedure for factors which enforce that exactly one of their binary variables be set to one and all others to zero. Consider the constraint $f(x_1, ..., x_K) = \infty\left(1 - \mathbb{I}(\sum_k x_k = 1)\right)$ for binary variables $x_k \in \{0, 1\}$.

Using $X_{\setminus i} = (x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_K)$ for X with x_i removed, Equation 3.14

becomes

$$\begin{aligned}\nu_{I\to i}(x_i) &= \min_{X_{\setminus i}|\sum_{k=1}^{K} x_k=1} \max_{k|k\neq i} \nu_k(x_k) \\ &= \begin{cases} \max_{k|k\neq i} \nu_k(0) & \textbf{if} \quad x_i = 1 \\ \min_{X_{\setminus i}\in\{(1,0,\ldots,0),(0,1,\ldots,0),\ldots,(0,0,\ldots,1)\}} \max_{k|k\neq i} \nu_k(x_k) & \textbf{if} \quad x_i = 0 \end{cases}\end{aligned} \tag{3.15}$$

Naive implementation of the above update is $\mathcal{O}(K^2)$ for each x_i , or $\mathcal{O}(K^3)$ for sending messages to all neighbouring x_i. However, further simplification is possible. Consider the calculation of $\max_{k|k\neq i} \nu_k(x_k)$ for $X_{\setminus i} = (1, 0, \ldots, 0)$ and $X_{\setminus i} = (0, 1, \ldots, 0)$. All but the first two terms in these two sets are the same (all zero), so most of the comparisons that were made when computing $\max_{k|k\neq i} \nu_k(x_k)$ for the first set, can be reused when computing it for the second set. This extends to all $K-1$ sets $(1, 0, \ldots, 0), \ldots, (0, 0, \ldots, 1)$, and also extends across the message updates for different x_i's. After examining the shared terms in the maximizations, we see that all that is needed is

$$k_i^{(1)} = \arg\max_{k|k\neq i} \nu_k(0), \quad k_i^{(2)} = \arg \max_{k|k\neq i, k_i^{(1)}} \nu_k(0), \tag{3.16}$$

the indices of the maximum and second largest values of $\nu_k(0)$ with i removed from consideration. Note that these can be computed for all neighbouring x_i in time linear in K, by finding the top three values of $\nu_k(0)$ and selecting two of them appropriately depending on whether $\nu_i(0)$ is among the three values. Using this notation, the above

update simplifies as follows:

$$\nu_{I\to i}(x_i) = \begin{cases} \nu_{k_i^{(1)}}(0) \quad \textbf{if} \quad x_i = 1 \\ \min\left(\min_{k|k\neq i,k_i^{(1)}} \max(\nu_{k_i^{(1)}}(0), \nu_k(1)), \max(\nu_{k_i^{(1)}}(1), \nu_{k_i^{(2)}}(0))\right) \quad \textbf{if} \quad x_j = 0 \end{cases}$$
$$= \begin{cases} \mu_{k_{ai}^{(1)}}(0) \quad \textbf{if} \quad x_i = 1 \\ \min\left(\max(\nu_{k_i^{(1)}}(0), \min_{k|k\neq i,k_i^{(1)}} \nu_k(1)), \max(\nu_{k_i^{(1)}}(1), \nu_{k_i^{(2)}}(0))\right) \quad \textbf{if} \quad x_i = 0 \end{cases} \tag{3.17}$$

The term $\min_{k|k\neq i,k_i^{(1)}} \nu_k(1)$ also need not be recomputed for every x_i, since terms will be shared. The index of the smallest value of $\nu_k(1)$ with i and $k_i^{(1)}$ removed from consideration is as follows:

$$s_i = \arg \min_{k\neq i,k_i^{(1)}} \nu_k(1). \tag{3.18}$$

This can be computed efficiently for all i in time that is linear in K by finding the smallest three values of $\nu_k(1)$ and selecting one of them appropriately depending on whether $\nu_i(1)$ and/or $\nu_{k_i^{(1)}}$ are among the three values. The resulting message update for K-choose-1 constraint becomes

$$\nu_{I\to i}(x_i) = \begin{cases} \nu_{k_i^{(1)}}(0) \quad \textbf{if} \quad x_i = 1 \\ \min\left(\max(\nu_{k_i^{(1)}}(0), \nu_{s_i}(1)), \max(\nu_{k_i^{(1)}}(1), \nu_{k_i^{(2)}}(0))\right) \quad \textbf{if} \quad x_i = 0 \end{cases} \tag{3.19}$$

This shows that messages to all neighbouring variables $x_1, ..., x_K$ can be obtained in time that is linear in K. This type of constraint also has a tractable form in min-sum and sum-product inference, albeit of a different form [27, 37].

3.6.3 Makespan minimization

The objective in the makespan problem is to schedule a set of given jobs, each with a load, on machines which operate in parallel such that the total load for the machine

which has the largest total load (*i.e.*, the makespan) is minimized [65].

Given N distinct jobs $\mathcal{N} = \{1, \ldots, n, \ldots, N\}$ and M machines $\mathcal{M} = \{1, \ldots, m, \ldots, M\}$, where p_{nm} represents the load of job n on machine m, we denote the assignment variable x_{nm} as whether or not job n is assigned to machine m. The task is to find the set of assignments $x_{nm} \, \forall \, n \in \mathcal{N}, \, \forall \, m \in \mathcal{M}$ which minimizes the total cost function below, while satisfying the associated set of constraints:

$$\min_{\mathcal{X}} \max_{m} \left(\sum_{n=1}^{N} p_{nm} x_{nm} \right) \quad \text{s.t.} \quad \sum_{m=1}^{M} x_{nm} = 1 \quad x_{nm} \in \{0, 1\} \quad \forall \, n \in \mathcal{N}, m \in \mathcal{M} \tag{3.20}$$

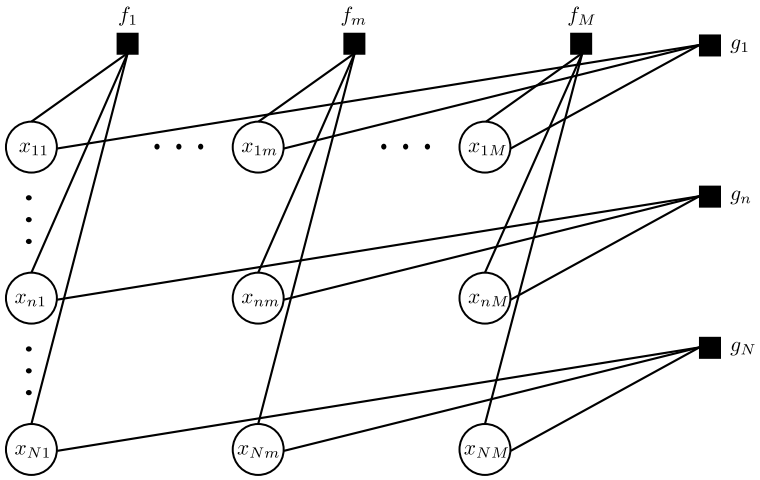


Figure 3.11: Factor graph for the makespan minimization problem.

The makespan minimization problem is NP-hard for $M = 2$ and strongly NP-hard for $M > 2$ [28]. Two well-known approximation algorithms are the 2-approximation greedy algorithm and the 4/3-approximation greedy algorithm. In the former, all machines are initialized as empty. We then select one job at random and assign it to the machine with least total load given the current job assignments. We repeat this process until no jobs remain. This algorithm is guaranteed to give a schedule with a makespan no more than 2 times larger than the one for the optimal schedule [7, 8] The

4/3-approximation algorithm, a.k.a. the Longest Processing Time (LPT) algorithm, operates similar to the 2-approximation algorithm with the exception that, at each iteration, we always take the job with the next largest load rather than selecting one of the remaining jobs at random [34].

Factor graph representation

Figure 3.11 shows the factor-graph with binary variables x_{nm}, where the factors are:

$$f_m(x_{1m}, \ldots, x_{Nm}) = \sum_{n=1}^{N} p_{nm} x_{nm} \tag{3.21}$$

$$g_n(x_{n1}, \ldots, x_{nM}) = \infty \left(1 - \mathbb{I}\left(\sum_{m=1}^{M} x_{nm} = 1\right)\right) \tag{3.22}$$

where f computes the total load for a machine and g enforces the constraint in Equation 3.20. It is easy to see that following min-max inference problem over this factor-graph minimizes the makespan

$$\min_{\mathcal{X}} \max \left(\max_{m} f_m(x_{1m}, ..., x_{Nm}), \max_{n} g_n(x_{n1}, ..., x_{nM})\right) \tag{3.23}$$

Using the procedure for passing messages through the g constraints in section 3.6.2 and using the procedure of section 3.6.1 for f, we can efficiently approximate the min-max solution of Equation 3.23 by message passing. Note that the factor f in the sum-product reduction of this factor-graph has a non-trivial form that does not allow efficient message update.

	M	N	Dist
E1	3,4,5	2M,3M,5M	U(1,20), U(20,50)
E2	2,3,4,6,8,10	10,30,50,100	U(100,800)
E3	3,5,8,10	3M+1,3M+2,4M+1, 4M+2, and 5M+2	U(1,100),U(100,200)
E4	2 3	9 10	U(1,20), U(20,50), U(50,100), U(100,200), U(100,800)

Table 3.4: Set of benchmark experiments for identical machines makespan minimization

Results

In an initial set of experiments, we compare MMP (with different decimation procedures) against LPT on a set of benchmark experiments designed in [36] for the identical machine version of the problem – *i.e.*, a task has the same processing time on all machines ($p_n = p_{1n} = p_{2n} = .. = p_{Mn}$).

In total there are four families our benchmark experiments, as shown in Table 3.4, where for each family all combinations of specified number of jobs N, machines M, and distributions from which the jobs are drawn are tested. The only exception to this is $E4$ where the $M = 2, N = 9$ combination is tested separately from the $M = 3, N = 10$. For all experiments the processing times are drawn from a uniform distribution with varying bounds.

The experiments are designed to set different difficulties of the problem. For example, $E1$ and $E4$ are designed to test small problem instances. In contrast, $E2$ and $E3$ are setup such as to test larger instance. For all experiments, the performance number reported is the ratio of the makespan achieved by the method being tested to a lower bound LB computed as:

$$LB = \max\left(\max_{1\leq i\leq n} p_n, \sum_{n=1}^{N} p_n/M\right). \tag{3.24}$$

Table 3.5 shows the scenario where MMP performs best against the LPT algorithm. We see that this scenario involves large instances. From this table, we also see that max-support decimation almost always outperforms the other decimation schemes. From the additional results in Tables 3.6 to 3.10, we see that our framework does not perform as well on small instances).

We then test the MMP with max-support decimation against a more difficult version of the problem: the unrelated machine model, where each job has a different processing time on each machine. Specifically, we compare our method against that of [75] which also uses distributive law for min-max inference to solve a load balancing problem. However, that paper studies a sparsified version of the unrelated machines problem where tasks are restricted to a subset of machines (*i.e.*, they have infinite processing time for particular machines). Nevertheless, we can still compare their results to what we can achieve using MMP using infinite-time constraints.

We use the same problem setup with three different ways of generating the processing times (uncorrelated, machine correlated, and machine/task correlated) and compare our answers to IBM's CPLEX solver exactly as the authors do in that paper (where a high ratio is better). Table 3.11 shows a summary of the processing time generation for each of the different ways (note that all normal distributions are bounded at the low end to have a minimum of 10).

Tables 3.12 to 3.14 show the results. Here again, min-max propagation works best for large instances. While the results of both methods are comparable, the main advantage of our method is that it makes no prior assumption about the sparsity of a

M	N	LPT	Min-Max Prop (Random Dec.)	Min-Max Prop (Max-Support Dec.)	Min-Max Prop (Min-Value Dec.)
3	10	1.127	1.124	1.078	**1.071**
	11	**1.067**	1.106	1.083	1.076
	13	1.098	1.112	**1.031**	1.044
	14	**1.052**	1.054	1.069	1.056
	16	1.083	**1.041**	1.048	1.047
	17	1.040	1.057	1.048	**1.032**
5	16	1.162	1.173	**1.098**	1.099
	17	1.123	1.178	1.097	**1.082**
	21	1.113	1.108	**1.055**	1.062
	22	1.092	1.116	1.084	**1.057**
	26	1.102	1.070	1.062	**1.053**
	27	1.073	1.075	**1.044**	1.048
8	25	1.178	1.183	**1.091**	1.128
	26	1.144	1.167	**1.079**	1.112
	33	1.135	1.144	**1.081**	1.093
	34	1.117	1.132	**1.071**	1.086
	41	1.112	1.117	**1.055**	1.077
	42	1.094	1.109	1.079	**1.074**
10	31	1.184	1.168	1.110	**1.105**
	32	1.165	1.186	**1.109**	1.111
	41	1.138	1.183	**1.077**	1.088
	42	1.124	1.126	**1.074**	1.090
	51	1.112	1.131	**1.077**	1.081
	52	1.102	1.100	**1.051**	1.076

Table 3.5: Makespan minimization experiment 3 with U(100,200). Min-max ratio to a lower bound (lower is better) obtained by LPT with 4/3-approximation guarantee versus min-max propagation using different decimation procedures. N is the number of jobs and M is the number of machines. In this setting, all jobs have the same run-time across all machines.

M	N	LPT	Min-Max Prop (Random Dec.)	Min-Max Prop (Max-Support Dec.)	Min-Max Prop (Min-Value Dec.)
3	6	**1.115**	1.197	1.132	1.140
	9	**1.052**	1.098	1.069	1.090
	15	**1.015**	1.057	1.035	1.050
4	8	**1.083**	1.242	1.128	1.178
	12	**1.048**	1.116	1.072	1.089
	20	**1.024**	1.052	1.036	1.049
5	10	**1.122**	1.230	1.144	1.186
	15	**1.051**	1.127	1.089	1.102
	25	**1.019**	1.061	1.029	1.053

Table 3.6: Makespan minimization experiment 1 with U(1,20).

M	N	LPT	Min-Max Prop (Random Dec.)	Min-Max Prop (Max-Support Dec.)	Min-Max Prop (Min-Value Dec.)
3	6	**1.048**	1.111	1.082	1.087
	9	**1.038**	1.063	1.057	1.069
	15	**1.012**	1.126	1.042	1.044
4	8	**1.055**	1.191	1.103	1.111
	12	**1.024**	1.135	1.071	1.074
	20	**1.006**	1.079	1.050	1.048
5	10	**1.049**	1.134	1.091	1.116
	15	**1.021**	1.142	1.080	1.112
	25	**1.007**	1.064	1.053	1.053

Table 3.7: Makespan minimization experiment 1 with U(20,50).

N	M	LPT	Min-Max Prop (Random Dec.)	Min-Max Prop (Max-Support Dec.)	Min-Max Prop (Min-Value Dec.)
10	2	**1.011**	1.018	1.022	1.022
	3	1.045	1.053	1.047	**1.035**
	4	**1.089**	1.145	1.108	1.092
	6	**1.103**	1.252	1.132	1.182
	8	**1.027**	1.135	1.043	1.046
	10	**1.000**	**1.000**	**1.000**	**1.000**
30	2	**1.001**	1.010	1.006	1.006
	3	**1.004**	1.046	1.014	1.015
	4	**1.017**	1.067	1.016	1.024
	6	**1.015**	1.073	1.032	1.062
	8	**1.042**	1.155	1.054	1.098
	10	**1.041**	1.223	1.073	1.128
50	2	**1.000**	1.004	1.004	1.004
	3	**1.006**	1.038	1.007	1.010
	4	**1.010**	1.038	1.011	1.014
	6	**1.019**	1.066	1.029	1.041
	8	**1.022**	1.091	1.044	1.047
	10	**1.017**	1.157	1.073	1.067
100	2	**1.000**	1.003	1.001	1.001
	3	**1.004**	1.007	1.004	1.003
	4	**1.000**	1.022	1.008	1.007
	6	**1.005**	1.057	1.024	1.016
	8	**1.010**	1.070	1.065	1.022
	10	**1.004**	1.063	1.107	1.038

Table 3.8: Makespan minimization experiment 2.

M	N	LPT	Min-Max Prop (Random Dec.)	Min-Max Prop (Max-Support Dec.)	Min-Max Prop (Min-Value Dec.)
3	10	**1.040**	1.065	1.059	1.063
	11	**1.038**	1.079	1.040	1.041
	13	**1.023**	1.045	1.039	1.050
	14	**1.021**	1.062	1.059	1.044
	16	**1.016**	1.050	1.029	1.033
	17	**1.013**	1.034	1.024	1.030
5	16	**1.031**	1.129	1.075	1.104
	17	**1.038**	1.116	1.066	1.112
	21	**1.032**	1.122	1.078	1.078
	22	**1.028**	1.106	1.035	1.059
	26	**1.017**	1.101	1.065	1.038
	27	**1.018**	1.093	1.049	1.050
8	25	**1.036**	1.141	1.078	1.121
	26	**1.045**	1.202	1.082	1.155
	33	**1.029**	1.126	1.049	1.097
	34	**1.038**	1.087	1.061	1.088
	41	**1.020**	1.081	1.068	1.071
	42	**1.021**	1.130	1.049	1.069
10	31	**1.057**	1.211	1.116	1.154
	32	**1.050**	1.144	1.107	1.148
	41	**1.028**	1.133	1.103	1.102
	42	**1.026**	1.164	1.073	1.114
	51	**1.020**	1.119	1.035	1.070
	52	**1.013**	1.122	1.042	1.075

Table 3.9: Makespan minimization experiment 3 with U(1,100).

M,N	U(,)	LPT	Min-Max Prop (Random Dec.)	Min-Max Prop (Max-Support Dec.)	Min-Max Prop (Min-Value Dec.)
2,9	U(1,20)	**1.016**	1.036	1.026	1.026
	U(20,50)	1.062	1.048	**1.028**	**1.028**
	U(1,100)	1.017	**1.016**	1.034	1.034
	U(50,100)	1.072	1.055	**1.042**	**1.042**
	U(100,200)	1.073	**1.051**	1.055	1.055
	U(100,800)	1.028	1.027	**1.026**	**1.026**
3,10	U(1,20)	**1.031**	1.127	1.057	1.076
	U(20,50)	1.103	1.095	**1.052**	1.075
	U(1,100)	**1.033**	1.096	1.039	1.048
	U(50,100)	1.142	1.120	**1.083**	1.068
	U(100,200)	1.127	1.133	**1.083**	1.094
	U(100,800)	**1.049**	1.056	1.061	1.078

Table 3.10: Makespan minimization experiment 4.

Mode	Dist
Uncorrelated (Mode 0)	$N(100, 10)$
Machine correlated (Mode 1)	$N(\alpha_i, 10)$ where $\alpha_i \sim U(50, 100)$ and α_i is different for each machine
Machine/Task correlated (Mode 2)	$N(\beta_k + \alpha_i, 10)$ where $\alpha_i, \beta_k \sim U(50, 100)$, α_i is different for each machine, and β_i is different for each job

Table 3.11: Processing time generation for unrelated machine makespan minimization

Mode	N/M	Vinyals [75]	Min-Max Prop
0	10	**0.94(0.01)**	0.92(0.01)
	15	**0.93(0.00)**	0.90(0.02)
1	10	**0.92(0.00)**	0.87(0.05)
	15	**0.89(0.00)**	0.82(0.02)
2	10	**0.87(0.00)**	0.85(0.05)
	15	**0.85(0.00)**	**0.85(0.04)**

Table 3.12: Makespan minimization with M=40. Min-max ratio (to that of LP relaxation), for min-max propagation versus the methods of [75] (higher is better). Mode 0, 1 and 2 corresponds to uncorrelated, machine correlated and machine-task correlated respectively.

Mode	N/M	Vinyals [75]	Min-Max Prop
0	10	**0.94(0.01)**	0.93(0.00)
	15	**0.94(0.00)**	0.89(0.03)
1	10	**0.90(0.00)**	0.87(0.01)
	15	**0.87(0.01)**	0.85(0.02)
2	10	0.87(0.01)	**0.89(0.03)**
	15	**0.87(0.01)**	0.79(0.09)

Table 3.13: Makespan minimization with M=60.

Mode	N/M	Vinyals [75]	Min-Max Prop
0	5	0.93(0.03)	**0.95(0.01)**
	10	**0.94(0.01)**	0.93(0.01)
	15	**0.94(0.00)**	0.90(0.01)
1	5	**0.90(0.01)**	0.86(0.07)
	10	**0.90(0.00)**	0.88(0.00)
	15	**0.87(0.01)**	0.73(0.03)
2	5	0.81(0.01)	**0.89(0.01)**
	10	0.81(0.01)	**0.89(0.01)**
	15	0.78(0.01)	**0.86(0.01)**

Table 3.14: Makespan minimization with M=80.

makespan instance nor requires such an assumption to solve it. The sparsity restriction of [75], allows for decomposition of their loopy graph into an almost equivalent tree structure, something which cannot be done for non-sparse instances. Instead, our factor graph formulation can accept any problem, whether sparse or not. It then places the burden on MMP and leverages its capability of iteratively passing these messages on a loopy graph to find a close-to-optimal schedule.As we've shown throughout this chapter, both MMP and iterative message passing are concepts that extend well beyond makespan minimization and require no such assumption about a specific makespan instance to solve it.

Chapter 4

Survey propagation beyond constraint satisfaction problems

In chapter 2 we mentioned that a loopy factor graph may contain more than one fixed point and that running LBP on it captures at most one of these fixed points at a time. In that same chapter, we presented SP as a method capable of performing a second summation across all BP fixed points. We showed how SP was able to find solutions to SAT problems when the solution space fractioned into clusters by summing across those clusters.We now derive an efficient SP implementation for marginalization on binary loopy pairwise factor graphs and show that it is capable of tracking many fixed points, thereby beating out BP on problems where multiple fixed points exist. The difference here is that the BP message space over which SP operates is now continuous due to the factors in the BP factor graph not solely containing hard constraints as is the case for CSPs - where SP is currently developed. This difference introduces numerical difficulties which we tackle throughout the chapter to produce our efficient version of SP for problems beyond CSPs.

4.1 Model

Starting from the SP equations outlined in chapter 2, we now derive efficient SP updates for a pairwise binary model. We make this choice of model because, for marginalization, if there are no hard constraints, any factor graph can be transformed to such a model [21]. The factors in any such factor graph can then be transformed to so-called Ising factors. The Ising model defines $p(x) \propto \prod_{ij} e^{x_i J_{ij} x_j} \prod_i e^{x_i \theta_i}$, where $x_i \in \{-1, 1\}$. This joint form is a product of two types of factors: local factors $f_i(x_i) = e^{x_i \theta_i}$, and pairwise factors $f_{ij}(x_i, x_j) = e^{x_i x_j J_{ij}}$.

Figure 4.1 shows a portion of the BP and SP factor graphs for the Ising model. For an N-variable Ising model where the degree of each variable node is D, one iteration of LBP is $\mathcal{O}(ND)$ or the number of edges in the Ising model.

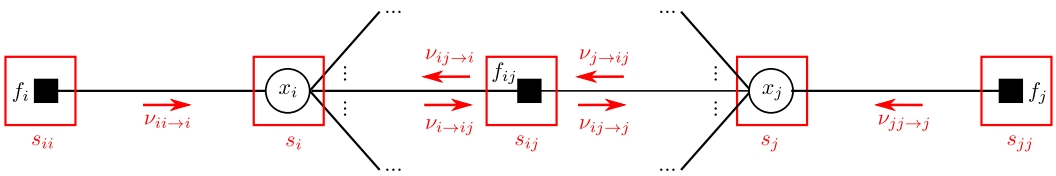


Figure 4.1: Portion of BP and SP factor graphs for the Ising model: BP factor graph is in black and SP factors and messages are in red. We refer to variable-type factors in the SP factor graph as variables and factor-type-factors as factors.

4.2 Approximation scheme

We now highlight the challenges in deriving efficient SP for marginalization on the Ising model when starting from the generic SP messages.

4.2.1 Quantization

Since the variables are binary, each BP message can be represented by a scalar, the ratio $b(x_i = 1)/b(x_i = -1)$. However, this ratio is real-valued which implies that

the SP messages would be continuous distributions over the domain $[0, +\infty)$. A first attempt is to quantize, as shown in Figure 4.2, this continuous space (where ∞ is replaced by a sufficiently large number) into K linearly spaced bins where each bin index k represents the ratio $r_k = b(x_i = 1)/b(x_i = -1)$ of BP message values for a particular BP message.

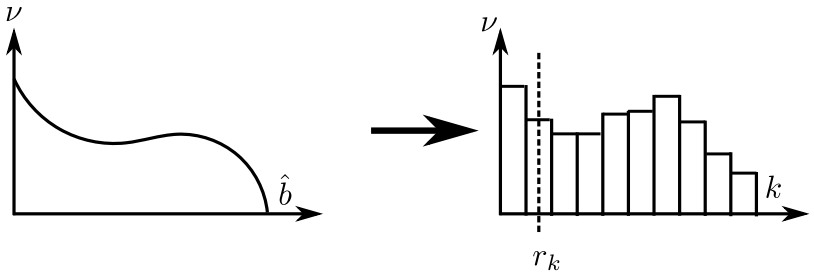


Figure 4.2: Quantization of SP message.

4.2.2 Representing an SP message

Sending out an SP message from a variable with degree D involves the combination of $(D-1)+1$ (*i.e.*, the $+1$ coming from the local factor) incoming SP messages where each message has length K. Calculating Equation 2.32 is therefore $\mathcal{O}(K^D)$, as one has to consider all combinations of bins (BP messages) from all incoming SP messages. Repeating this for all D outgoing SP messages from each variable is $\mathcal{O}(DK^D)$.

4.2.3 Logarithmic quantization & convolution

To improve this exponential time-complexity, we change the quantization such that each bin index k now represents the log-ratio $r_k = \ln(b(x_i = 1)/b(x_i = -1))$ of BP message values. Therefore, $b(x_i = -1) = 1 - \sigma(k)$ and $b(x_i = 1) = \sigma(k)$ where $\sigma(k) = e^{r_k}/(e^{r_k} + 1)$, and the center bin represents a log-ratio of 0. By working in the log-message space for BP, computing BP marginals and passing BP messages through

variables now involves "summation" of the log-messages rather than "multiplication" of the original ones. For SP, this enables convolution of the SP messages at the variables.

4.2.4 Using frequency domain

Rather than convolving SP messages at the variables, we go one step further and track their frequency transforms and multiply those. At every iteration, we send out messages from all variables by *multiplying* the frequency transforms of all incoming SP messages at each variable, which yields the *SP marginals* in the frequency domain, and then *divide* out the incoming frequency domain SP message on the edge for which we want to send out the message. Sending out a message from a variable with degree D involves the convolution of $(D-1)+1$ incoming SP messages of length K. In the frequency domain this implies that the transforms of the SP messages must have minimum length of $D(K-1)+1$ to avoid *aliasing*. Thus, sending out all messages from a single variable in the frequency domain involves multiplying $D+1$ incoming messages of length $D(K-1)+1$ then dividing out D of them, one at a time, resulting in a complexity of $\mathcal{O}(D^2K)$. Sending out messages for all variables is thus $\mathcal{O}(ND^2K)$.

4.2.5 Variable-to-factor SP message

Algorithm 1 summarizes the variable to factor subroutine. Here, the tuple $\nu_{ij\to i} = (\nu_{ij\to i}(r_1), \ldots, \nu_{ij\to i}(r_K))$ denotes a SP message of length K and the tuple $\sigma = (\sigma(1), \ldots, \sigma(K))$ contains the normalized BP probabilities for all K bins. Here, the product of two tuples (*e.g.*, $\sigma\,\nu_{ij\to i}$) is their element-wise product. $\mathcal{F}(.)$ and $\mathcal{F}^{-1}(.)$ represent taking *frequency* and *inverse frequency* transforms respectively. The result of $\mathcal{F}^{-1}(.)$ has length $D(K-1)+1$ since the frequency transforms of the SP messages

also have the same length. We reduce it to K bins by making an approximation where quantities assigned beyond the two most extreme bins are grouped into those two bins.

Algorithm 1 SP variable to factor subroutine

Input: incoming messages $\{\nu_{ij\to i}\}_{j\in\partial i}$ to node i
Output: outgoing messages $\{\nu_{i\to ij}\}_{j\in\partial i}$ from node i
for $ij \in \partial i$ **do**
  $\nu^0_{ij\to i} \leftarrow \mathcal{F}\left((1-\sigma)\nu_{ij\to i}\right)$
  $\nu^1_{ij\to i} \leftarrow \mathcal{F}\left(\sigma\,\nu_{ij\to i}\right)$
end
$\nu^0_i \leftarrow \prod_{ij\in\partial i} \nu^0_{ij\to i}$
$\nu^1_i \leftarrow \prod_{ij\in\partial i} \nu^1_{ij\to i}$
for $ij \in \partial i$ **do**
  $\nu_{i\to ij} \leftarrow \mathcal{F}^{-1}\left(\nu^0_i/\nu^0_{ij\to i} + \nu^1_i/\nu^1_{ij\to i}\right)$
end

By substituting the terms in Algorithm 1 into its last line, we see that:

$$\nu_{i\to ij} \propto \overset{*}{\prod_{gh\in\partial i\setminus ij}} (1-\sigma)\nu_{gh\to i} + \overset{*}{\prod_{gh\in\partial i\setminus ij}} \sigma\nu_{gh\to i} \tag{4.1}$$

where $\prod^*_{gh\in\partial i\setminus ij}$ denotes convolution over the set $gh \in \partial i \setminus ij$.

To show that the above is correct, we have, starting from Equation 2.34:

$$s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=-1) = \prod_{gh\in\partial i\setminus ij} \hat{b}_{gh\to i}(x_i=-1) = \prod_{gh\in\partial i\setminus ij} (1-\sigma(k_{gh\to i})) \tag{4.2}$$

$$s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=1) = \prod_{gh\in\partial i\setminus ij} \hat{b}_{gh\to i}(x_i=1) = \prod_{gh\in\partial i\setminus ij} (\sigma(k_{gh\to i})) \tag{4.3}$$

$$s_{i\to ij}(k_{\partial i\setminus ij\to i})(\emptyset) = s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=-1) + s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=1) \tag{4.4}$$

$$= \prod_{gh\in\partial i\setminus ij}(1-\sigma(k_{gh\to i})) + \prod_{gh\in\partial i\setminus ij}(\sigma(k_{gh\to i})) \tag{4.5}$$

$$\ln\left(\frac{s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=1)}{s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=-1)}\right) = \ln\left(\frac{\prod_{gh\in\partial i\setminus ij}(\sigma(k_{gh\to i}))}{\prod_{gh\in\partial i\setminus ij}(1-\sigma(k_{gh\to i}))}\right) \tag{4.6}$$

$$= \ln\prod_{gh\in\partial i\setminus ij}(e^{r_{k_{gh\to i}}}) = \sum_{gh\in\partial i\setminus ij} r_{k_{gh\to i}} \tag{4.7}$$

which yields the following variable-to-factor SP message:

$$\nu_{i\to ij}(k_{i\to ij}) \tag{4.8}$$

$$\propto \sum_{k_{\partial i\setminus ij\to i}} \mathbb{I}\left(k_{i\to ij} = \text{bin}\left(\ln\left(\frac{s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=1)}{s_{i\to ij}(k_{\partial i\setminus ij\to i})(x_i=-1)}\right)\right)\right) s_{i\to ij}(k_{\partial i\setminus ij\to i})(\emptyset)\prod_{gh\in\partial i\setminus ij}\nu_{gh\to i}\ (k_{gh\to i}) \tag{4.9}$$

$$= \sum_{k_{\partial i\setminus ij\to i}} \mathbb{I}\left(k_{i\to ij} = \text{bin}\left(\sum_{gh\in\partial i\setminus ij} r_{k_{gh\to i}}\right)\right) \prod_{gh\in\partial i\setminus ij}(1-\sigma(k_{gh\to i}))\nu_{gh\to i}\ (k_{gh\to i}) \tag{4.10}$$

$$+ \sum_{k_{\partial i\setminus ij\to i}} \mathbb{I}\left(k_{i\to ij} = \text{bin}\left(\sum_{gh\in\partial i\setminus ij} r_{k_{gh\to i}}\right)\right) \prod_{gh\in\partial i\setminus ij}\sigma(k_{gh\to i})\nu_{gh\to i}\ (k_{gh\to i}) \tag{4.11}$$

which yields:

$$\nu_{i\to ij} \propto \prod_{gh\in\partial i\setminus ij}^{*}(1-\sigma)\nu_{gh\to i} + \prod_{gh\in\partial i\setminus ij}^{*}\sigma\nu_{gh\to i} \tag{4.12}$$

To obtain the last line above, we note that indeed, for any set of functions $f_{gh\to i}$:

$$\sum_{k_{\partial i\setminus ij\to i}} \mathbb{I}\left(k_{i\to ij} = \text{bin}\left(\sum_{gh\in\partial i\setminus ij} r_{k_{gh\to i}}\right)\right) \prod_{gh\in\partial i\setminus ij} f_{gh\to i}\left(k_{gh\to i}\right) = \overset{*}{\prod_{gh\in\partial i\setminus ij}} f_{gh\to i} \quad (4.13)$$

Because the bin centers r_k are linearly spaced about a log-ratio of 0 the summation of any of these bins will also be linearly spaced about 0 with the same spacing between bin centers. In other words, the closest bin center found by the function bin() for the summation will always correspond exactly to a valid bin center value r_k. This means that $\mathbb{I}\left(k_{i\to ij} = \text{bin}\left(\sum_{gh\in\partial i\setminus ij} r_{k_{gh\to i}}\right)\right) = \mathbb{I}\left(k_{i\to ij} = \sum_{gh\in\partial i\setminus ij} k_{gh\to i}\right)$ which, when substituted into Equation 4.13, yields the definition of convolution.

4.2.6 Pairwise factor-to-variable SP message

The first step in passing a $D(K-1)+1$-length frequency domain SP message through a pairwise factor is taking its inverse frequency transform which is $\mathcal{O}(DK\log(DK))$ using FFT. The main procedure involves a one-to-one mapping of the bin indices of the incoming SP message (of length K) onto bin indices of the SP message leaving the factor, and therefore requires K steps. We then take the frequency transform of the result which also has complexity $\mathcal{O}(DK\log(DK))$ to get a frequency domain result of length $D(K-1)+1$. Combining these steps, the complexity of passing SP messages through all $\mathcal{O}(ND)$ pairwise factors is $\mathcal{O}(ND^2K\log(DK))$.

Algorithm 2 shows the resulting factor to variable subroutine. Here, the operation $2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\tanh\left(\frac{r_{k_{i\to ij}}}{2}\right)\right)$ is the BP factor to variable update for the Ising model.The function bin : $\Re \to \{1,\ldots,K\}$ finds the nearest bin index k for a log-ratio.

To show how we arrive at this update, we have, starting from Equation 2.35:

Algorithm 2 SP factor to variable subroutine

Input: incoming message $\nu_{i\to ij}$ to the factor ij
Output: outgoing messages $\nu_{ij\to j}$ from the factor ij
for $k \in \{1,\dots,K\}$ **do**
  $\nu_{ij\to j}(k) \leftarrow 0$ **for** $k' \in \{1,\dots,K\}$ **do**
    if $k = \mathrm{bin}\left(2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\tanh\left(\frac{r_{k'}}{2}\right)\right)\right)$ **then**
      $\nu_{ij\to j}(k) \leftarrow \nu_{ij\to j}(k) + \nu_{i\to ij}(k')$
    end
  end
end

$$s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=-1\right) = e^{J_{ij}}\hat{b}_{i\to ij}(x_i=-1) + e^{-J_{ij}}\hat{b}_{i\to ij}(x_i=1) \tag{4.14}$$

$$= e^{J_{ij}}(1-\sigma(k_{i\to ij})) + e^{-J_{ij}}\sigma(k_{i\to ij}) \tag{4.15}$$

$$s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=1\right) = e^{-J_{ij}}\hat{b}_{i\to ij}(x_i=-1) + e^{J_{ij}}\hat{b}_{i\to ij}(x_i=1) \tag{4.16}$$

$$= e^{-J_{ij}}(1-\sigma(k_{i\to ij})) + e^{J_{ij}}\sigma(k_{i\to ij}) \tag{4.17}$$

$$s_{ij\to j}\left(k_{i\to ij}\right)(\emptyset) = s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=-1\right) + s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=1\right) = e^{-J_{ij}} + e^{J_{ij}} \tag{4.18}$$

$$\ln\left(\frac{s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=1\right)}{s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=-1\right)}\right)=\ln\left(\frac{e^{-J_{ij}}(1-\sigma(k_{i\to ij}))+e^{J_{ij}}\sigma(k_{i\to ij})}{e^{J_{ij}}(1-\sigma(k_{i\to ij}))+e^{-J_{ij}}\sigma(k_{i\to ij})}\right) \tag{4.19}$$

$$=2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\left(\frac{2-1/\sigma(k_{i\to ij})}{1/\sigma(k_{i\to ij})}\right)\right) \tag{4.20}$$

$$=2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\left(\frac{2-(1+e^{-r_{k_{i\to ij}}})}{1+e^{-r_{k_{i\to ij}}}}\right)\right) \tag{4.21}$$

$$=2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\tanh\left(r_{k_{i\to ij}}/2\right)\right) \tag{4.22}$$

which yields the following pairwise factor-to-variables SP message:

$$\nu_{ij\to j}(k_{ij\to j}) \tag{4.23}$$

$$\propto\sum_{k_{i\to ij}}\mathbb{I}\left(k_{ij\to j}=\mathrm{bin}\left(\ln\left(\frac{s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=1\right)}{s_{ij\to j}\left(k_{i\to ij}\right)\left(x_j=-1\right)}\right)\right)\right)s_{ij\to j}\left(k_{i\to ij}\right)(\emptyset)\nu_{i\to ij}(k_{i\to ij})$$

$$\tag{4.24}$$

$$=\sum_{k_{i\to ij}}\mathbb{I}\left(k_{ij\to j}=\mathrm{bin}\left(2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\tanh\left(r_{k_{i\to ij}}/2\right)\right)\right)\right)(e^{-J_{ij}}+e^{J_{ij}})\nu_{i\to ij}(k_{i\to ij})$$

$$\tag{4.25}$$

$$\propto\sum_{k_{i\to ij}}\mathbb{I}\left(k_{ij\to j}=\mathrm{bin}\left(2\tanh^{-1}\left(\tanh\left(J_{ij}\right)\tanh\left(r_{k_{i\to ij}}/2\right)\right)\right)\right)\nu_{i\to ij}(k_{i\to ij}) \tag{4.26}$$

4.2.7 General complexity

Adding the complexity of sending all SP messages from the variables to the complexity of then passing them through all factors, we see that the overall algorithm complexity is $\mathcal{O}(ND^2K\log(DK))$ – *i.e.*, polynomial in D, N, and K. This shows that SP is scalable to large instances in the number of variables, bins, or the maximum degree of a variable.

Algorithm 3 summarizes this general algorithm. Note that in the initialization step

of this algorithm, $2\theta_i$ is the log-ratio of $f_i(x_i) = e^{\theta_i x_i}$, representing the local factor to variable BP update. Since this quantity remains static throughout the iterations of LBP, so will its corresponding SP message during SP iterations.

Algorithm 3 Approximate SP for the Ising Model

Input: The Ising model $\{J_{i,j}\}_{i,j}$, $\{h_i\}_i$; fidelity K
Output: Approximate SP marginals over BP marginals ν_i
Initialize $\nu_{ij\to j} \quad \forall ij$ **for** $i \in \{1,\dots,N\}$ **do**
  for $k \in \{1,\dots,K\}$ **do**
    Fix $\nu_{ii\to i}(k) \leftarrow \mathbb{I}\big(k = \text{bin}(2\theta_i)\big)$
  end
  Fix $\nu^0_{ii\to i} \leftarrow \mathcal{F}\left((1-\sigma)\nu_{ii\to i}\right)$
  Fix $\nu^1_{ii\to i} \leftarrow \mathcal{F}\left(\sigma\nu_{ii\to i}\right)$
end
while *not converged* **do**
  for $i \in \{1,\dots,N\}$ **do**
    Run SP variable-to-factor algorithm 1
  end
  for $i, j \neq i$ *with* $J_{ij} \neq 0$ **do**
    Run SP factor-to-variable algorithm 2
  end
end
for $i \in \{1,\dots,N\}$ **do**
  $\nu_i \leftarrow \mathcal{F}^{-1}\left(\nu^0_i + \nu^1_i\right)$
end

We show how we obtain this local factor message starting from Equation 2.35 we have:

$$s_{ii\to i}\left(\right)\left(x_i = -1\right) = e^{-\theta_i}, \tag{4.27}$$

$$s_{ii\to i}\left(\right)\left(x_i = 1\right) = e^{\theta_i}, \tag{4.28}$$

$$s_{ii\to i}\left(\right)\left(\emptyset\right) = s_{ii\to i}\left(\right)\left(x_i = -1\right) + s_{ii\to i}\left(\right)\left(x_i = 1\right) = e^{-\theta_i} + e^{\theta_i}, \tag{4.29}$$

and

$$\ln\left(\frac{s_{ii\to i}\left(\right)\left(x_i=1\right)}{s_{ii\to i}\left(\right)\left(x_i=-1\right)}\right)=\ln\left(\frac{e^{\theta_i}}{e^{-\theta_i}}\right)=2\theta_i, \tag{4.30}$$

which when applied to the generic factor-to-variable SP message yields the following local factor-to-variables SP message:

$$\nu_{ii\to i}(k_{ii\to i})\propto\mathbb{I}\left(k_{ii\to i}=\text{bin}\left(\ln\left(\frac{s_{ii\to j}\left(\right)\left(x_i=1\right)}{s_{ii\to i}\left(\right)\left(x_i=-1\right)}\right)\right)\right)s_{ii\to i}\left(\right)\left(\emptyset\right) \tag{4.31}$$

$$=\mathbb{I}\left(k_{ii\to i}=\text{bin}\left(2\theta_i\right)\right)\left(e^{-\theta_i}+e^{\theta_i}\right) \tag{4.32}$$

$$\propto\mathbb{I}\left(k_{ii\to i}=\text{bin}\left(2\theta_i\right)\right) \tag{4.33}$$

Lastly, we note that in Algorithm 3 the quantity ν_i is easy to compute since we continuously track ν_i^0 and ν_i^1 stemming from Algorithm 1 by having the product of their frequency transforms.

4.3 Tracking many fixed points

To show the capabilities of SP, we test our algorithm on the attractive homogeneous Ising model where local fields θ_i are set to zero for all variables and the coupling strengths J_{ij} are all set to the same value $J>0$.

When running LBP on this model, a phase transition exists as J increases [55]. Before this phase transition, LBP has a single fixed point. After the phase transition, it has two and picks the one closest to the BP messages at initialization.

For a D-regular Ising model with $N=1000$ variables and degree $D=4$, the phase transition occurs roughly at $J=0.347$. We run LBP and our approximate SP algorithm on this model for this critical value of J as well as values below and above

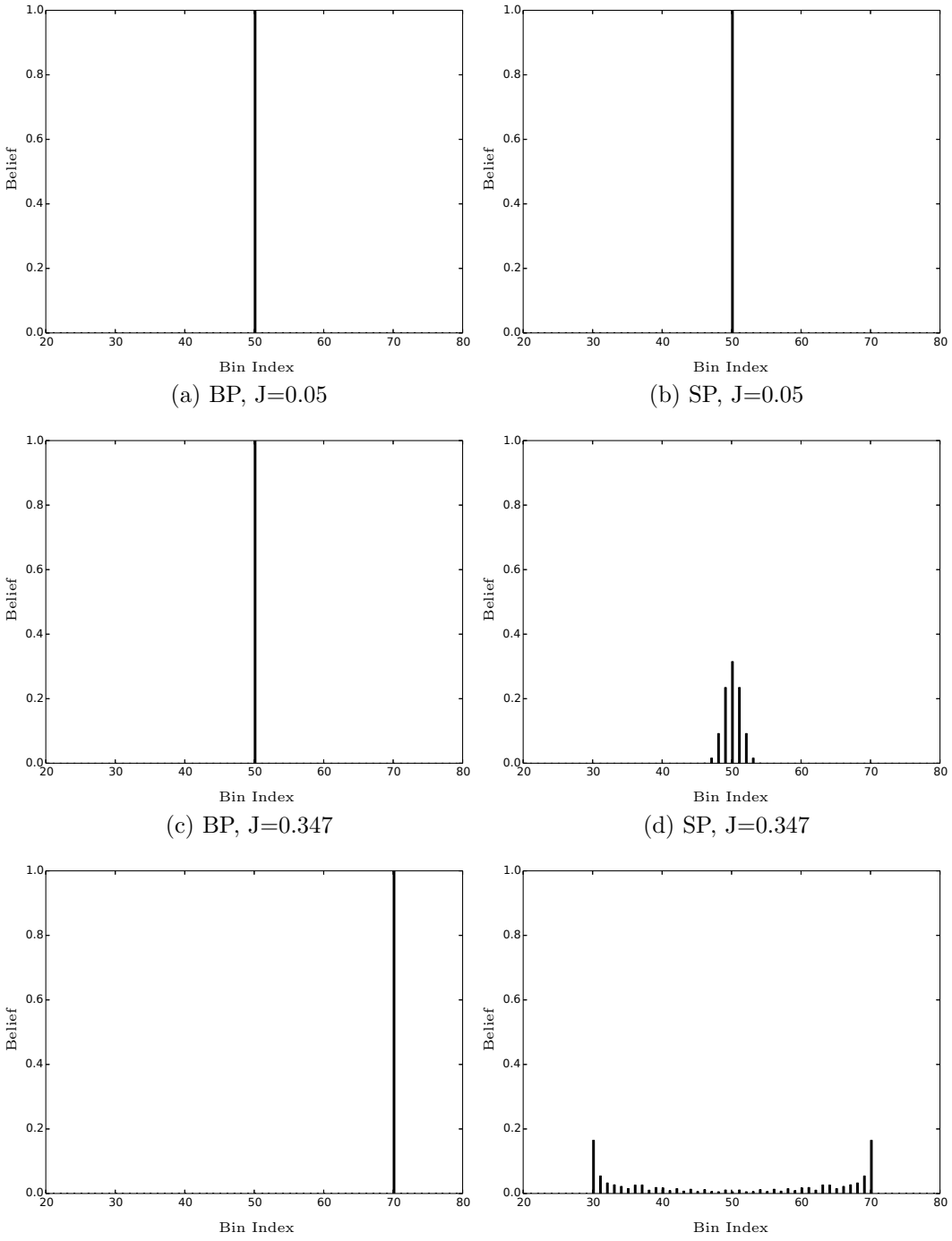
it. For SP we use 101 bins where the middle bin represents a log-ratio of zero. Here, leftmost bin 1 has a log-ratio of 0.0098 and rightmost bin 101 has a log-ratio of 0.9902. We plot one of the SP and BP messages for each value of J. Each BP message is expressed in the SP message space by taking its value and binning it into one of the 101 SP bins. From Figure 4.3, we see that before the phase transition the SP message also consists of a delta function, only one fixed point exists, and *SP emulates BP*. Here, the value on which the delta is placed corresponds to the fixed-point message produced by BP.

SP begins to provide useful information when its message hedges its bets on more than one value, as can be seen at the phase transition. This indicates that more than one BP fixed point exists and that SP is accounting for them all. This is clearer after the transition, where the SP message captures both fixed points. Note that the non-zero bins between the two fixed points is because the fixed points overlap to a certain degree. The assumption with SP is that the fixed points are disjoint which could only be valid if the number of variables tends to infinity. This overlap (*i.e.*, the residue) diminishes as we increase the size.

4.4 Experiments

We can use SP marginals, defined over BP marginals, to obtain an average marginal at each variable i as $\mathbb{E}[b_i(x_i = 1)] = \sum_{k=1}^{K} \sigma(k)\nu_i(k)$. This average is implicitly weighted by the sum of approximate partition function of BP fixed points with a particular marginal $\sigma(k)$.

We compare these averaged SP marginals to that of BP and Gibbs sampling for many graph types. For each type, we test on three types of pairwise couplings: attractive, mixed, and repulsive.



(a) BP, J=0.05 (b) SP, J=0.05

(c) BP, J=0.347 (d) SP, J=0.347

(e) BP, J=0.95 (f) SP, J=0.95

Figure 4.3: SP and BP messages for attractive homogeneous Ising model (N = 1000), before (1st row), during (middle row) and after (bottom row) phase transition.

All couplings are sampled from a uniform distribution. Attractive couplings are sampled in the range $[0, \beta]$, mixed from the range $[-\beta, \beta]$, and repulsive from the range $[-\beta, 0]$ where β ranges from 0 to 2. Local fields θ_i are always sampled uniformly in the range $[-0.05, 0.05]$. For each graph and coupling type, we run three versions of SP: SP with 11 bins, 101 bins, and 1001 bins. BP messages are initialized to be uniform while SP messages are initialized randomly. All message passing algorithms do parallel message updates and run for a maximum 1000 iterations or until convergence, whichever comes first. Gibbs sampling is run for 100,000 iterations where a sample is recorded after each 100 iterations. We repeat the experiments for 10 trials, each on a new instance, at each β value.

We limit the graphs' sizes so that exact marginals can be obtained via the junction tree algorithm [51]. However, as per our earlier analysis, this inference scheme can easily scale up, possibly to graphs up to millions of edges. At each value of β, each **column** of Figures 4.4 to 4.6 shows: *1st)* the mean absolute error of the marginals for all methods; *2nd)* the mean SP marginal entropy for the SP algorithms; *3rd)* the mean number of iterations for BP and SP. libDAI [58] is used for BP, Gibbs sampling, and the junction tree method.

Each **row** in Figures 4.4 to 4.6 is a graph type: *1st)* 10 variable, fully connected; *2nd)* 40 variable, bipartite with 20 variables blocks; *3rd)* 100 variable 3-regular; *4th)* 100 variable, 10x10 grid with periodic boundary; *5th)* 1000 variable, 10x100 grid without periodic boundary.

Analysis. In all experiments, SP outperforms BP at non-trivial temperatures (see first column of the figures). SP achieves this accuracy, by converging after only a few iterations (see last column of the figures). In fact, SP's average marginal error curves resemble that of Gibbs sampling using a large number of (*i.e.*, 10^5) iterations. While Gibbs sampling does better and has a per iteration complexity similar to that

of BP which is lower than SP, we need to run Gibbs for a significantly longer number of iterations than SP, as outlined in the experimental setup of this section, and from looking at the number of iterations that SP required in our results. As the number of modes/fixed points increases, this problem would only get worse. The average SP marginal entropy (see 2nd column of the figures) starts at zero at trivial temperatures, suggesting that BP indeed has a single fixed point, and increases for lower temperature (larger β) values.

For the fully connected and bipartite graphs (first two rows of the figure) in the mixed coupling scenario, the entropy points at a phase transition where it rises and then drops. After it drops, at β=2, the SP marginals show two distinct modes per marginal. Here, in contrast to the attractive homogeneous instance of section 4.3 with two fixed points, we hypothesize that these two modes identify a large number of BP fixed points, where the BP joint form is effectively choosing one of the two modes at each variable, producing incorrect marginals. While BP still converges to one of its large number of modes for these instances (see final column), for the 3-regular graph and both grid graphs (rows 3-5), BP has trouble converging at all. For these graphs (*i.e.*, the 3-regular and both grid graphs), SP beliefs are spread across many SP bins and the entropy climbs as β increases.

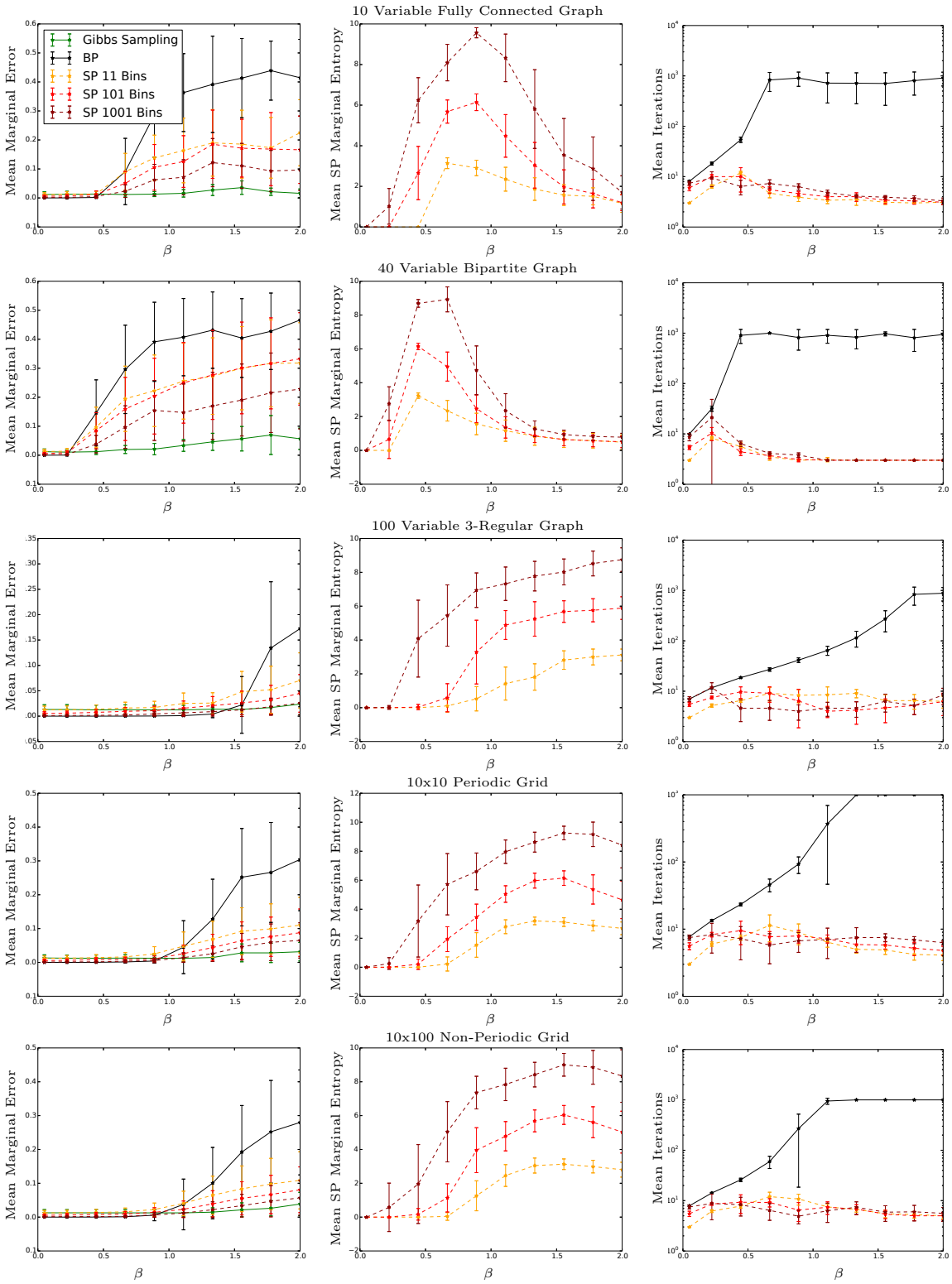


Figure 4.4: Results for mixed coupling. Green: Gibbs sampling, black: BP, dashed orange: SP 11 bins, dashed red: SP 101 bins, and dashed dark red: SP 1001 bins.

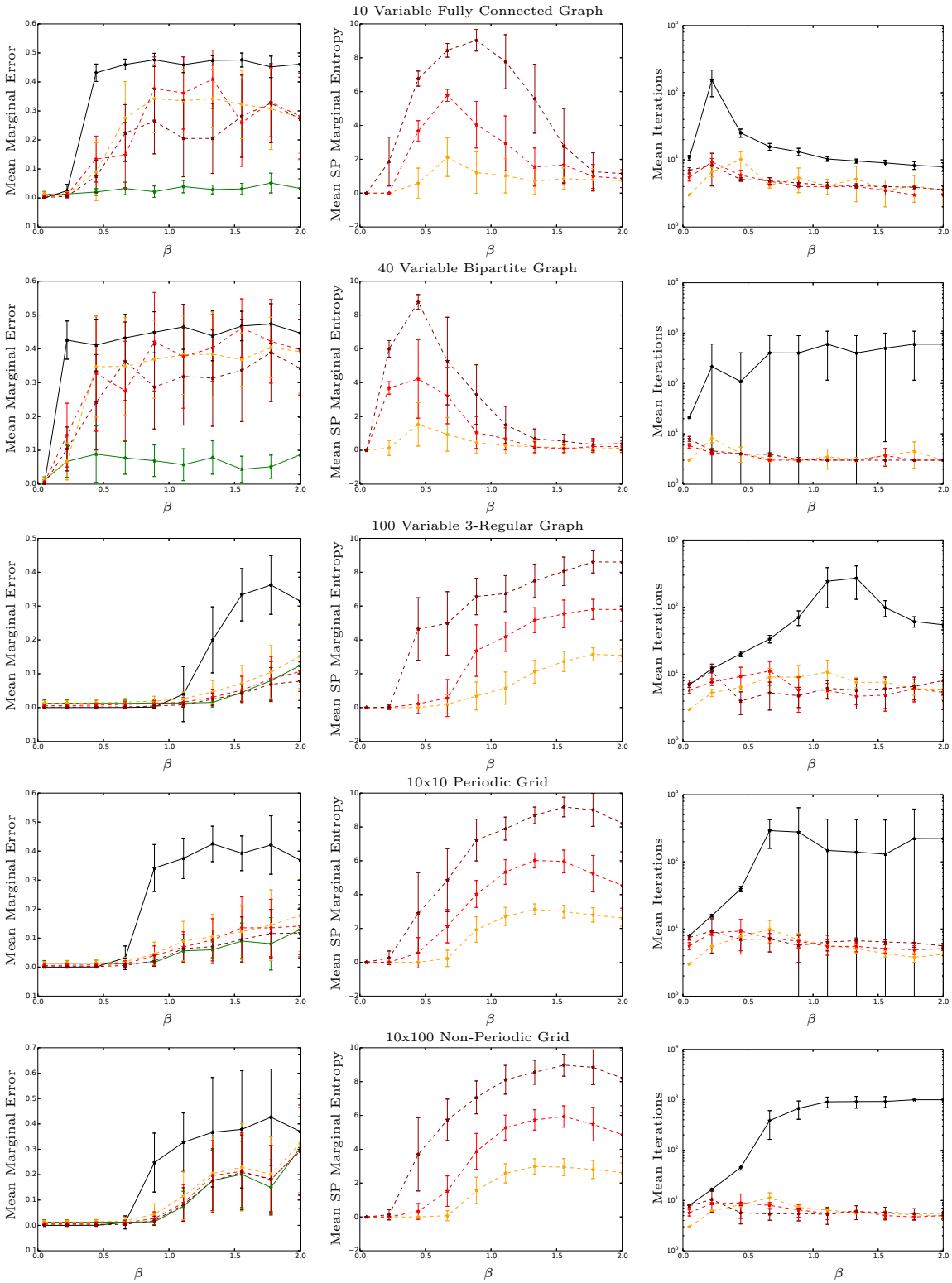


Figure 4.5: Results for attractive coupling. Green: Gibbs sampling, black: BP, dashed orange: SP 11 bins, dashed red: SP 101 bins, and dashed dark red: SP 1001 bins.

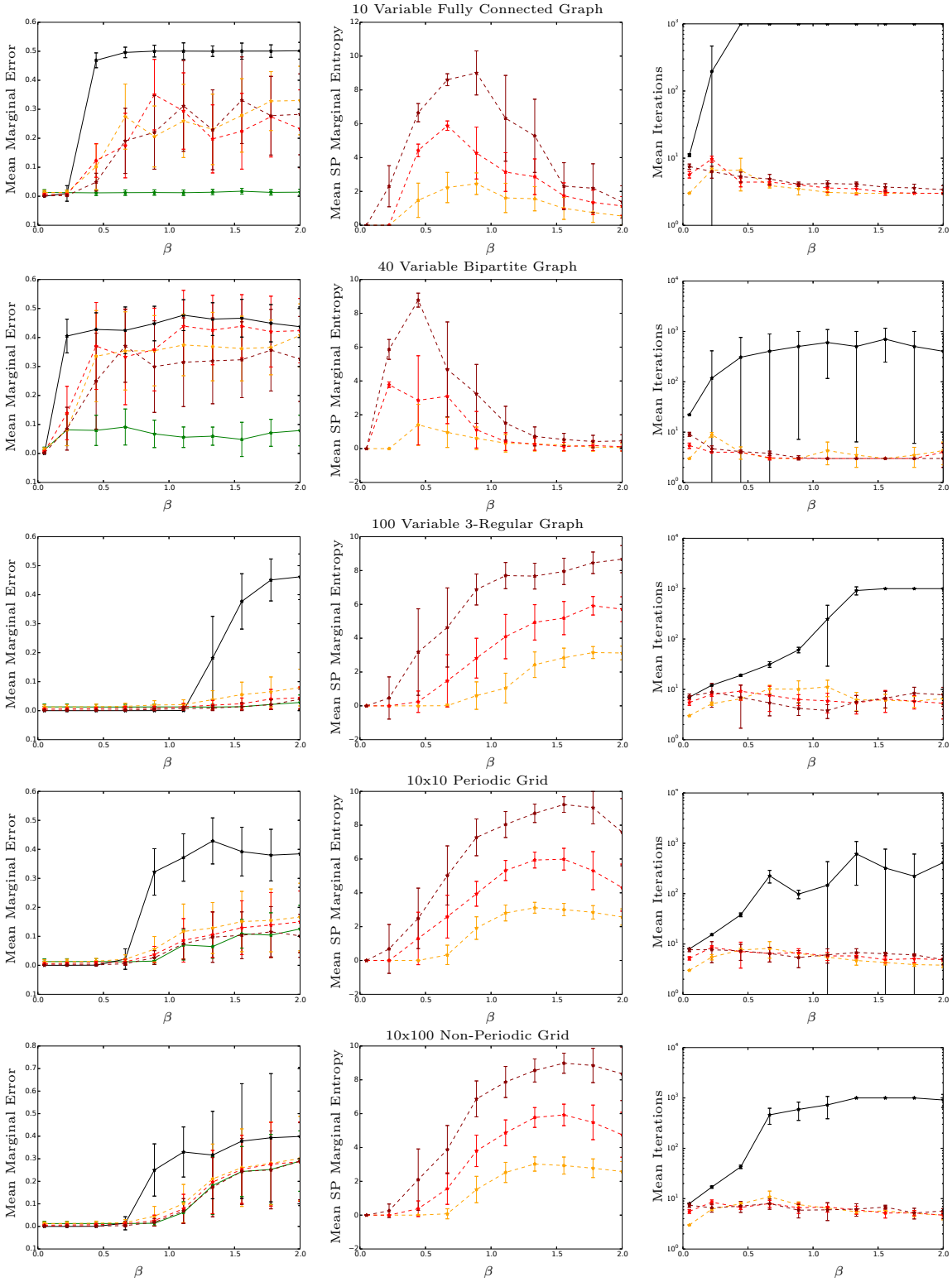


Figure 4.6: Results for repulsive coupling. Green: Gibbs sampling, black: BP, dashed orange: SP 11 bins, dashed red: SP 101 bins, and dashed dark red: SP 1001 bins.

Chapter 5

Conclusion and future work

This thesis focused on the distributive law of mathematics and fundamental ideas in the world of constraint satisfaction problems to make contributions in the areas of min-max inference and sum-product inference on multimodal distributions.

Within the context of min-max inference, we introduced the problem of min-max inference in factor graphs and provided two practical procedures for solving such problems. First, by leveraging the success of BP with alternate operators and formulating a min-max version of BP, namely MMP. Second, by casting the min-max problem as a CSP and using BP's success at solving CSPs combined with a bisection search to find a satisfiable min-max bound. We do a head-to-head comparison of both methods on random instances while also detailing various decimation procedures for the MMP propagation method. Next, factor graph representations for several important combinatorial problems such as min-max clustering, K-clustering, the bottleneck TSP and K-packing are represented and solved by message passing. Since the CSP reduction performs a little better than MMP on the random instances, we choose to use this method to solve the above factor graph representations. In doing so, a message passing approach to several NP-hard decision problems including the clique-cover,

max-clique, dominating-set, set-cover and Hamiltonian path problem are provided. For each problem we also analyzed the complexity of message passing and established its practicality using several relevant experiments. Next, we showed that for some types of high order factors, only MMP messages can be passed in polynomial time. After detailing those procedures, we show how we can use these factors can be combined to obtain a factor graph representing the makespan minimization problem. Using our efficient MMP message derivations for the factors, we show results for the application in question.

Within the context of multimodal sum-product inference, we introduced the first SP approximation scheme for inference beyond CSPs. Our scheme uses FFTs to scale gracefully with the number of variables and edges in the model. Our extensive experiments show that it converges to accurate posterior marginals within few iterations. We believe this opens the door for its application and extension in a variety of settings.

We now propose some directions for future work related to each of these topics.

5.1 Min-max inference

It should be noted that throughout the thesis, we have not suggested ways to perform min-max inference in clustered solution space. We already mentioned that in clustering phases, the set of fixed points can become disjoint and multimodal. Since min-max propagation and the CSP solvers presented in chapter 3 are all variants BP, only one of these fixed points is tracked. While we highlighted - in a sum-product context - that SP is a powerful algorithm for such regimes, we now propose, for future studies, two ways in which it could be integrated into min-max inference on factors graphs.

5.1.1 Survey propagation for MMP

The first direction would be to derive an efficient version of SP over min-max propagation. That is, we suggest deriving SP but this time over a BP algorithm which uses min and max operators.

We note that in this case the SP operators are still sum and product. In other words, SP ends up counting all possible fixed points reached by min-max propagation. The question is whether this can be achieved efficiently. If so, the final marginal at each variable would be a distribution over all possible min-max messages with possibly multiple modes. The probabilistic support on each message would represent the number of min-max propagation fixed points in which the message is involved. One could then decimate onto the best min-max solution by setting the variables to the lowest min-max message values with non-zero support.

5.1.2 Survey propagation over min-max as a CSP

The second method by which SP could help is to run the efficient version over sum-product BP derived in chapter 4, as the CSP solver to the min-max satisfiability problems in chapter 3. In that chapter, we already established a link between min-max problems and CSPs by casting one as the other. Furthermore, in chapter 2 we mentioned that when the CSP space becomes disjoint, SP - in its original form over with OR-AND operators yields better results than BP. Thus, this seems like a sensible algorithmic choice to solve any min-max instance cast as a CSP.

5.1.3 Relationship between MMP and the CSP formulation

In chapter 3 we did a head on performance comparison between the CSP min-max formulation and min-max propagation. One aspect which is of interest is how the

two methods are related beyond empirical comparison. In other words, as a problem instance becomes difficult what it is that min-max propagation sacrifices in its approximate messages passing, and how does that compare to what the CSP formulation sacrifices. We also showed that for certain high order factors such as the one in the makespan application, that the CSP formulation of the factor type does not allow for efficient message passing while there is no issue for min-max propagation. This suggest that min-max propagation "escapes" some sort of responsibility when passing messages through the factors. Does this suggest that min-max propagation places higher reliance on the distributive nature of message passing vs the CSP reduction?

5.2 Survey propagation

5.2.1 Free energy approximation

In chapter 4, we showed that running message passing on the Bethe Free Energy yields SP. One interesting question which remains is whether there exists a free energy approximation for SP itself and if so, what are its properties. In [55], the authors show that for the CSP scenario the survey propagation messages can indeed be described by a function. However, a generalization of this function to a free energy for the generalized inference scenario, as studied in chapter 4, is for future work.

5.2.2 Counting survey propagation

Central to the application of inference in machine learning is its role in maximum likelihood learning. That is, the marginals obtained via inference in chapter 4 are often used in machine learning algorithms to update parameter estimates. As we mentioned in chapter 2, loopy BP and its variants are a popular method to obtain

these marginals but when they fail, one often resorts to sampling algorithms combined with advantageous model structures for fast and accurate inference. A prime example of this is the Restricted Boltzmann Machine (RBM) and its contrastive divergence algorithm which uses block Gibbs sampling [40]. An RBM is a special case of the Ising model presented in chapter 4. In it, variables are separated into two groups and each variable from one group has pairwise connections to all variables in the other group but not to any variables in its own group. This bipartite structure for the variables creates favorable conditional independence where the variables in one group are all independent of each other if we know the values of the variables in the other group. This allows for efficient inference via block Gibbs sampling where we can alternate between using samples for variables in one group to generate samples for variables in the other group, and repeat until we collect enough samples to compute marginals.

Since SP is able to give better marginals than loopy BP, we hope to be able to use it rather than block Gibbs sampling to obtain marginals for RBMs. Furthermore, since SP does not require the model to have any particular structure, unlike block Gibbs sampling, we would be able to use it on models that are not necessarily structure specific but perhaps better representatives of the machine learning problem at hand.

The only restriction which our implementation paces is the SP message length in the frequency domain. As we showed in chapter 4 the message length is directly related to the cardinality of a variable in the graph to avoid aliasing when convolving the incoming messages. As such, it is in our interest to minimize the cardinality of each variable for computational/storage efficiency. For large scale problems, this suggests using patch based models and convolutional architectures. These allow efficient scaling by having low cardinality variables, copied multiple times, to span the full size of the problem. Furthermore, for such models there exists an efficient version of BP called counting BP. Counting BP takes advantage of such models by compressing

these multiple copies and running BP once with the appropriate compression ratios. Due to its similar message passing nature, one could easily imagine SP version of the same procedure.

5.2.3 Ensembles

Another direction is in modeling and marginalization for ensembles. For example, if we wish to de-noise over the set of all possible corruptions of an image [10] by a particular noise distribution (where the noise distribution may be different for each pixel), we could do so with SP by placing a prior over each local factor representing the noise distribution. This idea is closely related to an algorithm called *density evolution* used for error-correcting codes [70]. Density evolution operates by leveraging the symmetry in the error correcting codes to show that the density evolution variable-to-factor message updates are the same for all variables and, likewise, for all factor-to-variable messages, providing a closed form for the ensemble. SP can be thought of as a generalization of density evolution where the Bethe partition function of each fixed point is tracked. This is important if the goal is to infer marginals. Another method along the same lines is density propagation of [23] that uses message passing with convolution to estimate the number of configurations $\underline{x}$ with a probability $p(\underline{x})$.

5.2.4 Maximum a posteriori inference

Lastly, we are studying extensions of our scheme to perform more accurate max-sum inference. [16] proposes a method that uses SP for Maximum A Posteriori (MAP) inference. Their method relies on transferring the problem to a CSP and applying a parameterized version of SP [53] to perform inference by proxy. However, direct use of SP for MAP, similar to our scheme is feasible, where the convolution is replaced

with max-convolution, the zero temperature Bethe partition function represents the value of MAP assignment, and max-product SP tries to find the BP fixed point with the maximum approximate MAP value.

Appendices

Appendix A

Additional survey propagation equations

A.1 Variable and factor beliefs to BP messages

To establish the results in the following sections, we first establish the intermediate result that, for $\hat{b} \in \mathcal{B}$ (where $\mathcal{B}$ is the set of messages that satisfy BP equations):

$$\frac{s_i(\hat{b}_{\partial i \to i})(\emptyset)}{s_{i \leftrightarrow I}(\hat{b}_{I \to i}, \hat{b}_{i \to I})(\emptyset)} = s_{i \to I}(\hat{b}_{\partial i \setminus I \to i})(\emptyset) \tag{A.1}$$

and

$$\frac{s_I(\hat{b}_{\partial I \to I})(\emptyset)}{s_{i \leftrightarrow I}(\hat{b}_{I \to i}, \hat{b}_{i \to I})(\emptyset)} = s_{I \to i}(\hat{b}_{\partial I \setminus i \to I})(\emptyset) \tag{A.2}$$

To do so, we note that from the definitions of $s_I(\hat{b}_{\partial I \to I})$ in Equation 2.29 and $s_{I \to i}(\hat{b}_{\partial I \setminus i \to I})$ in Equation 2.35 that:

$$s_I(\hat{b}_{\partial I\to I})(x_i) = \hat{b}_{i\to I}(x_i)\sum_{x_{\partial I\setminus i}} f_I(x_I)\prod_{j\in\partial I\setminus i}\hat{b}_{j\to I}(x_j) \tag{A.3}$$

$$s_I(\hat{b}_{\partial I\to I})(x_i) = \hat{b}_{i\to I}(x_i)\left(\frac{\sum_{x_{\partial I\setminus i}} f_I(x_I)\prod_{j\in\partial I\setminus i}\hat{b}_{j\to I}(x_j)}{\sum_{x_{\partial I}} f_I(x_I)\prod_{j\in\partial I\setminus i}\hat{b}_{j\to I}(x_j)}\right)\left(\sum_{x_{\partial I}} f_I(x_I)\prod_{j\in\partial I\setminus i}\hat{b}_{j\to I}(x_j)\right) \tag{A.4}$$

$$s_I(\hat{b}_{\partial I\to I})(x_i) = \hat{b}_{i\to I}(x_i)\hat{b}_{I\to i}(x_i)s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(\emptyset) \tag{A.5}$$

$$\sum_{x_i} s_I(\hat{b}_{\partial I\to I})(x_i) = s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(\emptyset)\sum_{x_i}\hat{b}_{i\to I}(x_i)\hat{b}_{I\to i}(x_i) \tag{A.6}$$

$$s_I(\hat{b}_{\partial I\to I})(\emptyset) = s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(\emptyset)s_{i\leftrightarrow I}(\hat{b}_{I\to i},\hat{b}_{i\to I})(\emptyset) \tag{A.7}$$

and

$$s_i(\hat{b}_{\partial i\to i})(x_i) = \hat{b}_{I\to i}(x_i)\prod_{J\in\partial i\setminus I}\hat{b}_{J\to i}(x_i) \tag{A.8}$$

$$s_i(\hat{b}_{\partial i\to i})(x_i) = \hat{b}_{I\to i}(x_i)\left(\frac{\prod_{J\in\partial i\setminus I}\hat{b}_{J\to i}(x_i)}{\sum_{x_i}\prod_{J\in\partial i\setminus I}\hat{b}_{J\to i}(x_i)}\right)\left(\sum_{x_i}\prod_{J\in\partial i\setminus I}\hat{b}_{J\to i}(x_i)\right) \tag{A.9}$$

$$s_i(\hat{b}_{\partial i\to i})(x_i) = \hat{b}_{I\to i}(x_i)\hat{b}_{i\to I}(x_i)s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(\emptyset) \tag{A.10}$$

$$\sum_{x_i} s_i(\hat{b}_{\partial i\to i})(x_i) = s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(\emptyset)\sum_{x_i}\hat{b}_{I\to i}(x_i)\hat{b}_{i\to I}(x_i) \tag{A.11}$$

$$s_i(\hat{b}_{\partial i\to i})(\emptyset) = s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(\emptyset)s_{i\leftrightarrow I}(\hat{b}_{I\to i},\hat{b}_{i\to I})(\emptyset) \tag{A.12}$$

where $b_{i\leftrightarrow I}$ is defined in Equation 2.30 of chapter 4.

A.2 Decomposition of the mass

The key to understanding how Equation 2.31 is obtained is to note that any BP message b can be viewed as a normalized version of itself multiplied by its normalization constant. For a variable-to-factor this means $b_{i\to I}(x_i) = \hat{b}_{i\to I}(x_i)b_{i\to I}(\emptyset)$ and likewise for a factor-to-variable message $b_{I\to i}(x_i) = \hat{b}_{I\to i}(x_i)b_{I\to i}(\emptyset)$.

With these observations we can rewrite Equations 2.6 and 2.7 as:

$$b_{i\to I}(x_i) = \prod_{J\in\partial i\setminus I} b_{J\to i}(\emptyset)\hat{b}_{J\to i}(x_i) \tag{A.13}$$

$$b_{I\to i}(x_i) = \sum_{x_{\partial I\setminus i}} f_I(x_I) \prod_{j\in\partial I\setminus i} b_{j\to I}(\emptyset)\hat{b}_{j\to I}(x_j) \tag{A.14}$$

Next if we sum over both sides of these equations we obtain:

$$\sum_{x_i} b_{i\to I}(x_i) = \prod_{J\in\partial i\setminus I} b_{J\to i}(\emptyset) \sum_{x_i} \prod_{J\in\partial i\setminus I} \hat{b}_{J\to i}(x_i) \tag{A.15}$$

$$b_{i\to I}(\emptyset) = s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(\emptyset) \prod_{J\in\partial i\setminus I} b_{J\to i}(\emptyset) \tag{A.16}$$

$$\sum_{x_i} b_{I\to i}(x_i) = \prod_{j\in\partial I\setminus i} b_{j\to I}(\emptyset) \sum_{x_I} f_I(x_I) \prod_{j\in\partial I\setminus i} \hat{b}_{j\to I}(x_j) \tag{A.17}$$

$$b_{I\to i}(\emptyset) = s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(\emptyset) \prod_{j\in\partial I\setminus i} b_{j\to I}(\emptyset) \tag{A.18}$$

The results in Equations A.16 and A.18 can be seen as recursive method where the first equation gets substituted into the second and vice-versa. If we have a factor

graph that is a tree we can define an endpoint for this recursion at a root node r. Knowing that if this root node had a single factor:

$$G(\hat{b})(\emptyset) = \sum_{x_r} b_r(x_r) = \sum_{x_r} \prod_{I \in \partial r} b_{I \to r}(x_r) = b_{I \to r}(\emptyset) \tag{A.19}$$

we can write out the recursion mentioned above by defining $\uparrow i$ to restrict the ∂i to the factor that is located higher than variable i in the tree (*i.e.*, closer to root r) and $\uparrow I$ to be the variable that is closer to the root than I, yielding

$$G(\hat{b})(\emptyset) = \prod_{i, I = \uparrow i} s_{i \to I}(\hat{b}_{\partial i \setminus I \to i})(\emptyset) \prod_{I, i = \uparrow I} s_{I \to i}(\hat{b}_{\partial I \setminus i \to I})(\emptyset) \tag{A.20}$$

Knowing from the result of the previous section that $s_{i \to I}(\hat{b}_{\partial i \setminus I \to i})(\emptyset) = \frac{s_i(\hat{b}_{\partial i \to i})(\emptyset)}{s_{i \leftrightarrow I}(\hat{b}_{I \to i}, \hat{b}_{i \to I})(\emptyset)}$ and $s_{I \to i}(\hat{b}_{\partial I \setminus i \to I})(\emptyset) = \frac{s_I(\hat{b}_{\partial I \to I})(\emptyset)}{s_{i \leftrightarrow I}(\hat{b}_{I \to i}, \hat{b}_{i \to I})(\emptyset)}$, making the appropriate substitutions in the above equation yields

$$G(\hat{b})(\emptyset) = \prod_I s_I(\hat{b}_{\partial I \to I})(\emptyset) \prod_i s_i(\hat{b}_{\partial i \to i})(\emptyset) \left(\prod_{i, I \in \partial i} s_{i \leftrightarrow I}(\hat{b}_{i \to I}, \hat{b}_{I \to i})(\emptyset) \right)^{-1} \tag{A.21}$$

A.3 Full SP messages

Figure 2.8 in chapter 4 shows a BP factor graph and its equivalent SP factor graph. In this SP factor graph, each message from the BP factor graph is represented by a variable and three kinds of factors as we have highlighted in chapter 4. As we also mentioned in that chapter, SP turns out to be BP running on this new SP factor graph. As always, messages are exchanged between circular (variable) and square (factor) nodes. But in this case, we can reduce the messages down to only two kinds of messages, each only travelling from one kind of factor node to another

kind of factor node. To do so, the key lies in noting that the third kind of factor $s_{i\leftrightarrow I}(\hat{b}_{i\to I}, \hat{b}_{I\to i})(\emptyset)^{-1}$ always only connects two variables $\hat{b}_{i\to I}$ and $\hat{b}_{I\to i}$. Thus, the messages can be simplified to

$$\nu_{i\to I}(\hat{b}_{i\to I}, \hat{b}_{I\to i}) \propto \sum_{\setminus \hat{b}_{i\to I}, \hat{b}_{I\to i}} \left(\frac{s_i(\hat{b}_{\partial i\to i})(\emptyset)}{s_{i\leftrightarrow I}(\hat{b}_{i\to I}, \hat{b}_{I\to i})(\emptyset)} \prod_{J\in\partial i\setminus I} \nu_{J\to i}(\hat{b}_{i\to J}, \hat{b}_{J\to i}) \right) \tag{A.22}$$

$$\nu_{I\to i}(\hat{b}_{i\to I}, \hat{b}_{I\to i}) \propto \sum_{\setminus \hat{b}_{i\to I}, \hat{b}_{I\to i}} \left(\frac{s_I(\hat{b}_{\partial I\to I})(\emptyset)}{s_{i\leftrightarrow I}(\hat{b}_{i\to I}, \hat{b}_{I\to i})(\emptyset)} \prod_{j\in\partial I\setminus i} \nu_{j\to I}(\hat{b}_{j\to I}, \hat{b}_{I\to j}) \right) \tag{A.23}$$

where we always assume that the messages $\hat{b} \in \mathcal{B}$ are valid BP messages (*i.e.*, satisfy BP equations on the original BP factor graph). Using the result from the first section in this appendix we can simplify these equations to

$$\nu_{i\to I}(\hat{b}_{i\to I}) \propto \sum_{\hat{b}_{\partial i\setminus I\to i}} \left(\mathbb{I}\left(\hat{b}_{i\to I} = s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})\right) s_{i\to I}(\hat{b}_{\partial i\setminus I\to i})(\emptyset) \prod_{J\in\partial i\setminus I} \nu_{J\to i}\ (\hat{b}_{J\to i}) \right) \tag{A.24}$$

and

$$\nu_{I\to i}(\hat{b}_{I\to i}) \propto \sum_{\hat{b}_{\partial I\setminus i\to I}} \left(\mathbb{I}\left(\hat{b}_{I\to i} = s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})\right) s_{I\to i}(\hat{b}_{\partial I\setminus i\to I})(\emptyset) \prod_{j\in\partial I\setminus i} \nu_{j\to I}\ (\hat{b}_{j\to I}) \right) \tag{A.25}$$

Note that the identity functions in these results indeed enforce that the messages in the original BP factor graph are indeed valid BP messages (*i.e.*, respect BP updates).

Appendix B

Min-max CSP reductions proofs and analysis

B.1 Proofs

Proof 3.2.1 *(A) μ_y for $y \geq y^*$ is satisfiable:* It is enough to show that for any $y \geq y^*$, $\mu_y(x^*) > 0$. But since

$$\mu_y(x^*) \quad = \quad \frac{1}{Z_y} \prod_I \mathbb{I}(f_{\mathbb{I}}(x^*_I) \leq y)$$

and $f_I(x^*_I) \leq y^* \leq y$, all the indicator functions on the rhs are evaluated to 1, showing that $\mu_y(x^*) > 0$.

(B) μ_y for $y < y^*$ is not satisfiable: Let us assume that for some $\underline{y} < y^*$, $\mu_{\underline{y}}$ is satisfiable. Let $\underline{x}$ denote a satisfying assignment –*i.e.*, $\mu_{\underline{y}}(\underline{x}) > 0$. Using the definition of μ_y-reduction this implies that $\mathbb{I}(f_I(\underline{x}_I) \leq \underline{y}) > 0$ for all $I \in \mathcal{F}$. However, this means that $\max_I f_I(\underline{x}_I) \leq \underline{y} < y^*$, which means y^* is not the min-max value.

Proof 3.3.1 Assume they have different min-max assignments[1] –*i.e.*, $x^* = \arg_x \min \max_I f_I(x_I)$, $x'^* = \arg_x \min \max_I f'_I(x_I)$ and $x^* \neq x'^*$. Let y^* and y'^* denote the corresponding min-max values.

Claim B.1.1

$$y^* > \max_I f_I(x'^*_I) \quad \Leftrightarrow \quad y'^* < \max_I f'_I(x^*_I)$$
$$y^* < \max_I f_I(x'^*_I) \quad \Leftrightarrow \quad y'^* > \max_I f'_I(x^*_I)$$

This simply follows from the condition of Lemma 3.3.1. But in each case above, one of the assignments y^* or y'^* is not an optimal min-max assignment as an alternative assignment has a lower maximum over all factors.

Proof 3.3.2 First note that since $g(x) = 2^z$ is a monotonically increasing function, the rank of elements in the range of $\{f'_I\}_I$ is the same as their rank in the range of $\{f_I\}_I$. Using Lemma 3.3.1 this means

$$\arg_x \min \max_I f'_I(x_I) \quad = \quad \arg_x \min \max_I f_I(x_I). \tag{B.1}$$

Since $2^z > \sum_{l=0}^{z-1} 2^l$, by definition of $\{f'_I\}$, we have

$$\max_{I \in \mathcal{F}} f'_I(x_I) > \sum_{I \in \mathcal{F} \setminus I^*} f'_I(x_I) \quad \text{where } \; I^* = \arg_I \max f'_I(x_I)$$

[1]Here for simplicity, we are assuming each instance has a single min-max assignment. In case of multiple assignments there is a one-to-one correspondence between them. Here, one starts with the assumption that there is an assignment x^* for the first factor-graph that is different from all min-max assignments in the second factor-graph.

It follows that for $x^1, x^2 \in \mathcal{X}$,

$$\max_I f'_I(x^1_I) < \max_I f'_I(x^2_I) \quad \Leftrightarrow \quad \sum_I f'_I(x^1_I) < \sum_I f'_I(x^2_I)$$

Therefore,

$$\arg_x \min \max_I f'_I(x_I) \quad = \quad \arg_x \min \sum_I f'_I(x_I).$$

This equality combined with Equation B.1 prove the statement of the theorem.

Proof 3.2.2 Recall the definition $Z_y \triangleq \sum_{\mathcal{X}} \prod_I \mathbb{I}(f_I(x_I) \leq y)$. For $y_1 < y_2$ we have:

$$\begin{aligned}
&f_I(x_I) \leq y_1 \quad \rightarrow \quad f_I(x_I) \leq y_2 && \Rightarrow \\
&\mathbb{I}(f_I(x_I) \leq y_1) \leq \mathbb{I}(f_I(x_I) \leq y_2) && \Rightarrow \\
&\sum_{\mathcal{X}} \prod_I \mathbb{I}(f_I(x_I) \leq y_1) \leq \sum_{\mathcal{X}} \prod_I \mathbb{I}(f_I(x_I) \leq y_2) && \Rightarrow \\
&Z_{y_1} \leq Z_{y_2}
\end{aligned}$$

Proof 3.5.1 First, note that μ_y defines a uniform distribution over its support as its unnormalized value is only zero or one.

(A) *Every K-clique-cover over $\mathcal{G}(D, y)$, defines K unique assignments x with $\mu_y(x) > 0$:* Let $x \in \mathcal{X}$ denote a K-clique-cover over $\mathcal{G}$, such that $x_i \in \{1, \ldots, K\}$ indicates the clique assignment of node $i \in \mathcal{V}$. Since all the permutations of cliques produce a unique assignment x, there are K assignments per K-clique-cover. For any clique-cover of $G(D, y)$, i and j are connected only if $D_{i,j} \leq y$. Therefore, two nodes can belong to the same clique $\mathcal{C}_k$ only if $D_{i,j} \leq y$. On the other hand, the μ_y-reduction

of the Potts factor for min-max clustering is

$$f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i \neq x_j) + \mathbb{I}(x_i = x_j \wedge D_{i,j} \leq y) \tag{B.2}$$

Here, either i and j belong to different cliques where $f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i \neq x_j) > 0$ or they belong to the same clique in the clique-cover x: $f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i = x_j \wedge D_{i,j} \leq y) > 0$. Therefore, for any such x, $\mu_y(x) > 0$.

(B) *Every x for which $\mu_y(x) > 0$, defines a unique K-clique-cover over $\mathcal{G}(D, y)$:* Let i and j belong to the same cluster *iff* $x_i = x_j$. $\mu_y(x) > 0$ implies all its factors as given by Equation B.2 have non-zero values. Therefore, for every pair of nodes i and j, either they reside in different clusters (*i.e.*, $x_i \neq x_j$), or $D_{i,j} \leq y$, which means they are connected in $\mathcal{G}(D, y)$. However, if all the nodes in the same cluster are connected in $\mathcal{G}(D, y)$, they also form a clique. Since $\forall i \;\; x_i \in \{1, \ldots, K\}$ can take at most K distinct values, every assignment x with $\mu_y(x) > 0$ is a K-clique-cover.

Proof 3.5.2 Here we prove Proposition 3.5.2 for the factor-graph of section 3.5.2. The proof for the binary variable model follows the same procedure. Since μ_y defines a uniform distribution over its support, it is enough to show that any clique of size K over $\mathcal{G}(-D, -y)$ defines a unique set of assignments all of which have nonzero probability ($\mu_y(x) > 0$) and any assignment x with $\mu_y(x) > 0$ defines a unique clique of size at least K on $\mathcal{G}(-D, -y)$. First, note that the basic difference between $\mathcal{G}(D, y)$ and $\mathcal{G}(-D, -y)$ is that in the former all nodes that are connected have a distance of at most y while in the later all nodes that have a distance of at least y are connected to each other. Consider the μ_y-reduction of the pairwise factors of the factor-graph defined in section 3.5.2:

$$f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i \neq x_j \wedge -D_{x_i,x_j} \leq y) \tag{B.3}$$

(A) Any clique of size K in $\mathcal{G}(-D,-y)$, defines K unique assignments, such that for any such assignment x, $\mu_y(x) > 0$: For a clique $\mathcal{C} = \{c_1, \ldots, c_K\}$ of size K, define $x_i = c_{\pi(i)}$, where $\pi : \{1, \ldots, K\} \to \{1, \ldots, K\}$ is a permutation of nodes in clique $\mathcal{C}$. Since there are K such permutations we may define as many assignments x. Now, consider one such assignment x. For every two nodes x_i and x_j, since they belong to the clique $\mathcal{C}$ over $\mathcal{G}(-D,-y)$, they are connected and $D_{x_i,x_j} \geq y$. This means that all the pairwise factors defined by Equation B.3 have non-zero values and therefore, $\mu_y(x) > 0$.

(B) Any assignment x with $\mu_y(x) > 0$ corresponds to a unique clique of size K in $\mathcal{G}(-D,-y)$: Let $\mathcal{C} = \{x_1, \ldots, x_K\}$. Since $\mu_y(x) > 0$, all pairwise factors defined by Equation B.3 are non-zero. Therefore, $\forall i, j \neq i \;\; D_{x,x_j} \geq y$, which means all x_i and x_j are connected in $\mathcal{G}(-D,-y)$, forming a clique of size K.

Proof 3.5.3 The μ_y-reduction of the factors of factor-graph of section 3.5.3 are:

A. *local factors* $\forall i \neq j \quad f^y_{\{i-j\}}(x_{i-j}) = \mathbb{I}(D_{i,j} \leq y \vee x_{i-j} = 0)$.

B. *uniqueness factors* For every i consider $I = \{i - j \mid 1 \leq j \leq N\}$, then $f^y_I(x_I) = \mathbb{I}(\sum_{i-j \in \partial I} x_{i-j} = 1)$.

C. *consistency factors* $\forall j, i \neq j$, $f^y(x_{\{j-j,i-j\}}) = \mathbb{I}(x_{j-j} = 0 \wedge x_{i-j} = 1)$.

D. *K-of-N factor* Let $\mathcal{K} = \{i - i \mid 1 \leq i \leq N\}$, then $f^y_{\mathcal{K}}(x_{\mathcal{K}}) = \mathbb{I}(\sum_{i-i \in \mathcal{K}} x_{i-i} = K)$.

(A) Any assignment x with $\mu_y(x) > 0$ defines a K-dominating set of $\mathcal{G}(D,y)$: Since $\mu_y(x) > 0$, all the factors above have a non-zero value for x. Let $\mathcal{D} = \{i \mid x_{i-i} = 1\}$.

Claim B.1.2 *$\mathcal{D}$ defines a K-dominating set of graph $\mathcal{G}$.*

The reason is that first (a) Since the K-of-N factor is nonzero $|D| = K$. (b) For any node $j \in \mathcal{V} \setminus \mathcal{D}$, the uniqueness factors and consistency factors, ensure that they are

associated with a node $i \in \mathcal{D}$ – $\exists i \in \mathcal{D} \mid x_{j,i} = 1$. (c) Local factors ensure that if $x_{j,i} = 1$ then $D_{j,i} \leq y$, therefore, i and j are connected in the neighborhood graph –*i.e.*, $(i, j) \in \mathcal{E}$. (a), (b) and (c) together show that if all factors above are non-zero, x defines a K-dominating set for $\mathcal{G}$.

(B) Any K-dominating set of $\mathcal{G}(D, y)$ defines an x with nonzero probability $\mu_y(x)$: Define x as follows: For all $i \in \mathcal{D}$ set $x_{i-i} = 1$. For any j with $x_{j-j} = 1$, select and $(j, i) \in \mathcal{E}$ where $x_{i-i} = 1$ and set $x_{j-i} = 1$. Since $\mathcal{D}$ is a dominating set, the existence of such an edge is guaranteed. It is easy to show that for an assignment x constructed this way, all μ_y-reduced factors above are non-zero and therefore $\mu_y(x) > 0$.

Proof 3.5.4 Here we re-enumerate the μ_y-reduction of factors for factor-graph of section 3.5.3.

A. *local factors* $\forall i \neq j \quad f^y_{\{i-j\}}(x_{i-j}) = \mathbb{I}(D_{i,j} \leq y \vee x_{i-j} = 0)$.

B. *uniqueness factors* For every i consider $I = \{i - j \mid 1 \leq j \leq N\}$, then $f^y_I(x_I) = \mathbb{I}(\sum_{i-j \in \partial I} x_{i-j} = 1)$.

C. *consistency factors* $\forall j, i \neq j$, $f^y(x_{\{j-j,i-j\}}) = \mathbb{I}(x_{j-j} = 0 \wedge x_{i-j} = 1)$.

D. *K-of-N factor* Let $\mathcal{K} = \{i - i \mid 1 \leq i \leq N\}$, then $f^y_{\mathcal{K}}(x_{\mathcal{K}}) = \mathbb{I}(\sum_{i-i \in \mathcal{K}} x_{i-i} = K)$.

(A) Any assignment x with $\mu_y(x) > 0$ defines an induced K-set-cover for $\mathcal{G}$: Let $\mathcal{C} = \{\mathcal{V}_i \mid x_{i-i} = 1\}$, where $\mathcal{V}_i = \{j \in \mathcal{V} \mid (j, i) \in \mathcal{E}\}$, as in the definition of an induced set-cover. Note that $\mathcal{E}$ refers to the edge-set of $\mathcal{G}(D, y)$.

Claim B.1.3 *$\mathcal{C}$ defines an induced K-set-cover for $\mathcal{G}(D, y)$.*

First, note that $\mathcal{C}$ as defined here is a subset of $\mathcal{S}$ as defined in the definition of an induced set-cover. Since $\mu_y(x) > 0$, all the μ_y-reduced factors above have a non-zero value for x. (a) Since the K-of-N factor is nonzero, by definition of $\mathcal{C}$ above, $|\mathcal{C}| = K$.

(b) Since uniqueness factors are non-zero for every node i, either $x_{i-i} = 1$, in which case i is covered by $\mathcal{V}_i \in \mathcal{C}$, or $x_{i-j} = 1$ for some $j \neq i$, in which case non-zero consistency factors imply that $\mathcal{V}_j \in \mathcal{C}$. (c) It only remains to show that if $x_{i-j} = 1$, then $(i-j) \in \mathcal{E}$. The non-zero local factors imply that for every $x_{i-j} = 1$, $D_{i-j} \leq y$. However, by definition of $\mathcal{G}(D, y)$, this also means that $(i, j) \in \mathcal{E}$. Therefore for any assignment x with $\mu_y(x) > 0$, we can define a unique $\mathcal{C}$ (an induced K-set-cover for $\mathcal{G}(D, y)$).

(B) Any induced K-set-cover for $\mathcal{G}$, defines an assignment x with , $\mu_y(x) > 0$: Here we need to build an assignment x from an induced K-set-cover $\mathcal{C}$ and show that all the factors in μ_y-reduction, have non-zero value and therefore, $\mu_y(x) > 0$. To this end, for each $\mathcal{V}_i \in \mathcal{C}$ set $x_{i-i} = 1$. Since $|\mathcal{C}| = K$, the K-of-N factor above will have a non-zero value. For any node j such that $x_{j-j} = 0$, select a cover $\mathcal{V}_i$ and set $x_{j-i} = 1$. Since $\mathcal{C}$ is a set-cover, the existence of at least one such $\mathcal{V}_i$ is guaranteed. Since we have selected only a single cover for each node, the uniqueness factor is non-zero. Also $x_{j-i} = 1$ only if $\mathcal{V}_i$ is a cover (and therefore $x_{i-i} = 1$), the consistency factors are non-zero. Finally, since $\mathcal{C}$ is an induced cover for $G(D, y)$, for any $j \in \mathcal{V}_i$, $D_{j,i} \leq y$ and therefore, $x_{j,i} = 1 \Rightarrow D_{j,i} \leq y$. This ensures that local factors are non-zero. Since all factors in the μ_y-reduced factor-graph are non-zero, $\mu_y(x) > 0$.

Proof 3.5.5 First, note that μ_y defines a uniform distribution over its support as its unnormalized value is only zero or one. Here, without loss of generality we distinguish between two Hamiltonian cycles that have a different starting point but otherwise represent the same tour. Consider the p_y-reduction of the pairwise factor of Equation

3.10

$$f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(f_{\{i,j\}}(x_i, x_j) \leq y) \tag{B.4}$$

$$= \mathbb{I}(|x_i - x_j| > 1) + \mathbb{I}(x_i = x_j - 1 \wedge D_{i,j} \leq y) \tag{B.5}$$

$$+ \mathbb{I}(x_i = x_j + 1 \wedge D_{j,i} \leq y) \tag{B.6}$$

(A) *Every Hamiltonian cycle over $\mathcal{G}(D, y)$, defines a unique assignments x with $\mu_y(x) > 0$:* Given the Hamiltonian cycle $H = h_0, h_2, \ldots, h_{N-1}$ where $h_i \in \{1, \ldots, N\}$ is the i^{th} node in the path, for each i define $x_i = j$ *s.t.* $h_j = i$. Now we show that all pairwise factors of eq. (B.4) are non-zero for x. Consider two variables x_i and x_j. If they are not consecutive in the Hamiltonian cycle then $f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(|x_i - x_j| > 1) > 0$. Now, without loss of generality assume i and j are consecutive and x_i appears before x_j. This means $(i, j) \in \mathcal{E}$ and therefore, $D_{i,j} \leq y$, which in turn means $f^{y}_{\{i,j\}}(x_i, x_j) = \mathbb{I}(x_i = x_j - 1 \wedge D_{i,j} \leq y) > 0$ Since all pairwise factors are non-zero, $\mu_y(x) > 0$.

(B) *Every x for which $\mu_y(x) > 0$, defines a unique Hamiltonian path over $\mathcal{G}(D, y)$:* Given assignment x, construct $H = h_0, \ldots, h_{N-1}$ where $h_i = j$ *s.t.*$x_j = i$. Now we show that if $\mu(x) > 0$, H defines a Hamiltonian path. If $\mu(x) > 0$, for every two variables x_i and x_i, one of the indicator functions of Equation B.4 should evaluate to one. This means that first of all, $x_i \neq x_j$ for $i \neq j$, which implies H is well-defined and $h_i \neq h_j$ for $i \neq j$. Since all $x_i \in \{0, \ldots, N-1\}$ values are distinct, for each $x_i = s$ there are two variables $x_j = s - 1$ and $x_k = s + 1$ (recall that we are using modular arithmetic) for which the pairwise factor of Equation B.4 is non-zero. This means $D_{j,i} \leq y$ and $D_{i,k} \leq y$ and therefore, $(j, i), (i, k) \in \mathcal{E}$ (the edge-set of $\mathcal{G}(D, y)$). But by definition of H, $h_s = i$, $h_{s-1} = j$ and $h_{s+1} = k$ are consecutive nodes in H and therefore, H is a Hamiltonian path.

B.2 Efficient message passing

B.2.1 K-of-N factors

For binary variables, it is convenient to assume all variable-to-factor messages are normalized such that $\nu_{j\to I}(0) = 1$. Now consider the task of calculating $\nu_{I\to i}(0)$ and $\nu_{I\to i}(1)$ for at-least-K-of-N factors. Let $\mathcal{A}(K)$ denote the subsets of $\mathcal{A}$ with at least K members. Then

$$\begin{aligned}\nu_{I\to i}(0) \quad &= \quad \sum_{\mathcal{X}_{\partial I\setminus i}} \mathbb{I}\Big(\sum_{j\in\partial I\setminus i} x_j \geq K\Big) \prod \nu_{j\to I}(x_j) = \\ &\qquad \sum_{(\partial I\setminus i)(K)} \prod_{j\in(\partial I\setminus i)(K)} \nu_{j\to I}(1) \end{aligned} \tag{B.7}$$

where in deriving Equation B.7 we have used the fact that $\nu_{j\to I}(0) = 1$. To calculate $\nu_{I\to i}(1)$ we follow the same procedure, except that here the factor is replaced by $\mathbb{I}(\sum_{j\in\partial I\setminus i} x_j \geq K-1)$. This is because here we assume $x_i = 1$ and therefore it is sufficient for $K-1$ other variables to be nonzero.

To evaluate Equation B.7, we use the dynamic programming recursion where another variable x_l for some $l \in \partial I$ is either zero or one:

$$\begin{aligned}\sum_{(\partial I\setminus i)(K)} \prod_{j\in(\partial I\setminus i)(K)} \nu_{j\to I}(1) = &\sum_{(\partial I\setminus i,l)(K)} \prod_{j\in(\partial I\setminus i,l)(K)} \nu_{j\to I}(1) \\ &+\nu_{l\to I}(1) \sum_{(\partial I\setminus i,l)(K)-1} \prod_{j\in(\partial I\setminus i,l)(K-1)} \nu_{j\to I}(1)\end{aligned}$$

This allows us to calculate these messages in $\mathcal{O}(NK)$.

B.2.2 Bottleneck TSP factors

The μ_y-reduction of the min-max factors of BTSP is given by:

$$f^y_{\{i,j\}}(x_i, x_j) = \mathbb{I}(f_{\{i,j\}}(x_i, x_j) \leq y) \tag{B.8}$$

$$= \mathbb{I}(|x_i - x_j| > 1) + \mathbb{I}(x_i = x_j - 1 \wedge D_{i,j} \leq y) \tag{B.9}$$

$$+ \mathbb{I}(x_i = x_j + 1 \wedge D_{j,i} \leq y) \tag{B.10}$$

The matrix-form of this factor (depending on the order of $D_{i,j}$, $D_{j,i}, y$) takes several forms all of which are band-limited. Assuming the variable-to-factor messages are normalized (*i.e.*, $\sum_{x_i} \nu_{j\to I}(x_i) = 1$) the factor-to-variable message is given by

$$\nu_{\{i,j\}\to i}(x_i) = 1 - \nu_{j\to\{i,j\}}(x_i) +$$
$$\mathbb{I}(D_{i,j} \leq y)(1 - \nu_{j\to\{i,j\}}(x_i - 1)) +$$
$$\mathbb{I}(D_{j,i} \leq y)(1 - \nu_{j\to\{i,j\}}(x_i + 1))$$

B.3 Analysis of complexity

Here, for factor-graphs used in section 3.5, we provide a short analysis of complexity.

Min-max Clustering:

This formulation includes N^2 pairwise factors – one for every pair of variables – and the cost of sending each factor to variable message is $\mathcal{O}(K)$, resulting in $\mathcal{O}(N^2K)$ complexity for all factor-to-variable messages. The cost of sending variable-to-factor messages is also $\mathcal{O}(N^2K)$. This gives $\mathcal{O}(N^2K\log(N))$ cost for finding the approximate min-max solution. The $\log(N)$ factor reflects the complexity of binary search, which depends on the diversity of range of factors –*i.e.*, $\log(|\mathcal{Y}|) = \log(N^2) = 2\log(N)$.

K-Packing (binary variable):

The time complexity of Perturbed BP iterations in this factor-graph is dominated by factor-to-variable messages sent from $f_{\mathcal{K}}(x)$ to all variables. Assuming asynchronous update, the complexity for min-max procedure is $\mathcal{O}(N^2K\log(N))$.

K-Packing (categorical variable):

Since the factors are not sparse, the complexity of factor-to-variable messages are $\mathcal{O}(N^2)$, resulting in $\mathcal{O}(N^2K^2)$ cost per iteration for μ_y-reduction. Since the diversity of pairwise distances is $|\mathcal{Y}| = \mathcal{O}(N^2)$, the general cost of finding an approximate min-max solution by message passing is $\mathcal{O}(N^2K^2\log(N))$.

Sphere-Packing (Hamming distance):

The total number of variables (including x and z) in this factor-graph is $N = Kn + \frac{K(K-1)}{2}n = \mathcal{O}(nK^2)$. The factor-graph contains $\frac{K(K-1)}{2}n$ z-factors and $\frac{K(K-1)}{2}$ distance factors. Each x_{i-k} variable receives messages from $K-1$ z-factors while each $z(i,j)_k$ variable is connected to one z-factor and one distance factor. Assuming an asynchronous message update, the cost of factor-to-variable messages from distance-factors (*i.e.*, $\mathcal{O}(n^2y)$) dominates the complexity of Perturbed BP's iterations. Since there are K^2 such factors, the complexity of message passing per iteration is $\mathcal{O}(K^2n^2y)$.

K-Center Problem:

Assuming an asynchronous update schedule, the cost of sending messages from uniqueness factors dominates the complexity of Perturbed BP's iteration resulting in $\mathcal{O}(N^3)$ complexity for μ_y-reduction and $\mathcal{O}(N^3\log(N)$ for the min-max problem.

Bottleneck Traveling Salesman Problem:

In Section B.2.2 we show that the μ_y-reduction of BTSP factors of Figure 3.10 allows efficient marginalization in $\mathcal{O}(N)$. Since there are N^2 factors in this factor-graph and the cost of sending factor-to-variable messages is $\mathcal{O}(N)$, the general cost of min-max

inference is $\mathcal{O}(N^3 \log(N))$.

Bibliography

[1] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, 2009.

[2] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46:325–343, 2000.

[3] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law and free energy minimization. In *Proceedings of the 39th Allerton Conference*, 2001.

[4] Hyung-Chan An, Robert D. Kleinberg, and David B. Shmoys. Approximation algorithms for the bottleneck asymmetric traveling salesman problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 1–11. Springer, 2010.

[5] Igor Averbakh. On the complexity of a class of combinatorial optimization problems with uncertainty. *Mathematical Programming*, 90(2):263–272, 2001.

[6] Mohsen Bayati, Devavrat Shah, and Mayank Sharma. Maximum weight matching via max-product belief propagation. In *ISIT*, pages 1763–1767. IEEE, 2005.

[7] Dhiren K. Behera. Complexity on parallel machine scheduling: A review. In S Sathiyamoorthy, B. Elizabeth Caroline, and J Gnana Jayanthi, editors, *Emerg-*

ing Trends in Science, Engineering and Technology, Lecture Notes in Mechanical Engineering, pages 373–381. Springer India, 2012.

[8] Dhiren Kumar Behera and Dipak Laha. Comparison of heuristics for identical parallel machine scheduling. *Advanced Materials Research*, 488:1708–1712, 2012.

[9] Indaco Biazzo, Alfredo Braunstein, and Riccardo Zecchina. Performance of a cavity-method-based algorithm for the prize-collecting steiner tree problem on graphs. *Physical Review E*, 86(2):026706, 2012.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[11] Alfredo Braunstein, Marc Mézard, Martin Weigt, and Riccardo Zecchina. Constraint satisfaction by survey propagation. *Computational Complexity and Statistical Physics*, page 107, 2005.

[12] Alfredo Braunstein, Marc Mezard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27:201–226, 2005.

[13] Alfredo Braunstein, Roberto Mulet, Andrea Pagnan, Martin Weigt, and Riccardo Zecchina. Polynomial iterative algorithms for coloring and analyzing random graphs. *Physical Review E*, 68:36702, 2003.

[14] Alfredo Braunstein and Riccardo Zecchina. Survey propagation as local equilibrium equations. *Journal of Statistical Mechanics: Theory and Experiment*, 2004:06007, 2004.

[15] Michael Chertkov and Vladimir Y. Chernyak. Loop series for discrete statistical models on graphs. *Journal of Statistical Mechanics: Theory and Experiment*, 2006(06):P06009, 2006.

[16] Hai L. Chieu, Wee S. Lee, and Yee W. Teh. Cooled and relaxed survey propagation for MRFs. In *Advances in Neural Information Processing Systems*, pages 297–304, 2007.

[17] Julia Chuzhoy, Sudipto Guha, Eran Halperin, Sanjeev Khanna, Guy Kortsarz, Robert Krauthgamer, and Joseph Seffi Naor. Asymmetric k-center is log* n-hard to approximate. *Journal of the ACM (JACM)*, 52(4):538–551, 2005.

[18] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2):393–405, 1990.

[19] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1):5–41, 2001.

[20] Martin E. Dyer and Alan M. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

[21] Frederik Eaton and Zoubin Ghahramani. Model reductions for inference: Generality of pairwise, binary, and planar factor graphs. *Neural Computation*, 25:1213–1260, 2013.

[22] Jack Edmonds and Delbert Ray Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299–306, 1970.

[23] Stefano Ermon, Ashish Sabharwal, Bart Selman, and Carla P. Gomes. Density propagation and improved bounds on the partition function. In *Advances in Neural Information Processing Systems*, pages 2762–2770, 2012.

[24] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[25] Brendan J. Frey and David J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In *Advances in Neural Information Processing Systems (NIPS)*, 1997.

[26] Delbert R. Fulkerson. Flow networks and combinatorial operations research. *The American Mathematical Monthly*, 73(2):115–138, 1966.

[27] Mitchell H. Gail, Jay H. Lubin, and Lawrence V. Rubinstein. Likelihood calculations for matched case-control studies and survival studies with tied death times. *Biometrika*, 68(3):703–707, 1981.

[28] Michael R. Garey and David S. Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, 1979.

[29] Robert S. Garfinkel. An improved algorithm for the bottleneck assignment problem. *Operations Research*, pages 1747–1751, 1971.

[30] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.

[31] Inmar E. Givoni and Brendan J. Frey. A binary variable model for affinity propagation. *Neural computation*, 21(6):1589–1600, 2009.

[32] Vicenç Gómez, Joris M. Mooij, and Hilbert J. Kappen. Truncating the loop series expansion for belief propagation. *Journal of Machine Learning Research*, 8:1987–2016, 2007.

[33] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[34] Roland L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[35] O. Gross. The bottleneck assignment problem. Technical report, DTIC Document, 1959.

[36] Jatinder N. D. Gupta and Alex J. Ruiz-Torres. A listfit heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control*, 12(1):28–36, 2001.

[37] Rahul Gupta, Ajit A. Diwan, and Sunita Sarawagi. Efficient inference with cardinality-based clique potentials. In *Proceedings of the 24th international conference on Machine learning*, pages 329–336. ACM, 2007.

[38] Tom Heskes. Stable fixed points of loopy belief propagation are local minima of the bethe free energy. In *Advances in Neural Information Processing Systems 15*, pages 359–366. 2003.

[39] Tom Heskes. Convexity arguments for efficient minimization of the bethe and kikuchi free energies. *Journal of Artificial Intelligence Research*, 26:153–190, 2006.

[40] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[41] Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

[42] Morteza Ibrahimi, Adel Javanmard, Yashodhan Kanoria, and Andrea Montanari. Robust max-product belief propagation. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, pages 43–49. IEEE, 2011.

[43] Santosh N. Kabadi and Abraham P. Punnen. The bottleneck tsp. In *The Traveling Salesman Problem and Its Variations*, pages 697–735. Springer, 2004.

[44] Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Incomplete algorithms. In *Handbook of Satisfiability*, pages 185–203. 2009.

[45] Michael Kearns, Michael L. Littman, and Satinder Singh. Graphical models for game theory. In *Proceedings of the Seventeenth conference on UAI*, pages 253–260. Morgan Kaufmann Publishers Inc., 2001.

[46] Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

[47] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1568–1583, 2006.

[48] Lukas Kroc, Ashish Sabharwal, and Bart Selman. Survey propagation revisited. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 217–226, 2007.

[49] Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498 –519, 2001.

[50] Steffen L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[51] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[52] Simon Litsyn, Rains E. M., and Sloane N. J. A. The table of nonlinear binary codes. `http://www.eng.tau.ac.il/~litsyn/tableand/`, 1999. [Online; accessed Jan 30 2014].

[53] Elitza N. Maneva, Elchanan Mossel, and Martin J. Wainwright. A new look at survey propagation and its generalizations. *Journal of ACM*, 54:2–41, 2007.

[54] Shigeru Masuyama, Toshihide Ibaraki, and Toshiharu Hasegawa. The computational complexity of the m-center problems on the plane. *IEICE TRANSACTIONS (1976-1990)*, 64(2):57–64, 1981.

[55] Marc Mezard and Andrea Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009.

[56] Marc Mezard, Giorgio Parisi, and Riccardo Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812–815, 2002.

[57] Andrea Montanari and Tommaso Rizzo. How to compute loop corrections to the bethe approximation. *Journal of Statistical Mechanics: Theory and Experiment*, 2005:16, 2005.

[58] Joris M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, 2010.

[59] Joris M. Mooij and Hilbert J. Kappen. Loop corrections for approximate inference on factor graphs. *Journal of Machine Learning Research*, 8:1113–1143, 2007.

[60] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[61] Rina Panigrahy and Sundar Vishwanathan. An o(log*n) approximation algorithm for the asymmetric p-center problem. *Journal of Algorithms*, 27(2):259–268, 1998.

[62] R Gary Parker and Ronald L Rardin. Guaranteed performance heuristics for the bottleneck travelling salesman problem. *Operations Research Letters*, 2(6):269–272, 1984.

[63] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988.

[64] Alessandro Pelizzola. Cluster variation method in statistical physics and probabilistic graphical models. *Journal of Physics A: Mathematical and General*, 38:36, 2005.

[65] Michael Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012.

[66] Brian Potetz and Tai Sing Lee. Efficient belief propagation for higher-order cliques using linear constraint nodes. *Computer Vision and Image Understanding*, 112(1):39–54, 2008.

[67] Abolfazl Ramezanpour and Riccardo Zecchina. Cavity approach to sphere packing in hamming space. *Physical Review E*, 85(2):021106, 2012.

[68] Siamak Ravanbakhsh and Russell Greiner. Revisiting algebra and complexity of inference in graphical models. *arXiv.org preprint*, arXiv:1409.7410, 2014.

[69] Siamak Ravanbakhsh and Russell Greiner. Perturbed message passing for constraint satisfaction problems. *Journal of Machine Learning Research*, 16:1249–1274, 2015.

[70] Thomas J. Richardson and Rudiger L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47:599–618, 2001.

[71] Alexander Schrijver. *Min-max results in combinatorial optimization*. Springer, 1983.

[72] Daniel Tarlow, Inmar E. Givoni, and Richard S. Zemel. Hop-map: Efficient message passing with high order potentials. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 812–819, 2010.

[73] Daniel Tarlow, Kevin Swersky, Richard S. Zemel, Ryan Prescott Adams, and Brendan J. Frey. Fast exact inference for recursive cardinality models. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pages 825–834, 2012.

[74] Ronghui Tu, Yongyi Mao, and Jiying Zhao. Is SP BP? *IEEE Transactions on Information Theory*, 56:2999–3032, 2010.

[75] Meritxell Vinyals, Kathryn S. Macarthur, Alessandro Farinelli, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A message-passing approach to decentralized parallel machine scheduling. *The Computer Journal*, 57(6):856–874, 2014.

[76] Martin J. Wainwright, Tommi S. Jakkolla, and Alan S. Willsky. Map estimation via agreement on trees: Message-passing and linear programming. *IEEE Transactions on Information Theory*, 51:3697–3717, 2005.

[77] Martin J. Wainwright and Elitza Maneva. Lossy source encoding via message-passing and decimation over generalized codewords of LDGM codes. In *Proceedings of the International Symposium on Information Theory*, 2005.

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

[79] Yair Weiss, Chen Yanover, and Talya Meltzer. MAP estimation, linear programming and belief propagation with convex free energies. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, 2007.

[80] Chen Yanover, Talya Meltzer, and Yair Weiss. Linear programming relaxations and belief propagation an empirical study. *Journal of Machine Learning Research*, 7:18871907, 2006.

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

[82] Jonathan S. Yedidia, William T. Freeman, Yair Weiss, et al. Generalized belief propagation. In *NIPS*, volume 13, pages 689–695, 2000.

[83] Wei Yu and Marko Aleksic. Coding for the blackwell channel: a survey propagation approach. In *Proceedings of the International Symposium on Information Theory*, 2005.

[84] Alan L. Yuille. CCCP algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14:1691–1722, 2002.

[85] Lenka Zdeborov and Florent Krzkaa. Phase transitions in the coloring of random graphs. *Physical Review E*, 76(3):031131, September 2007.