Technical University of Crete
School of Electrical and Computer Engineering

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Delay-Constrained Distributed Inference in Wireless Networks

by Mitrolaris Stavros

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DIPLOMA OF

ELECTRICAL AND COMPUTER ENGINEERING

February 2023

THESIS COMMITTEE

Professor Aggelos Bletsas, *Thesis Supervisor*
Professor Dionysios Christopoulos
Professor Antonios Deligiannakis

Abstract

Belief propagation (BP)-based algorithms are powerful message-passing algorithms that exploit the conditional independences of specific variables on a carefully crafted graph and efficiently compute marginal distributions or maximum a posteriori (MAP) estimates (of the involved variables). Motivated by the convergence guarantees of Gaussian belief propagation (GBP) under high-order factorization and asynchronous scheduling, we study how this algorithm can be utilized by resource-limited wireless networks for in-network processing. We show that there are cases where a particular asynchronous variant of GBP converges in theory but diverges when transmission delays are included. A simple solution is proposed, based on a coordinator, and its performance in terms of convergence time is examined, through simulations; optimized resource allocation is performed, including the bottleneck assignment problem (BAP) formulation, taking into account transmission delays, as well as bandwidth-limited wireless channels and external interference. Next, we study the max-product BP algorithm, as a distributed solver of BAP, i.e., a matching problem between tasks and agents, with the objective of minimizing the costliest pairing, for which only a couple of distributed algorithms currently exist. We examine a line of work which addresses this problem, provided that a unique solution exists and simplify the message calculations; specifically, we propose new BP message expressions, with linear complexity (as opposed to quadratic of prior art). Furthermore, we provide an asynchronous variant and study its convergence and correctness guarantees, using the notion of generalized computation trees. Simulations results show convergence of both the synchronous and asynchronous variant.

Acknowledgments

This thesis marks the end of my undergraduate studies and there are several people I would like to thank.

First and foremost, I owe a debt of gratitude to my supervisor, Prof. Aggelos Bletsas. His support during the trials and tribulations of my thesis taught me how to stay focused and motivated when nothing goes as planned. His passion for research set an example for me.

I would like to thank Prof. Athanasios Liavas for his awesome courses, genuine interest and support when I needed it most. I am also grateful to Prof. Michail Lagoudakis, who with his kindness and superb time management skills, helped me in critical times. Additionally, I would like to thank Prof. Daphne Manoussaki for giving me the opportunity to serve as teaching assistant, and Profs. Christopoulos and Deligiannakis for taking the time to evaluate my thesis.

Last but not least, my family and close friends have always been there for me. I can not thank them enough.

Contents

Chapter 1

Introduction

Many real world problems are modeled as a collection of random variables with a joint probability distribution, which is learned from data. Through this probabilistic approach we can perform inference; i.e., make predictions under uncertainty. This usually involves calculating posterior beliefs or finding maximum a posteriori (MAP) estimates, which in general are NP-hard problems if no assumptions are made [1]. Belief propagation algorithms exploit the conditional independencies of the joint distribution; a graphical model (GM) is used to represent them and messages over the edges of the GM are exchanged, managing to compute the desired quantities efficiently [2]. The algorithms' wide range of applicability, inherent distributed characteristics and ease of implementation, are the main reasons for their popularity.

Wireless sensor networks (WSNs) consist of a number of terminals each of which collects ambient information. When a central controller (CC) is utilized, all the information from the terminals is aggregated and the CC solves the problem centrally. On the other hand, in the case of in-network processing, terminals communicate with each other and solve the problem collectively. The algorithms used in the latter case need not only to be distributed, but also robust to asynchronous operations, since the terminals have limited energy and their communication may fail.

Recent theoretical guarantees for the convergence of Gaussian Belief Propagation (GBP)(BP when the joint distribution is Gaussian) under asynchronous scheduling [3] were extended and utilized by [4, 5] to show that the algorithm can be used by a WSN for autonomous, in-network processing; a proof of concept was also provided.

In the first part of this thesis, it is shown that an asynchronous variant of GBP which converges, in theory, may diverge when implemented on a WSN. A simple and efficient solution is proposed and simulations results are provided, where wireless channels which are limited in energy and bandwidth, as well as subject to noise and interference, are considered.

In the second part we study a combinatorial optimization problem, namely, the Bottleneck Assignment Problem (BAP). Even though the BAP has been studied thoroughly over the last 70 years [6], only two distributed algorithms have been proposed [7, 8], in 2019 and 2020, respectively. Generic criteria for solving similar problems with BP (which is inherently distributed) exist, however those hold when the cost function is linear; in BAP it's not. A line of work which solves the BAP with BP [9] is studied and some improvements are offered,

namely, new BP message expressions as well as convergence and correctness guarantees for its asynchronous variant.

Thesis Outline

In Chapter 2 we provide some details about BP, in Chapter 3 we discuss how a WSN can utilize the GBP for in-network processing, why an asynchronous variant may diverge and how we can solve this issue. In Chapter 4 we explain the BAP, discuss some existing algorithms and the improvements we propose. Finally, in Chapter 5 we state the conclusions of this thesis and possible future work directions.

Notation

Scalars, vectors and matrices are denoted by lower-case letters, bold lower-case letters and bold upper-case letters, respectively. For a vector $\mathbf{x}$, x_i denotes its i-th element. For a matrix $\mathbf{A}$, A_{ij} is the entry in its i-th row and j-th column. $\mathbf{O}$ is the matrix whose all entries are zero. The notation $\mathbf{C} \succ \mathbf{O}$ indicates that $\mathbf{C}$ is positive definite. $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ indicates that the random vector $\mathbf{x}$ follows multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{C}$. When two random variables x, y are independent, we write $x \perp\!\!\!\perp y$. $\rho(\mathbf{A})$ denotes the spectral radius of matrix $\mathbf{A}$, namely, $\rho(\mathbf{A}) = \max\{|\lambda_1|, |\lambda_2|, \ldots, |\lambda_m|\}$, where $\lambda_1, \lambda_2, \ldots, \lambda_m$ are the eigenvalues of $\mathbf{A}$. For a set $\mathcal{B}$, the notation $\mathcal{B}\backslash i$ denotes all the elements in $\mathcal{B}$ except i. The notation $a \propto b$ denotes that a is proportional to b.

Chapter 2

Inference and Probabilistic Graphical Models

2.1 Inference

Consider a collection of random variables $\mathbf{x} = (\mathsf{x}_1, \ldots, \mathsf{x}_N)$ and let the observations about them be represented by random variables $\mathbf{y} = (\mathsf{y}_1, \ldots, \mathsf{y}_N)$. Let each of these random variables $\mathsf{x}_i, 1 \leq i \leq N$, take on a value in $\mathcal{X}$ (e.g., $\mathcal{X} = \{0, 1\}$ or $\mathbb{R}$) and each observation variable $\mathsf{y}_i, 1 \leq i \leq N$ take on a value in $\mathcal{Y}$. Given observation $\mathbf{y} = \mathbf{y}$, the goal is to say something meaningful about possible realizations of $\mathbf{x} = \mathbf{x}$ for $\mathbf{x} \in \mathcal{X}^N$ [10, 2]. There are two primary computation problems of interest:

1. Calculating (posterior) beliefs:

$$\begin{aligned} p_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y}) &= \frac{p_{\mathbf{x},\mathbf{y}}(\mathbf{x}, \mathbf{y})}{p_{\mathbf{y}}(\mathbf{y})} \\ &= \frac{p_{\mathbf{x},\mathbf{y}}(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{x}' \in \mathcal{X}^N} p_{\mathbf{x},\mathbf{y}}(\mathbf{x}', \mathbf{y})}. \end{aligned}$$

   The denominator is also called the partition function. In general, computing the partition function is expensive. This is because, without any additional structure, a distribution over N variables with each variable taking on values in $\mathcal{X}$ requires storing a table of size $|\mathcal{X}|^N$, where each entry contains the probability of a particular realization. So without structure, computing posterior has complexity exponential in the number of variables N.

2. Calculating the most probable configuration also called the maximum a posteriori (MAP) estimate:

$$\hat{\mathbf{x}} \in \arg\max_{\mathbf{x} \in \mathcal{X}^N} p_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y}).$$

Therefore,

$$\begin{aligned}\hat{\mathbf{x}} &\in \arg\max_{\mathbf{x}\in\mathcal{X}^N} p_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y}) \\ &= \arg\max_{\mathbf{x}\in\mathcal{X}^N} \frac{p_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y})}{p_{\mathbf{y}}(\mathbf{y})} \\ &= \arg\max_{\mathbf{x}\in\mathcal{X}^N} p_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y}).\end{aligned}$$

Without any additional structure, the above optimization problem requires searching over the entire space X^N, resulting in an exponential complexity in the number of variables N.

The above computations can be greatly simplified if we exploit the (conditional) independences. In a best-case scenario, the N random variables from before are independent, meaning that we have the following factorization:

$$p_{\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_N}(x_1,x_2,\ldots,x_N) = p_{\mathsf{x}_1}(x_1)p_{\mathsf{x}_2}(x_2)\ldots p_{\mathsf{x}_N}(x_N).$$

Then posterior belief calculation can be done separately for each variable. Computing the posterior belief of a particular variable has complexity $|\mathcal{X}|$. Similarly, MAP estimation can be done by finding each variable's assignment that maximizes its own probability. As there are N variables, the computational complexity of MAP estimation is thus $N|\mathcal{X}|$.

Thus, independence or some form of factorization enables efficient computation of both posterior beliefs (marginalization) and MAP estimation. By exploiting factorizations of joint probability distributions and representing these factorizations via graphical models, we achieve huge computational efficiency gains.

2.2 Probabilistic Graphical Models

There exist three main types of graphical models, here are two of them[1] which we will use later:

1. *Undirected Graphical Models*

   An undirected graphical model is a graph $G = (V, E)$, where the nodes V correspond to variables and the undirected edges $E \subset V \times V$ tell us about the conditional independence structure. The undirected graph defines a family of probability distributions which satisfy the following graph separation property.

   - $\mathsf{x}_A \perp\!\!\!\perp \mathsf{x}_B|\mathsf{x}_C$ whenever there is no path from a node in A to a node in B which does not pass through a node in C.

   Undirected graphical models represent the distribution as a product of functions called potentials, times a normalization constant. A clique is a fully connected set of nodes. A maximal clique is a clique that is not a strict subset of another clique. Given a set of variables $\mathsf{x}_1,\ldots,\mathsf{x}_N$ and a set $\mathcal{C}$ of maximal cliques, the following factorization

[1] The other is Directed Acyclic graph or Bayesian network.

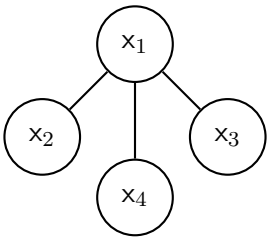


Figure 2.1: Example of undirected graphical model with 4 nodes.

characterizes undirected graphical models, provided that the distribution p is strictly positive:

$$\begin{aligned} p_{\mathbf{x}}(\mathbf{x}) &\propto \prod_{C \in \mathcal{C}} \psi_{\mathsf{x}_C}(x_c) \\ &= \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{\mathsf{x}_C}(x_c), \end{aligned}$$

where the functions ψ can be any non-negative valued functions, and Z is called the partition function which is chosen such that it normalizes the probabilities.

$$Z = \sum_{\mathbf{x}} \prod_{C \in \mathcal{C}} \psi_{\mathsf{x}_C}(x_c).$$

For many calculations, such as conditional probabilities and finding the most likely joint assignment, we do not need to compute Z.

Consider the undirected graphical model of Fig. 2.1, since each edge corresponds to a maximal clique, the family of distributions that represents can be factorized as

$$p_{\mathbf{x}}(\mathbf{x}) \propto \psi_{\mathsf{x}_1,\mathsf{x}_2}(x_1, x_2) \cdot \psi_{\mathsf{x}_1,\mathsf{x}_4}(x_1, x_4) \cdot \psi_{\mathsf{x}_1,\mathsf{x}_3}(x_1, x_3).$$

2. *Factor Graphs* A factor graph consists of a vector of random variables $\mathbf{x} = (\mathsf{x}_1, \ldots, \mathsf{x}_N)$ and a graph $G = (V, E, \mathcal{F})$, which in addition to normal nodes also has factor nodes $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$. Furthermore, the graph is constrained to be a bipartite graph between variable nodes and factor nodes.

   The joint probability distribution associated to a factor graph is given by:

$$p_{\mathbf{x}}(x_1, \ldots, x_N) = \frac{1}{Z} \prod_{j=1}^{m} f_j(X_j),$$

   where X_j is a subset of $(x_1, \ldots, x_N)$. The only constraints imposed on the factors is that they must be non-negative.

   Consider the factor graph of Fig. 2.2, then the following factorization can be derived

$$p_{\mathbf{x}}(x_1, \ldots, x_4) \propto f_1(x_1, x_3, x_4) \cdot f_2(x_2, x_3).$$

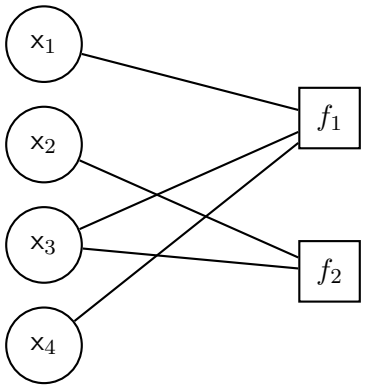


Figure 2.2: Example of factor graph with 4 variables and 2 factors.

2.3 Computation Trees

Depending on the context, we use the term Belief Propagation when we refer to the sum-product or max-product algorithm. The first aims at computing marginal distributions whereas the latter at finding maximum a posteriori estimates through computing max-marginal beliefs [2, 10]. Both operate in the same fashion: they exploit the conditional independences represented by a graphical model and use messages exchanged among the nodes to compute (max-)marginal distributions.

Each message is a function of incoming messages, hence, in order to update the messages serially, a sequence of updates needs to be established. When the GM has a tree structure, an arbitrary node is chosen as the root, and we start by computing the messages of the leaf nodes and work our way up towards the root, then we work outwards from the root. BP is also parallelizable. In that case, messages are properly initialized and all of them are updated in parallel. For tree GM, it is known that this procedure converges to the correct messages in d iterations, where d is the length of the longest path on the tree (a.k.a. diameter of the tree).

On loopy graphs we do not have the notion of leaves, thus we resort to the parallel version, referring to it as loopy BP, since the serial one can not be utilized. The messages may not converge at all, but even they do there are cases where they yield nonsensical marginal distributions. In many special cases of loopy graphs, BP converges to the correct solution.

A popular method of analysis is that of computation trees. We use dependency diagrams to provide more of a dynamic view of loopy BP through visualizing how messages across iterations contribute to the computation of a node marginal.

Let $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$, be an undirected graph. By $N(i)$ denote the set of neighbors of node i. The computation tree is constructed by replicating the local connectivity of the original graph. For any $i \in V$, the computation tree corresponding to i after t iterations is a tree of height $t+1$ rooted at i. All tree nodes have labels from the set $\{1, \ldots, n\}$ according to the following recursive rules [11]:

1. The root has label i.
2. Each $j \in N(i)$ is a children of the root i.

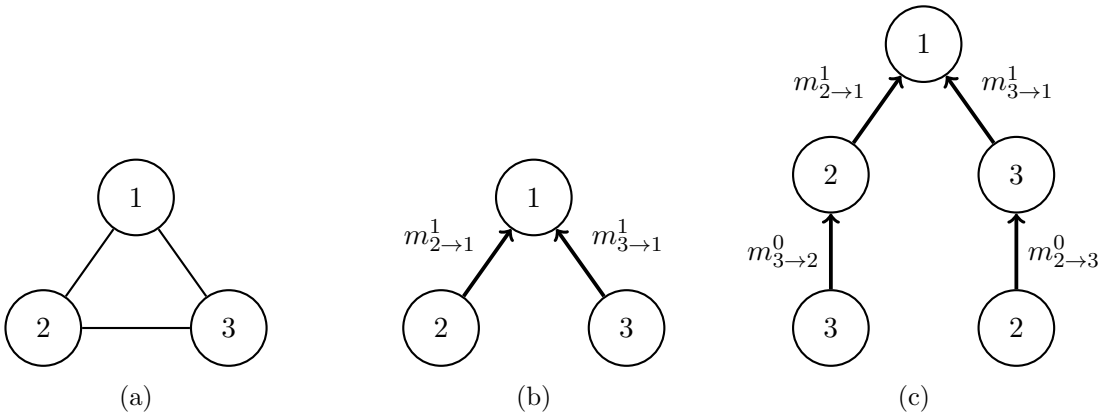


Figure 2.3

3. If s is a non-leaf node whose parent has label r, then the set of labels of its children is $N(s)\backslash\{r\}$.

Here is an example from [10]. Consider a probability distribution with the loopy graph in Fig. 2.3a. Suppose we ran loopy BP for exactly one iteration and computed the marginal at node 1. Then this would require us to look at messages $m^1_{2\to1}$ and $m^1_{3\to1}$, where the superscripts denote the iteration number. We visualize these two dependencies in Fig. 2.3b. To compute $m^1_{2\to1}$, we needed to look at message $m^0_{3\to2}$, and to compute $m^1_{3\to1}$, we needed to look at $m^0_{2\to3}$. These dependencies are visualized in Fig. 2.3c. Note that loopy BP is operating as if the underlying graph is the computation tree, not realizing that some nodes are duplicates.

In general, for the computation tree with root node i, the messages received by node i in the belief propagation algorithm after t iterations in graph G are equivalent to those that would have been received by the root i in the computation tree, if the messages were passed up along the tree from the leaves to the root.

Chapter 3

Distributed Inference in Wireless Sensor Networks with Transmission Delays

In this chapter we firstly discuss how a Wireless Sensor Network (WSN) can autonomously, without a central controller, solve an affine fixed point (AFP) problem. Then, we study the Gaussian Belief Propagation (GBP) algorithm and see why its convergence, under synchronous and asynchronous scheduling, comes down to solving an AFP problem. The transmission delays of a WSN can affect the convergence of an asynchronous variant, so we propose a simple solution. Through simulations, it is shown that what we propose is efficient, in terms of convergence time.

3.1 Computations in a Wireless Sensor Network

Autonomous, in-network processing refers to the capability of the wireless sensor network (WSN) to solve a problem without a central processing unit, which accumulates all the information. In particular, each terminal performs computations based on some local information and by exchanging messages with others, they manage to find the solution collectively. In this work we are interested in utilizing a WSN so as to solve a fixed point problem in a distributed manner. Mathematically, let $\mathbf{x}^{(0)}, \mathbf{b} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{n \times n}$. We want to find the fixed point of the recursion:

$$\mathbf{x}^{(l)} = \mathbf{A}\mathbf{x}^{(l-1)} + \mathbf{b}, \tag{3.1}$$

that is, $\mathbf{x}^* \triangleq \lim_{l \to \infty} \mathbf{x}^{(l)}$. Notice that in Eq. 3.1 all elements of $\mathbf{x}$ are updated in every iteration. We refer to this property as *synchronous scheduling*.

Consider a WSN with N physical terminals. Each terminal w_i is responsible for the calculation of a unique subset of the elements of $\mathbf{x}^{(l)}$, denoted by $\mathbf{x}_{\mathcal{I}_i}^{(l)}$, where $\cap_{i=1}^{N} \mathcal{I}_i = \emptyset$ and $\cup_{i=1}^{N} \mathcal{I}_i = \{1, \ldots, n\}$. The elements of $\mathbf{x}^{(l)}$ are calculated as

$$x_k^{(l)} = \sum_{j=1}^{n} \alpha_{kj} x_j^{(l-1)} + b_k, \quad l = 1, 2, \ldots, \quad k = 1, 2, \ldots, n, \tag{3.2}$$

where $x_k^{(l)}$ and b_k denote the k-th element of $\mathbf{x}^{(l)}$ and $\mathbf{b}$, respectively, and α_{ij} the element of the i-th row and j-th column of $\mathbf{A}$. From Eq. 3.2 we can see that for the k-th element of $\mathbf{x}^{(l)}$, the terminal responsible for its update must know the k-th row of $\mathbf{A}$ and the corresponding element of $\mathbf{b}$; thus, each terminal w_i must store locally those values. Additionally, note that for the calculation of $x_k^{(l)}$, access to elements that are allocated to different terminals may be required, hence communication between terminals is necessary.

Eq. 3.1 assumes that the communication between terminals is always successful. However, energy limitations and transmission failures interrupt the necessary message passing stochastically. In such scenarios, there are elements of $\mathbf{x}^{(l)}$ that are not updated and retain their previous values from the $l-1$ iteration. That is what from now on we will refer to as *asynchronous scheduling*.

We adopt the formulation of [3] in order to provide a new expression, for the recursion of Eq. 3.1, that adheres to asynchronous scheduling. At iteration l, define the functions $\psi_k^{(l)}$, $\forall k \in \{1, 2, \ldots, n\}$ as

$$\psi_k^{(l)} = \begin{cases} 1, & \text{if } x_k \text{ is updated at iteration } l, \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Let $\boldsymbol{\psi}^{(l)}$ denote the binary vector that is created if we stack all $\psi_k^{(l)}$ at iteration l and $\boldsymbol{\Psi}^{(l)} \triangleq \operatorname{diag}\left\{\boldsymbol{\psi}^{(l)}\right\}$ the corresponding diagonal matrix with $\boldsymbol{\psi}^{(l)}$ at its diagonal. The asynchronous formulation of Eq. 3.1 is:

$$\begin{aligned} \mathbf{x}^{(l)} &= \boldsymbol{\Psi}^{(l)} \left(\mathbf{A}\mathbf{x}^{(l-1)} + \mathbf{b}\right) + \left(\mathbf{I} - \boldsymbol{\Psi}\right) \mathbf{x}^{(l-1)} \\ &= \left(\boldsymbol{\Psi}^{(l)}\mathbf{A} + \mathbf{I} - \boldsymbol{\Psi}^{(l)}\right) \mathbf{x}^{(l-1)} + \boldsymbol{\Psi}^{(l)}\mathbf{b}. \end{aligned} \tag{3.4}$$

Notice that Eq. 3.1 is a special case of Eq. 3.4, where $\boldsymbol{\Psi}^{(l)} = \mathbf{I}$; i.e., when all elements are updated in every iteration.

In this work we assume that there are no transmission failures. As a consequence, whether a terminal i updates its values $\mathbf{x}_{\mathcal{I}_i}^{(l)}$, solely depends on whether it has sufficient energy to participate in iteration l. If it does not, it is still considered capable of receiving and storing the most up-to-date messages. Intuitively, we want those with low energy status to update their values as rarely as possible, in order to save energy, whereas those with more energy should be able to update more frequently. Note that the energy level of a sensor is a probabilistic quantity.

Following the work in [3, 4], we assume that the functions $\psi_k^{(l)}$ are Bernoulli random variables, independent across the different iterations l but possibly dependent and not identically distributed across k, with parameters p_k, $\forall k \in \{1, 2, \ldots, n\}$. We will refer to this asynchronous scheduling as *non-i.i.d. asynchronous scheduling*. Now we can define the expected

value of matrices $\mathbf{\Psi}^{(l)}$, $\mathbb{E}[\mathbf{\Psi}^{(l)}] \triangleq \mathbf{P} = \text{diag}\{\mathbf{p}\}$, where $\mathbf{p} = \mathbb{E}[\boldsymbol{\psi}^{(l)}]$, a quantity that as we will see plays a major role in the convergence of recursion Eq. 3.4.

Remark 1. Throughout this work we assume that each terminal has a Bernoulli random variable with parameter p_{w_i}, so all messages $\mathbf{x}_{\mathcal{I}_i}$, of terminal w_i, are updated according to it; i.e., $p_{w_i} = p_k$, $\forall k \in \mathcal{I}_i$, and $\psi_k^{(l)} = \psi_{w_i}^{(l)}$, $\forall k \in \mathcal{I}_i$, with $\psi_{w_i}^{(l)}$ being the sample drawn from the random variable of w_i at iteration l.

3.2 Gaussian Belief Propagation under High-Order Factorization

In this section we are going to present some results from [3, 4]. In simple words, among other things, they show that, with proper initialization, the convergence of Gaussian Belief Propagation (GBP), under high-order factorization and asynchronous scheduling, comes down to the convergence of Eq. 3.4.

Let $\mathbf{x} \in \mathbb{R}^n$ be a Gaussian vector. Its probability density function (PDF) is

$$p(\mathbf{x}) \propto \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal \mathbf{\Lambda}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \tag{3.5}$$

where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}]$ its expected value and $\mathbf{\Lambda} = \mathbb{E}[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\intercal]$ its covariance matrix. We can also write the PDF of $\mathbf{x}$ in information form, namely:

$$\begin{aligned} p(\mathbf{x}) &\propto \exp\left\{-\frac{1}{2}\left(\mathbf{x}^\intercal\mathbf{\Lambda}^{-1}\mathbf{x} - \mathbf{x}^\intercal\mathbf{\Lambda}^{-1}\boldsymbol{\mu} - \boldsymbol{\mu}^\intercal\mathbf{\Lambda}^{-1}\mathbf{x} + \boldsymbol{\mu}^\intercal\mathbf{\Lambda}^{-1}\boldsymbol{\mu}\right)\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left(\mathbf{x}^\intercal\mathbf{\Lambda}^{-1}\mathbf{x} - 2(\mathbf{\Lambda}^{-1}\boldsymbol{\mu})^\intercal\mathbf{x}\right)\right\} \\ &\propto \exp\left\{-\frac{1}{2}\mathbf{x}^\intercal\mathbf{\Lambda}^{-1}\mathbf{x} + (\mathbf{\Lambda}^{-1}\boldsymbol{\mu})^\intercal\mathbf{x}\right\} \\ &\propto \exp\left\{-\frac{1}{2}\mathbf{x}^\intercal\mathbf{J}\mathbf{x} + \mathbf{h}^\intercal\mathbf{x}\right\}, \end{aligned} \tag{3.6}$$

denoted $\mathbf{x} \sim \mathcal{N}^{-1}(\mathbf{h}, \mathbf{J})$, with $\mathbf{J} = \mathbf{\Lambda}^{-1} \succ \mathbf{O}$ is the information matrix and $\mathbf{h} = \mathbf{J}\boldsymbol{\mu}$ is the potential vector. This form is preferred when dealing with Gaussian graphical models because the non zero elements of $\mathbf{J}$ indicate how the different nodes are connected [2].

Consider a factor graph that encodes the above probability density function, then we can write

$$p(\mathbf{x}) = \prod_{i=1}^{n} f_i(x_i) \prod_{j=1}^{m} g_j(\mathcal{X}_j), \tag{3.7}$$

where $f_i(x_i)$ are functions of the variable nodes x_i, $\forall i \in \{1, 2, \ldots, n\}$, and $g_j(\mathcal{X}_j)$ functions of the sets $\mathcal{X}_j \subseteq \{x_1, x_2, \ldots, x_n\}$, $\forall j \in \{1, 2, \ldots, m\}$, which contain the variable nodes connected to each factor node. When at least one factor has more than two variable nodes connected to

it; i.e., $|\mathcal{X}_j| > 2$, then we refer to Eq. 3.7 as a *high-order factorization* of the joint Gaussian probability density function. When $|\mathcal{X}_j| \leq 2$, $\forall j$, then we have a pairwise factorization.

In [3] specific factorizations of the information matrix and potential vector are considered, namely: $\mathbf{J} = \mathbf{\Lambda} + \mathbf{\Xi}^\intercal \mathbf{\Sigma} \mathbf{\Xi}$ and $\mathbf{h} = \mathbf{\Lambda}\boldsymbol{\xi} + \mathbf{\Xi}^\intercal \mathbf{\Sigma} \mathbf{u}$, where $\mathbf{\Lambda} \triangleq \text{diag}\{\eta_1, \eta_2, \ldots, \eta_n\}$, $\mathbf{\Sigma} \triangleq \text{diag}\{\zeta_1, \zeta_2, \ldots, \zeta_m\}$, with $\eta_i \geq 0$, $\forall i \in \{1, 2, \ldots, n\}$, $\zeta_j > 0$, $\forall j \in \{1, 2, \ldots, m\}$, $\mathbf{\Xi} \in \mathbb{R}^{m \times n}$, $\boldsymbol{\xi} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$. Now we can rewrite Eq. 3.6 as

$$\begin{aligned} p(\mathbf{x}) &\propto \exp\left\{-\frac{1}{2}\mathbf{x}^\intercal \left(\mathbf{\Lambda} + \mathbf{\Xi}^\intercal \mathbf{\Sigma} \mathbf{\Xi}\right) \mathbf{x} + \left(\mathbf{\Lambda}\boldsymbol{\xi} + \mathbf{\Xi}^\intercal \mathbf{\Sigma} \mathbf{u}\right)^\intercal \mathbf{x}\right\} \\ &\propto \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\xi})^\intercal \mathbf{\Lambda} (\mathbf{x} - \boldsymbol{\xi})\right\} \\ &\quad \times \exp\left\{-\frac{1}{2}(\mathbf{\Xi}\mathbf{x} - \mathbf{u})^\intercal \mathbf{\Sigma} (\mathbf{\Xi}\mathbf{x} - \mathbf{u})\right\} \\ &\propto \prod_{i=1}^{n} \exp\left\{-\frac{1}{2}\eta_i \left(x_i - \xi_i\right)^2\right\} \\ &\quad \times \prod_{j=1}^{m} \exp\left\{-\frac{1}{2}\zeta_j \left(\mathbf{\Xi}_{j:}\mathbf{x} - u_j\right)^2\right\}. \end{aligned} \tag{3.8}$$

where $\mathbf{\Xi}_{j:}$ denotes the j-th row of the matrix $\mathbf{\Xi}$. Comparing Eq. 3.8 to 3.7 we obtain:

$$f_i(x_i) \propto \exp\left\{-\frac{1}{2}\eta_i (x_i - \xi_i)^2\right\}, \tag{3.9}$$

$$g_j(\mathcal{X}_j) \propto \exp\left\{-\frac{1}{2}\zeta_j \left(\sum_{k \in \mathcal{V}_j} \Xi_{jk} x_k - u_j\right)^2\right\}. \tag{3.10}$$

where $\mathcal{X}_j \triangleq \{x_k | \Xi_{jk} \neq 0\}$ and $\mathcal{V}_j \triangleq \{k | x_k \in \mathcal{X}_j\}$. In a factor graph there are two types of messages: factor-to-variable and variable-to-factor. Additionally, we know that Belief Propagation on Gaussian graphical models yields messages that follow Gaussian distributions [2], which can be characterized completely with two values: the mean and variance.

With $m^{(l)}_{g_j \to x_i}(x_i)$ we denote the message sent from factor g_j to random variable x_i and $m^{(l)}_{x_i \to g_j}(x_i)$ the one sent from random variable x_i to factor g_j, both at iteration l. Those messages are computed as

$$m^{(l)}_{g_j \to x_i}(x_i) \propto \int_{-\infty}^{+\infty} g_j(\mathcal{X}_j) \prod_{k \in \mathcal{V}_j \setminus i} m^{(l-1)}_{x_k \to g_j}(x_k) \, d\mathcal{X}_j \setminus x_i, \tag{3.11}$$

$$m^{(l)}_{x_i \to g_j}(x_i) \propto f_i(x_i) \prod_{k \in \mathcal{G}_i \setminus j} m^{(l)}_{g_k \to x_i}(x_i), \tag{3.12}$$

where $\mathcal{G}_i$ denotes the indices of factors g_j connected directly to random variable x_i in the factor graph. Substituting Eq. 3.12 into Eq. 3.11, the expression of factor-to-variable messages becomes

$$m_{g_j\to x_i}^{(l)}(x_i) \propto \int_{-\infty}^{+\infty} g_j(\mathcal{X}_j) \prod_{k\in\mathcal{V}_j\backslash i} \left(f_k(x_k) \prod_{k'\in\mathcal{G}_k\backslash j} m_{g_{k'}\to x_k}^{(l)}(x_k) \right) d\mathcal{X}_j\backslash x_i. \tag{3.13}$$

Eq. 3.13 dictates that any factor-to-variable message can be written as a function of other factor-to-variable messages, thus we have eliminated the need of variable-to-factor messages. Without loss of generality, we assume that $m_{g_{k'}\to x_k}^{(l-1)}(x_k) \sim \mathcal{N}\left(x_k; \mu_{g_{k'}\to x_k}^{(l-1)}, \frac{1}{v_{g_{k'}\to x_k}^{(l-1)}}\right)$, with $\mu_{g_{k'}\to x_k}^{(l-1)}$ and $v_{g_{k'}\to x_k}$ their mean and precision, respectively. Substituting the expression for $f_i(x_i)$ and $g_j(\mathcal{X}_j)$ from Eq. 3.9 and 3.10, into Eq. 3.13 we get

$$\begin{aligned} m_{g_j\to x_i}^{(l)}(x_i) \propto & \int_{-\infty}^{+\infty} \exp\left\{ -\frac{1}{2}\zeta_j \left(\sum_{k\in\mathcal{V}_j} \Xi_{jk} x_k - u_j \right)^2 \right\} \\ & \times \prod_{k\in\mathcal{V}_j\backslash i} \exp\left\{ -\frac{1}{2} \left(\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} \nu_{g_{k'}\to x_k}^{(l-1)} \right) x_k^2 + \left(\eta_k \xi_k + \sum_{k'\in\mathcal{G}_k\backslash j} \nu_{g_{k'}\to x_k}^{(l-1)} \mu_{g_{k'}\to x_k}^{(l-1)} \right) x_k \right\} d\mathcal{X}_j\backslash x_i. \end{aligned} \tag{3.14}$$

The messages $m_{g_j\to x_i}(x_i)$ follow Gaussian distribution [3] with their mean and precision given by

$$\mu_{g_j\to x_i}^{(l)} = \begin{cases} \Xi_{ji}^{-1} u_j - \sum_{k\in\mathcal{V}_j\backslash i} \frac{\Xi_{ji}^{-1}\Xi_{jk}\left(\eta_k\xi_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)} \mu_{g_{k'}\to x_k}^{(l-1)}\right)}{\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}}, \\ \qquad\qquad \text{if } \eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)} > 0, \forall k \in \mathcal{V}_j\backslash i, \\ 0, \quad \text{otherwise} \end{cases} \tag{3.15}$$

$$v_{g_j\to x_i}^{(l)} = \frac{\Xi_{ji}^2}{\zeta_j^{-1} + \sum_{k\in\mathcal{V}_j\backslash i} \Xi_{jk}^2 \left(\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}\right)^{-1}}, \tag{3.16}$$

respectively.

The belief of x_i at iteration l, $b^{(l)}(x_i)$, can be computed using the incoming factor-to-variable messages and the function $f_i(x_i)$ as follows:

$$b^{(l)}(x_i) \propto f_i(x_i) \prod_{k\in\mathcal{G}_i} m_{g_k\to x_i}^{(l)}(x_i). \tag{3.17}$$

Again, we can substitute the expressions of $f_i(x_i)$ and $m_{g_k\to x_i}^{(l)}(x_i)$ into Eq. 3.17 and obtain

$$b^{(l)}\left(x_i\right) \propto \exp\left\{-\frac{1}{2}\left(\eta_i + \sum_{k\in\mathcal{G}_i} v_{g_k\to x_i}^{(l)}\right)x_i^2 + \left(\eta_i\xi_i + \sum_{k\in\mathcal{G}_i} v_{g_k\to x_i}^{(l)}\mu_{g_k\to x_i}^{(l)}\right)x_i\right\}. \tag{3.18}$$

It is proven in [3] that $b^{(l)}(x_i)$ are valid Gaussians with $b^{(l)}(x_i) \sim \mathcal{N}\left(x_i; \epsilon_i^{(l)}, \sigma_i^{(l)}\right)$, where

$$\epsilon_i^{(l)} = \frac{\eta_i\xi_i + \sum_{k\in\mathcal{G}_i} v_{g_k\to x_i}^{(l)}\mu_{g_k\to x_i}^{(l)}}{\eta_i + \sum_{k\in\mathcal{G}_i} v_{g_k\to x_i}^{(l)}}, \tag{3.19}$$

$$\sigma_i^{(l)} = \frac{1}{\eta_i + \sum_{k\in\mathcal{G}_i} v_{g_k\to x_i}^{(l)}}. \tag{3.20}$$

With $\mathbf{v}^{(l)}$ and $\boldsymbol{\mu}^{(l)}$ we denote the vectors constructed if we stack all $v_{g_j\to x_i}^{(l)}$ and $\mu_{g_j\to x_i}^{(l)}$, $\forall(i,j)\in\mathcal{E} \triangleq \{(i,j)|i\in\{1,2,\ldots,n\}, j\in\mathcal{G}_i\}$, respectively, with the order of (i,j) ascending first on i and then on j. Additionally, we define $\Theta \triangleq \left\{\boldsymbol{\theta}|\theta_{g_j\to x_i} \geq \Xi_{ji}^2\zeta_j,\ \forall(i,j)\in\mathcal{E}\right\}$, where $\boldsymbol{\theta}$ is a vector containing the elements $\theta_{g_j\to x_i}$, $\forall(i,j)\in\mathcal{E}$ with the same order as in $\mathbf{v}^{(l)}$ and $\boldsymbol{\mu}^{(l)}$. The Belief Propagation (BP) message mean $\boldsymbol{\mu}^{(l)}$ can be rewritten in a vector form as

$$\boldsymbol{\mu}^{(l)} = \mathbf{A}\boldsymbol{\mu}^{(l-1)} + \mathbf{c}, \tag{3.21}$$

where $\mathbf{A}\in\mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$, such that $\mathbf{A}\boldsymbol{\mu}^{(l-1)}$ is a column vector containing elements α_{ij} for all $(i,j)\in\mathcal{E}$, arranged first on i and then on j, with

$$\alpha_{ij} = \begin{cases} -\sum_{k\in\mathcal{V}_j\backslash i} \frac{\Xi_{ji}^{-1}\Xi_{jk}\sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}\mu_{g_{k'}\to x_k}^{(l-1)}}{\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}}, & \text{if } \eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)} > 0 \text{ for all } k\in\mathcal{V}_j\backslash i, \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

and $\mathbf{c}$ is a column vector containing elements β_{ij} for all $(i,j)\in\mathcal{E}$ arranged first on i and then on j, with

$$\beta_{ij} = \begin{cases} \Xi_{ji}^{-1}u_j - \sum_{k\in\mathcal{V}_j\backslash i} \frac{\Xi_{ji}^{-1}\Xi_{jk}\eta_k\xi_k}{\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}}, & \text{if } \eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)} > 0 \text{ for all } k\in\mathcal{V}_j\backslash i, \\ 0, & \text{otherwise.} \end{cases} \tag{3.23}$$

In order to find the analytical expression for the elements of $\mathbf{A}$, observe that when $\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)} > 0$ for all $k\in\mathcal{V}_j\backslash i$

$$\alpha_{ij} = -\sum_{k\in\mathcal{V}_j\backslash i} \frac{\Xi_{ji}^{-1}\Xi_{jk}\sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}\mu_{g_{k'}\to x_k}^{(l-1)}}{\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}} = -\sum_{k\in\mathcal{V}_j\backslash i}\sum_{k'\in\mathcal{G}_k\backslash j} \underbrace{\frac{\Xi_{ji}^{-1}\Xi_{jk}v_{g_{k'}\to x_k}^{(l-1)}}{\eta_k + \sum_{k'\in\mathcal{G}_k\backslash j} v_{g_{k'}\to x_k}^{(l-1)}}}_{=\mathbf{A}_{ij,\,kk'}} \mu_{g_{k'}\to x_k}^{(l-1)}, \tag{3.24}$$

where $\mathbf{A}_{ij,\,kk'}$ is the element of $\mathbf{A} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$ in the (ij) row and (kk') column.

3.2.1 Asynchronous Scheduling

Under asynchronous scheduling, each Belief Propagation (BP) message may only be updated in a subset of iterations, rather than all iterations. Mathematically, let $\mathcal{L}_{g_j\to x_i}$ be the set with the updating times of the messages from g_j to x_i, and by $\tau_{g_j\to x_i}^{g_{k'}\to x_k}(l-1)$ denote the time index of the available message from $g_{k'} \to x_k$ during the update of the message from g_j to x_i at the l-th iteration. Then, at any iteration only one of the following can happen

(a) If $l \in \mathcal{L}_{g_j\to x_i}$, the message parameters $\mu_{g_j\to x_i}$ and $v_{g_j\to x_i}$ are updated according to equations 3.15 and 3.16, respectively, but with $l-1$ replaced by $\tau_{g_j\to x_i}^{g_{k'}\to x_k}(l-1)$.

(b) If $l \notin \mathcal{L}_{g_j\to x_i}$, the message parameters $\mu_{g_j\to x_i}^{(l)}$ and $v_{g_j\to x_i}^{(l)}$ maintain the corresponding values of the previous iteration $l-1$.

Obviously, when $\mathcal{L}_{g_j\to x_i} = \{1, 2, 3, \ldots\}$ and $\tau_{g_j\to x_i}^{g_{k'}\to x_k}(l-1) = l-1$, asynchronous scheduling reduces to synchronous.

Definition 1 (**Totally asynchronous scheduling** [3, 12])**.** The sets $\mathcal{L}_{g_j\to x_i}$ for all $(i,j) \in \mathcal{E}$ are infinite. Furthermore, if t_r is a sequence of elements of $\mathcal{L}_{g_j\to x_i}$ that tends to infinity, then $\lim_{r\to\infty} \tau_{g_j\to x_i}^{g_{k'}\to x_k}(t_r) = \infty$ for all $k \in \mathcal{V}_j\backslash i$ and $k' \in \mathcal{G}_k\backslash j$

The sets $\mathcal{L}_{g_j\to x_i}$ for all $(i,j) \in \mathcal{E}$ being infinite indicate that all messages are updated infinitely often, while the limit $\lim_{r\to\infty} \tau_{g_j\to x_i}^{g_{k'}\to x_k}(t_r) = \infty$, that old messages will be eventually purged out.

Remark 2. This is the most general form of asynchronous scheduling.

As shown in [3], if $\mathbf{v}^{(0)} \in \Theta$, then $\mathbf{v}^{(l)}$ converges to a unique non-negative fixed point $\mathbf{v}^*$, under totally asynchronous scheduling. Once the precision $\mathbf{v}^{(l)}$ converges, the update of Belief Propagation message means follows the expression Eq. 3.21, with the values of $v_{g_{k'}\to x_k}^{(l-1)}$ being replaced by the converged values $v_{g_{k'}\to x_k}^{*}$. Then, Eq. 3.21 converges under totally asynchronous scheduling if

$$\rho(|\mathbf{A}|) < 1. \tag{3.25}$$

Under totally asynchronous scheduling, the convergence condition of the BP message variances is necessary and sufficient, while the convergence condition of BP message means is only sufficient. Now we consider a subclass of totally asynchronous scheduling, the *non-i.i.d. asynchronous scheduling* (introduced in Sect. 3.1), for which necessary and sufficient conditions, for the convergence of the expectation of BP message means, exist.

Definition 2 (**non-i.i.d. asynchronous scheduling**)**.** This scheduling is a subclass of totally asynchronous scheduling with $\tau_{g_j\to x_i}^{g_{k'}\to x_k}(l-1) = l-1$. Furthermore, the elements in $\mathcal{L}_{g_j\to x_i}$ are determined by $\psi_{g_j\to x_i}^{(l)} \in \{0,1\}$, with $l \in \mathcal{L}_{g_j\to x_i}$ if $\psi_{g_j\to x_i}^{(l)} = 1$. The $\psi_{g_j\to x_i}^{(l)}$ for

all $(i,j) \in \mathcal{E}$ and $l \geq 1$ are Bernoulli random variables, independent across the different iterations l but possibly dependent and not identically distributed across different messages, with $Pr(\psi_{g_j \to x_i}^{(l)} = 1) = p_{g_j \to x_i}$ and $Pr(\psi_{g_j \to x_i}^{(l)} = 0) = 1 - p_{g_j \to x_i}$, and $p_{g_j \to x_i} \in (0, 1]$.

We utilize Eq. 3.4 and express the fixed point problem of Eq. 3.21 as

$$\begin{aligned} \boldsymbol{\mu}^{(l)} &= \boldsymbol{\Psi}^{(l)} \left(\mathbf{A}\boldsymbol{\mu}^{(l-1)} + \mathbf{c} \right) + \left(\mathbf{I} - \boldsymbol{\Psi}^{(l)} \right) \boldsymbol{\mu}^{(l-1)} \\ &= \left(\boldsymbol{\Psi}^{(l)} \mathbf{A} + \mathbf{I} - \boldsymbol{\Psi}^{(l)} \right) \boldsymbol{\mu}^{(l-1)} + \boldsymbol{\Psi}^{(l)} \mathbf{c}. \end{aligned} \tag{3.26}$$

Theorem 1 ([5])**.** With high-order decomposition and $\mathbf{v}^{(0)} \in \Theta$, the expectation of Belief Propagation message mean $\mathbb{E}[\boldsymbol{\mu}^{(l)}]$ converges under the non-i.i.d. asynchronous scheduling if and only if $\rho\left(\mathbf{P}\left(\mathbf{A} - \mathbf{I}\right) + \mathbf{I}\right) < 1$, where $\mathbb{E}[\boldsymbol{\Psi}^{(l)}] \triangleq \mathbf{P}$, $\forall l$.

Notice that the notion of convergence in the non-i.i.d. asynchronous scheduling is more relaxed when compared to the totally asynchronous. The reason for this is that, in the first case, we are interested in the average behavior; i.e., there may be some experiments where BP diverges, even though the corresponding convergence conditions are satisfied, but the average of multiple experiments will converge.

The asynchronous scheduling can be utilized to solve problems for which the synchronous variant of Gaussian BP under high-order factorization diverges [3, 4]. Apparently, there are also instances for which the synchronous variant converges and the asynchronous diverges. We also note that due to Theorem 1, there exist problems that can be solved with non-i.i.d. asynchronous scheduling and not with totally asynchronous scheduling, since we may encounter a matrix $\mathbf{A}$ with $\rho(|\mathbf{A}|) > 1$ and, by carefully selecting the probability vector $\mathbf{p}$, we may have $\rho\left(\mathbf{P}\left(\mathbf{A} - \mathbf{I}\right) + \mathbf{I}\right) < 1$.

3.2.2 Solving a Linear System of Equations

A particularly interesting application of Gaussian Belief Propagation is that of solving a system of linear equations [3, 13]. We want to solve

$$\mathbf{M}\mathbf{x} = \mathbf{s}, \tag{3.27}$$

where $\mathbf{M} \in \mathbb{R}^{m \times n}$ with $m \geq n$ a full rank matrix and $\mathbf{s} \in \mathbb{R}^m$ a real vector. The least-squares solution is obtained by

$$\mathbf{x} = (\mathbf{M}^\intercal \mathbf{M})^{-1} \mathbf{M}^\intercal \mathbf{s}. \tag{3.28}$$

If we set $\boldsymbol{\Lambda} = \mathbf{O}$, $\boldsymbol{\Xi} = \mathbf{M}$, $\boldsymbol{\Sigma} = \mathbf{I}$, $\boldsymbol{\xi} = \mathbf{0}$ and $\mathbf{u} = \mathbf{s}$ in Eq. 3.8, then Gaussian Belief Propagation can be utilized for the following distribution

$$
\begin{aligned}
p(\mathbf{x}) &= \mathcal{N}\left(\mathbf{x}; \left(\mathbf{M}^\intercal\mathbf{M}\right)^{-1}\mathbf{M}^\intercal\mathbf{s}, \left(\mathbf{M}^\intercal\mathbf{M}\right)^{-1}\right) \\
&\propto \exp\left\{-\frac{1}{2}\mathbf{x}^\intercal\mathbf{M}^\intercal\mathbf{M}\mathbf{x} + \mathbf{s}^\intercal\mathbf{M}\mathbf{x}\right\}.
\end{aligned} \tag{3.29}
$$

whose inference of expected value will yield the desired solution.

3.3 Transmission Delays and Asynchronous Scheduling

Transmission delays are inevitable when considering a WSN and must be taken into account when studying an algorithm. In this section we show that they can affect the convergence of Gaussian Belief Propagation under high-order factorization and non-i.i.d. asynchronous scheduling.

Recall that in the non-i.i.d. asynchronous scheduling, at iteration l, every terminal samples its Bernoulli random variable $\psi_{w_i}^{(l)}$ with parameter p_{w_i}, and based on the outcome, decides whether to transmit or not. Additionally, it is assumed that the same most up-to-date messages of the previous iteration are available at all the nodes (Eq. 3.26), namely $\tau_{g_j \to x_i}^{g_{k'} \to g_k}(l-1) = l-1$, for all the messages $g_j \to x_i$ and $g_{k'} \to g_k$. This assumption in a WSN setting cannot be satisfied without some modifications to the algorithm. A terminal does not know if its neighbors have decided to transit at a given iteration, unless it receives their messages. Moreover, due to the stohastic nature of the wireless channel, messages have different transmission delays, thus it's challenging for the terminals, which operate with local information, to know when all the messages of an iteration have been transmitted. In other words, a terminal does not know when it's the right time to move on with the next iteration, and as a result the scheduling is messed up.

To better understand the above we provide a simple example. Assume that we have a WSN with three terminals w_1, w_2, w_3 and that each terminal sends messages m_{w_1}, m_{w_2} and m_{w_3}, respectively. Let $\mathcal{T}_1 = \{1,1,0\}$, $\mathcal{T}_2 = \{1,0,1\}$ and $\mathcal{T}_3 = \{1,0,1\}$ be the transmission sequence, generated by the sampling of the Bernoulli random variables, for w_1, w_2 and w_3, respectively. As you can see in the lower diagram of Fig. 3.1, due to different transmission delays, at time t_1 the message m_{w_1} of the first iteration is still being transmitted, whereas m_{w_2} and m_{w_3} have already been updated twice without the most up-to-date values of m_{w_1}.

Now we provide numerical results which verify that Gaussian Belief Propagation, under high-order factorization and non-i.i.d. asynchronous scheduling, diverges because of the transmission delays. We use the algorithm to solve the system of linear equations, as described in Sect. 3.2.2, with

$$
\mathbf{M} = \begin{bmatrix} 0.9448 & 0.3377 & 0.1112 \\ 0.4909 & 0.9001 & 0.7803 \\ 0.4893 & 0.3692 & 0.3897 \end{bmatrix} \tag{3.30}
$$

and $\mathbf{s} = [1\ 1\ 1]^\intercal$.

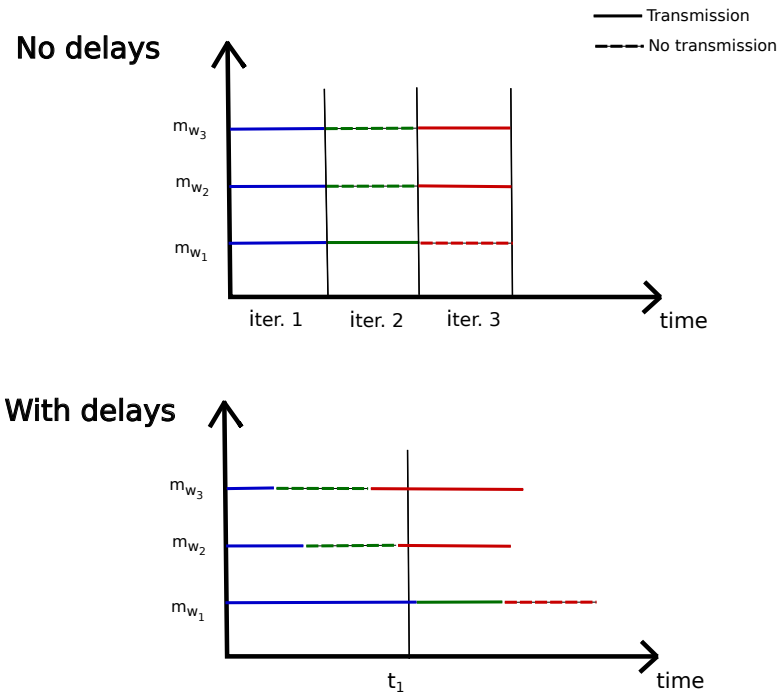


Figure 3.1: Message updates with and without transmission delays.

The factor graph stems from the matrix $\mathbf{M}$. Its rows correspond to factor nodes, its columns to variables and its non-zero elements dictate how the different nodes are connected. Because $\mathbf{M}$ does not have zero elements, we have a complete bipartite graph with three variable nodes and equal factor nodes.

As we described in Sect. 3.2, at iteration l, the message means $\mu_{g_j \to x_i}$ of Gaussian Belief Propagation, under non-i.i.d. asynchronous scheduling, are updated linearly, according to Eq. 3.26. We distribute the computations by assigning the elements of $\boldsymbol{\mu}^{(l)}$ to different WSN terminals and update them as discussed in Sect. 3.1.

We assume that we have a WSN with three terminals w_1, w_2 and w_3. The terminal w_1 has the necessary information for the local update of the message means $\{\mu^{(l)}_{g_1 \to x_1}, \mu^{(l)}_{g_1 \to x_2}, \mu^{(l)}_{g_1 \to x_3}\}$, w_2 is responsible for $\{\mu^{(l)}_{g_2 \to x_1}, \mu^{(l)}_{g_2 \to x_2}, \mu^{(l)}_{g_2 \to x_3}\}$, and w_3 for $\{\mu^{(l)}_{g_3 \to x_1}, \mu^{(l)}_{g_3 \to x_2}, \mu^{(l)}_{g_3 \to x_3}\}$. The parameters for the Bernoulli random variables are $p_{w_1} = 0.5$, $p_{w_2} = 0.3$ and $p_{w_3} = 0.3$. Every 1 second the terminals update their local values using the most recent messages received from their neighbors and they transmit; we assume that the computations happen instantly. In Fig. 3.2 you can see the transmission delays that we consider.

We initialize the Gaussian Belief Propagation properly ($\mathbf{v}^{(0)} \in \Theta$) and we update the precision vector $\mathbf{v}^{(l)}$ until it converges to $\mathbf{v}^*$; we assume that the terminals know the corresponding values of $\mathbf{v}^*$, hence are only concerned with the convergence of message means $\boldsymbol{\mu}^{(l)}$. We note that $\rho(|\mathbf{A}|) = 1.3044 > 1$ but $\rho\left(\mathbf{P}\left(\mathbf{A} - \mathbf{I}\right) + \mathbf{I}\right) = 0.9461 < 1$, thus from Theorem 1, we know that by taking the average of multiple experiments we will get the desired solution, which can not be obtained under totally asynchronous scheduling.

In Fig. 3.3 we present the results of the algorithm when transmission delays are considered

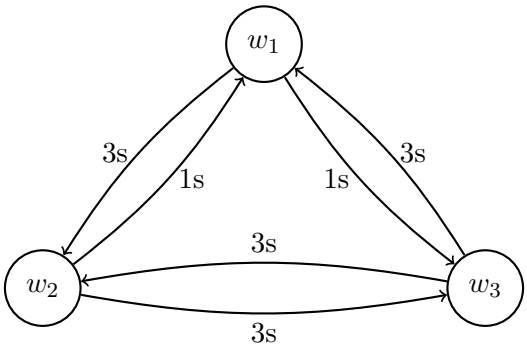


Figure 3.2: Transmission delays in the WSN.

and when they are not. In particular, we plot the per iteration expected value $\mathbb{E}[\|\hat{\mathbf{x}} - \mathbf{x}_*\|_2]$ using 20 independent experiments,[1] with $\hat{\mathbf{x}}$ being the vector that is constructed if we stack all belief means (Eq 3.19), and $\mathbf{x}_* = (\mathbf{M}^\intercal\mathbf{M})^{-1}\mathbf{M}^\intercal\mathbf{s}$, the least squares solution of Eq. 3.27.

As Fig. 3.1 shows, in theory we expect the algorithm to converge under non-i.i.d. asynchronous scheduling, but when implementing it on a WSN, the induced transmission delays can cause the algorithm to diverge.

Remark 3. The totally asynchronous scheduling is not affected by transmission delays, provided that they are bounded [12].

The above remark can be easily verified by looking at the definition of totally asynchronous scheduling, which is provided in Sect. 3.2.1 (it does not assume $\tau_{g_j \to x_i}^{g_{k'} \to x_k}(l-1) = l-1$). Moreover, we verify this through simulations. Specifically, we repeat the same experiment as before[2] but with

$$\mathbf{M} = \begin{bmatrix} 0.3383 & 0.4447 & 0.4530 \\ 0.0852 & 0.8936 & 0.0141 \\ 0.8949 & 0.5272 & 0.2678 \end{bmatrix} \tag{3.31}$$

which yields $\rho(|\mathbf{A}|) = 0.5949 < 1$; i.e., the message means converge under totally asynchronous scheduling and consequently they also converge under non-i.i.d. asynchronous scheduling.

In Fig. 3.4 we plot the per iteration distance of the belief means from their optimal values $\|\hat{\mathbf{x}} - \mathbf{x}_*\|_2$. We note that in totally asynchronous scheduling the convergence is guaranteed for every experiment (Sect. 3.2.1). As we expected, the algorithm does converge despite the transmission delays. An interesting observation here is that the induced delays increase the number of iterations needed for convergence, compared to the case in which messages arrive instantly.

In summary, for problems where $\rho(|\mathbf{A}|) < 1$, the transmission delays can not cause the algorithm to diverge but they may increase the number of iterations required. When $\rho(|\mathbf{A}|) > 1$

[1] The same transmission sequences, generated in each experiment, were used for the two examined cases.
[2] Recall that non-i.i.d. asynchronous scheduling is a subclass of totally asynchronous when $p_k > 0$, $\forall k$.

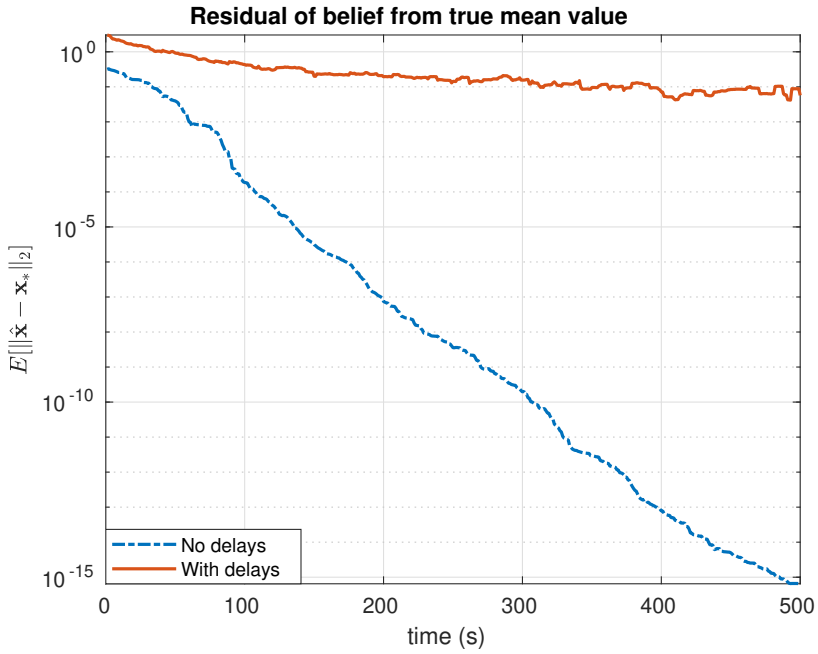


Figure 3.3: Gaussian Belief Propagation under non-i.i.d asynchronous scheduling, with $\rho(|\mathbf{A}|) = 1.3044$ and $\rho\left(\mathbf{P}\left(\mathbf{A}-\mathbf{I}\right)+\mathbf{I}\right) = 0.9461$.

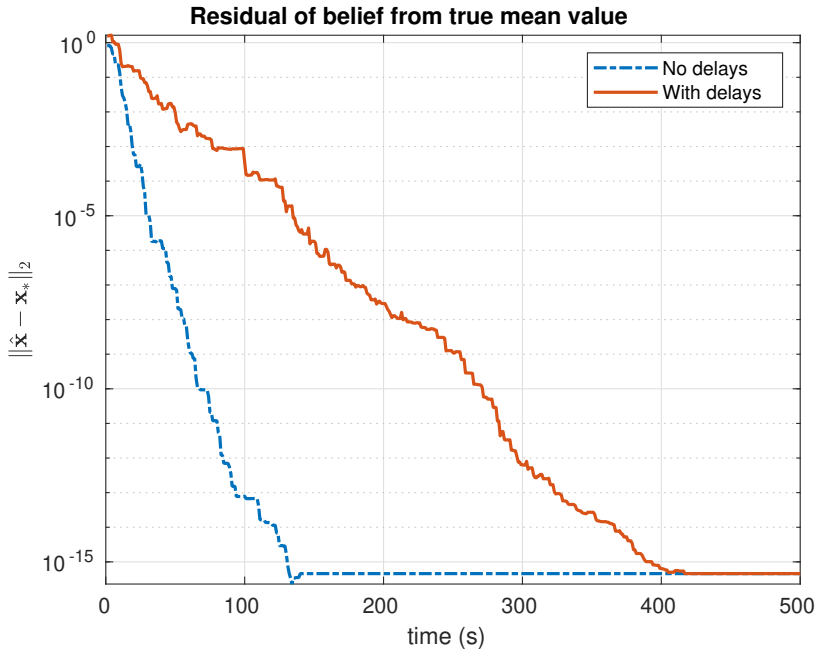


Figure 3.4: Gaussian Belief Propagation under non-i.i.d asynchronous scheduling, with $\rho(|\mathbf{A}|) = 0.5949$ and $\rho\left(\mathbf{P}\left(\mathbf{A}-\mathbf{I}\right)+\mathbf{I}\right) = 0.8145$.

but $\rho\left(\mathbf{P}\left(\mathbf{A}-\mathbf{I}\right)+\mathbf{I}\right)<1$, the algorithm can not always be utilized directly, without modifications, by a WSN.

3.4 Communication Model

Terminals exchange messages over wireless links, so we need to specify the communication model. We will consider wireless channels which are limited in energy and bandwidth, as well as subject to noise and interference.

We assume that there are R resource blocks (RBs), i.e. frequency bins, each of bandwidth B. With F_c^r we denote the carrier frequency of the RB r. Additionally, we consider frequency flat fading channels, that is, intersymbol interference does not occur, with Gaussian noise and interference which is caused by an outside system. During the propagation of an electromagnetic signal two phenomena can be observed, the large-scale and small-scale propagation effects [14, 15].

The large-scale propagation effects are determined mainly by the distance between the transmitter and the receiver, and by the morphology of the propagation environment. We can assume that these effects vary slowly in time. In this work we use the simplified Path-Loss model of [14] (Sect. 2.6). Specifically, when a terminal w_i transmits to a terminal w_j over the RB r, with the transmitted signal having power P_t and the received P_r:

$$\frac{P_r}{P_t} \triangleq L_{ij}^r = \left(\frac{\lambda_r}{4\pi d_0}\right)^2 \left(\frac{d_0}{d_{ij}}\right)^\gamma, \tag{3.32}$$

where $\lambda_r = c/F_c^r$, is the wavelength of the signal, d_{ij} the distance between the terminals w_i and w_j, d_0 a reference distance (assumed to be $1-10$m indoors and $10-100$m outdoors), and γ the path-loss exponent which is estimated empirically. We will assume $\gamma = 2$:

$$L_{ij}^r = \left(\frac{\lambda_r}{4\pi d_{ij}}\right)^2. \tag{3.33}$$

The small-scale propagation effects encapsulate the fast changes of the wireless channel, which may be caused by the movement of the transmitter, the receiver or the constructive and destructive addition of multipath signal components. In practice, deterministic channel models are rarely available and so we characterize multipath channels statistically. To that end, we provide the definition of *circularly-symmetric Gaussian* random variables.

Let X_k and X_l be independent Gaussian random variables with zero mean and variance $\frac{\sigma^2}{2}$, $X_k, X_l \sim \mathcal{N}(0, \frac{\sigma^2}{2})$. Then, the complex random variable $X = X_k + jX_l$ is circularly-symmetric Gaussian, $X \sim \mathcal{CN}(0, \sigma^2)$. Its probability density function is given by:

$$p_X(x) = p_X(x_k + jx_l) = p_{X_k}(x_k)p_{X_l}(x_l) = \frac{1}{\pi\sigma^2} e^{-\frac{|x|^2}{\sigma^2}}. \tag{3.34}$$

It can be shown that the real random variable $|X| = \sqrt{X_k^2 + X_l^2}$ follows Rayleigh distribution:

$$p_{|X|}(x) = \frac{2x}{\sigma^2} e^{-\frac{x^2}{\sigma^2}},\ x \geq 0, \tag{3.35}$$

and that $|X|^2 = X_k^2 + X_l^2$ follows exponential distribution with mean σ^2:

$$p_{|X|^2}(x) = \frac{1}{\sigma^2} e^{-\frac{x}{\sigma^2}},\ x \geq 0. \tag{3.36}$$

We can write $X = |X|e^{j\Phi}$, with $\Phi \triangleq \arctan \frac{X_l}{X_k}$. It can be shown that $|X|$ and Φ are independent random variables, with $|X|$ following Rayleigh distribution and $\Phi \sim \mathcal{U}[0, 2\pi)$.

Assume that terminal w_i transmits to terminal w_j. We model the multipath channel by a random time-varying impulse response. If a single pulse is transmitted, the received signal will appear as a pulse train, with each pulse in the train corresponding to a distinct multipath component. If we transmit a signal, the different paths induce phase shifts and change its amplitude; what is received is the superposition of all the different variants of the original signal. In the case of *Rayleigh fading*, which we consider in this work, each multipath component is modeled as an independent circularly-symmetric Gaussian random variable. Utilizing the Central Limit Theorem, the impulse response of the multipath channel, h_{ij}, follows circularly-symmetric Gaussian distribution $h_{ij} \sim \mathcal{CN}(0, \sigma^2)$, where $\sigma^2 \propto d_{ij}^{-2}$. Consequently, the phase shift of the received signal, Φ_{ij}, is a random variable uniformly distributed in $[0, 2\pi)$ and the amplitude change, $|h_{ij}|$, follows Rayleigh distribution. Let $s_{ij}(t)$ be the transmitted signal, and $z_{ij}^r(t)$ be the corresponding channel output when RB r is used, then

$$z_{ij}^r(t) = \widetilde{h}_{ij}^r\, s_{ij}(t) + w_{ij} + v_r(t), \tag{3.37}$$

where $\widetilde{h}_{ij}^r = \sqrt{L_{ij}^r} h_{ij}$ is the channel gain when the RB r is used, $w_{ij}(t)$ is additive white Gaussian noise with power spectral density N_0, $v_r(t)$ is the interference over RB r whose energy is I_r, and the channel input $s_{ij}(t)$ is constrained to maximal transmit power of P.

The data rate is given by:

$$C_{ij}^r = B \log_2 \left(1 + \frac{P|\widetilde{h}_{ij}^r|^2}{I_r + BN_0}\right) \quad \text{bit/sec} \tag{3.38}$$

$$= B \log_2 \left(1 + \frac{PL_{ij}^r|h_{ij}|^2}{I_r + BN_0}\right) \quad \text{bit/sec.} \tag{3.39}$$

Let M_{ij} denote the length (in bits) of the message. The transmission time is given by:

$$d_{ij}^r = \frac{M_{ij}}{C_{ij}^r} = \frac{L_{ij}}{B \log_2 \left(1 + \frac{PL_{ij}^r|h_{ij}|^2}{I_r + BN_0}\right)} \quad \text{sec.} \tag{3.40}$$

We note that when terminal w_i is active, it transmits to all of its neighbors sequentially, using the same RB. Denoting with $N(i)$ the set of terminals adjacent to w_i, the total transmission time required by the terminal is

$$d_i^r = \sum_{j \in N(i)} d_{ij}^r. \tag{3.41}$$

3.5 The Coordinator Scheme

In order to enable a WSN to use the Gaussian Belief Propagation algorithm under non-i.i.d. asynchronous scheduling, so that transmission delays do not affect its convergence, we suggest that a "synchronization" mechanism is utilized. The main idea is that a terminal will act as *coordinator*, gathering some extra information from the others and orchestrating the execution of the algorithm. The proposed changes are the following:

- A terminal is selected that will act as coordinator. For convenience we refer to it as w_{cord}.
- Before the first iteration of the algorithm, each terminal w_i samples its Bernoulli random variables K times, where K is the total number of GBP iterations. This bit sequence is transmitted to w_{cord} along with the transmission delays, $d_i^r \ \ \forall 1 \leq r \leq R$.
- At the l-th iteration, w_{cord} examines the l-th element of every received bit sequence and determines which terminals should transmit. Additionally, using the transmission delays, allocates the RBs optimally (see Sect. 3.5.1) and informs the active terminals which RB to use.
- A terminal transmits its messages only if it's contacted by w_{cord}.
- When the channel conditions cause the transmission delays to change, the new values are sent to w_{cord}.

Based on the above, w_{cord} not only knows which terminals should transmit, but also the corresponding transmission delays. That is, w_{cord} knows when is the last message of an iteration received, thus can guarantee that all terminals have the most up-to-date messages of their neighbors before proceeding to the next iteration. In other words, the assumption of the non-i.i.d. asynchronous scheduling $\tau_{g_j \to x_i}^{g_{k'} \to x_k}(l-1) = l-1$, for all messages $g_j \to x_i$ and $g_{k'} \to x_k$, is satisfied.

We note that these changes affect neither the distributed nor the asynchronous characteristics of GBP. Even though w_{cord} has some extra information to process, the computations of the algorithm are performed locally and each terminal participates in a subset of the total iterations, as indicated by its Bernoulli random variable.

3.5.1 Optimal Resource Allocation

Consider a terminal, w_i, which is active at iteration l. The coordinator can allocate any of the R resources to w_i. Let $\chi_i^r \in \{0, 1\}$ be an allocation index with $\chi_i^r = 1$ implying that RB

r is allocated to w_i, otherwise $\chi_i^r = 0$, and $\boldsymbol{\chi}_i = [\chi_i^1, \chi_i^2, \ldots, \chi_i^R]$. Each terminal is allocated exactly one RB, hence $\boldsymbol{\chi}_i$ is a vector with a single nonzero component equal to one. Assuming that $\chi_i^k = 1$, $1 \leq k \leq R$, we can express the delay of w_i as

$$d_i^k = \sum_{r=1}^{R} \chi_i^r d_i^r. \tag{3.42}$$

Based on whether the number of active terminals is greater than that of available RBs, we devise two different integer optimization problems. By $\mathsf{N}^{(l)}$ we denote the set of active terminals at iteration l. When $R \geq |\mathsf{N}^{(l)}|$: each terminal allocates exactly one RB and each RB is allocated to at most one terminal. We want to minimize the total transmission delay, or bottleneck delay, at iteration l which is given by:

$$\max_{i \in \mathsf{N}^{(l)}} \sum_{r=1}^{R} \chi_i^r d_i^r, \tag{3.43}$$

therefore we can formulate the optimization problem as follows:

$$\min_{\boldsymbol{\chi}} \max_{i \in \mathsf{N}^{(l)}} \quad \sum_{r=1}^{R} \chi_i^r d_i^r \tag{3.44}$$

$$\text{s.t.} \quad \chi_i^r \in \{0, 1\}, \qquad \forall 1 \leq i \leq |\mathsf{N}^{(l)}|, \, \forall 1 \leq r \leq R \tag{3.45}$$

$$\sum_{i=1}^{|\mathsf{N}^{(l)}|} \chi_i^r \leq 1, \qquad \forall 1 \leq r \leq R \tag{3.46}$$

$$\sum_{r=1}^{R} \chi_i^r = 1, \qquad \forall 1 \leq i \leq |\mathsf{N}^{(l)}| \tag{3.47}$$

where $\boldsymbol{\chi} = [\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_{|\mathsf{N}^{(l)}|}]$. This is known as the Bottleneck Assignment Problem and we study it in Chap. 4.

When $R \leq |\mathsf{N}^{(l)}|$: each terminal allocates exactly one RB and each RB may be allocated to more than one terminals. In this case, apart from changing the constraint 3.46, we also need to alter the cost function. At any given time, at most one terminal can use a RB. If two or more are assigned the same RB, they use it sequentially in some order[3]. For instance, if RB r is allocated to w_1 and w_2, then w_2 may start using it once w_1 is done transmitting; the delay for w_1 is d_1^r but for w_2 is $d_1^r + d_2^r$, which can not be described by Eq. 3.44. In order to find the total transmission delay, we need to maximize across the RBs and not the terminals.

[3]e.g., the terminal with the lower index has higher priority

$$\min_{\boldsymbol{\chi}} \max_{1\leq r\leq R} \quad \sum_{i=1}^{|\mathsf{N}^{(l)}|} \chi_i^r d_i^r \tag{3.48}$$

$$\text{s.t.} \quad \chi_i^r \in \{0,1\}, \qquad \forall 1 \leq i \leq |\mathsf{N}^{(l)}|,\ \forall 1 \leq r \leq R \tag{3.49}$$

$$\sum_{i=1}^{|\mathsf{N}^{(l)}|} \chi_i^r \geq 0, \qquad \forall 1 \leq r \leq R \tag{3.50}$$

$$\sum_{r=1}^{R} \chi_i^r = 1, \qquad \forall 1 \leq i \leq |\mathsf{N}^{(l)}| \tag{3.51}$$

3.5.2 Simulations

In this section we evaluate the performance (convergence time) of the coordinator scheme through simulations. We utilize the GBP under high-order factorization and non-i.i.d. asynchronous scheduling in order to solve linear systems of equations. The communication model of Sec. 3.4 is used.

For the first system which we solve

$$\mathbf{M} = \begin{bmatrix} 3.63 & -6.12 & 0 & 0 & 0 & -2.61 & 0 & 0 \\ 0 & -10.65 & 7.59 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.92 & 7.05 & 0 & 0 & -10.46 & 0 \\ 0 & 0 & 0 & 0.18 & 3.27 & 0 & 0 & 0 \\ -2.01 & 0 & 0 & 0 & -0.97 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8.01 & 0.37 \\ 5.18 & -1.86 & 0 & 4.63 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.91 & 0 & 0 & 0 & 0 & 0.13 \end{bmatrix} \tag{3.52}$$

and

$$\mathbf{s} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^\intercal.$$

This example has also been studied in [3, 4] and is known that GBP under non-i.i.d. asynchronous scheduling can converge, provided that the values of $\mathbf{p}$ are carefully chosen. In Fig. 3.5 you can see the resulting factor graph and how the different parts of it are assigned to 4 WSN terminals. For

$$\mathbf{p} = [p_{w_1} \quad p_{w_2} \quad p_{w_3} \quad p_{w_4}]^\intercal = [0.6271 \quad 0.5314 \quad 0.7938 \quad 0.8454]^\intercal,$$

$\rho\left(\mathbf{P}\left(\mathbf{A}-\mathbf{I}\right)+\mathbf{I}\right) = 0.8985 < 1$, so the algorithm converges in the mean sense. We compute the converged message precision vector $\mathbf{v}^*$ and assume that the terminals know its corresponding values, hence are only concerned with the convergence of message means $\boldsymbol{\mu}^{(l)}$.

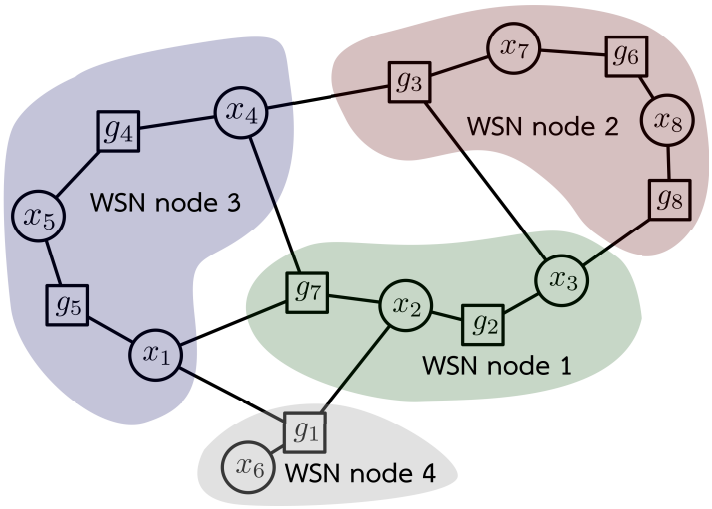


Figure 3.5: Factor graph assignment into a WSN with 4 nodes.

We have mentioned that we have only factor-to-variable messages and that each of those can be expressed as a function of other such messages. Specifically, from Eq. 3.15 we can write

$$\mu_{g_j \to x_i} = f(m_{g_{k'} \to x_k}) \quad \forall k \in \mathcal{V}_j \backslash i,\ \forall k' \in \mathcal{G}_k \backslash j. \tag{3.53}$$

Because of this relationship, when trying to find the messages that are sent among WSN terminals, looking at the connectivity of the factor graph can be misleading. For instance, observing Fig. 3.5 and thinking that only factor-to-variable messages exist, can lead someone to believe that no message is sent from w_1 to w_2. This is wrong. Notice that $\mu_{g_8 \to x_8} = f(m_{g_3 \to x_3}, m_{g_2 \to x_3})$, which means that w_1 should transmit the value of $m_{g_2 \to x_3}$ to w_2. Moreover, observe that $\mu_{g_3 \to x_7} = f(m_{g_4 \to x_4},\ m_{g_7 \to x_4},\ m_{g_2 \to x_3})$, thus $m_{g_7 \to x_4}$ should also be sent from w_1 to w_2, even though the variable x_4 belongs to w_3. In the following matrix the (i, j) component corresponds to the number of different GBP message values sent from w_i to w_j:

$$\mathsf{M}_{\text{WSN}} = \begin{bmatrix} 0 & 2 & 2 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \end{bmatrix} \tag{3.54}$$

In Fig. 3.6 we show how the WSN terminals are placed. Regarding the parameters of the communication model: we consider 2 RBs with $B = 60$MHz, $F_c^1 = 1800$MHz, $F_c^2 = 1860$MHz, $N_0 = -174$dBm, $I_r = [0.02 \quad 0.025]$mW, $P = 1$W and each GBP message has length 128 bit.

We select w_1 to act as coordinator. For the transmission of the bit sequence and delay values, w_2 and w_3 use RB 1 sequentially and RB 2 is used by w_4. Each delay value (d_i^r) consists of 64 bit and $K = 800$ (GBP iterations). w_{cord} contacts terminals using RB 1. The corresponding messages consist of 32 bit that indicate which RB should be used, and another 32 bit for the transmission priority, in case the RB must be shared. Every 10 GBP iterations, new channel estimations are obtained and transmitted to w_{cord} as described.

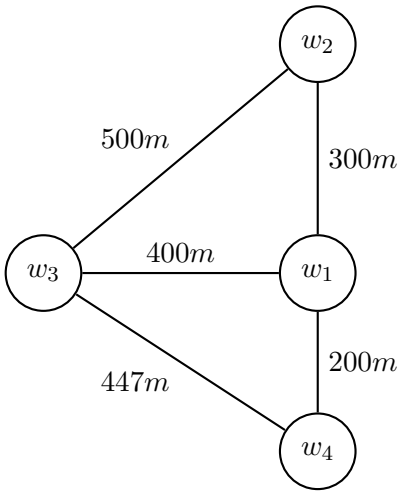


Figure 3.6: Placement of 4 WSN terminals.

We compare the coordinator scheme against two baselines, Random1 and Random2, in both of which we assume that the transmission delays do not affect the algorithm's convergence, that is, all terminals know exactly when is the last message of each iteration received. In Random1, at each iteration, the terminals select uniformly at random a RB to use. In Random2, the terminals select a RB, uniformly at random, only once, at the beginning of the experiment.

In Fig. 3.7 we plot the per iteration expected value $\mathbb{E}[\|\hat{\mathbf{x}} - \mathbf{x}_*\|_2]$ using 20 independent experiments, with $\hat{\mathbf{x}}$ being the vector that is constructed if we stack all belief means (Eq. 3.19) and $\mathbf{x}_* = (\mathbf{M}^\intercal\mathbf{M})^{-1}\mathbf{M}^\intercal\mathbf{s}$, the least squares solution of $\mathbf{M}\mathbf{x} = \mathbf{s}$.

In addition to the fact that the coordinator scheme allows WSNs to utilize GBP under non-i.i.d. asynchronous scheduling, it also improves the convergence time of the algorithm, as we can see clearly in Fig. 3.7. The latter is due to the optimal resource allocation.

Next, we consider an example with increased communication burden. Specifically

$$\mathbf{M} = \begin{bmatrix} 0.3001 & -0.0729 & -0.0530 & -0.0583 & -0.0044 & 0.0988 & -0.0927 & 0.0455 \\ -0.0933 & 0.4105 & -0.0416 & 0.0111 & -0.1659 & 0.0025 & -0.0207 & 0.0922 \\ -0.1273 & -0.2814 & 0.1817 & -0.2247 & -0.1084 & -0.2656 & -0.1417 & -0.1642 \\ -0.0283 & -0.1863 & -0.1090 & 0.2964 & -0.2808 & -0.4310 & -0.1124 & -0.1207 \\ -0.0219 & -0.0860 & 0.0116 & -0.0416 & 0.3797 & -0.1266 & -0.0217 & -0.0391 \\ -0.0112 & 0.0030 & 0.0654 & -0.0329 & 0.1350 & 0.6705 & -0.0365 & -0.0969 \\ 0.1092 & 0.2956 & 0.0996 & 0.2238 & 0.3042 & 0.3649 & 0.6328 & 0.2252 \\ 0.2532 & 0.2632 & 0.1026 & 0.2044 & 0.1944 & 0.0646 & 0.2214 & 0.5204 \end{bmatrix} \tag{3.55}$$

and

$$\mathbf{s} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^\intercal.$$

The resulting graphical model is a complete factor graph, with 8 variables and 8 factors. We

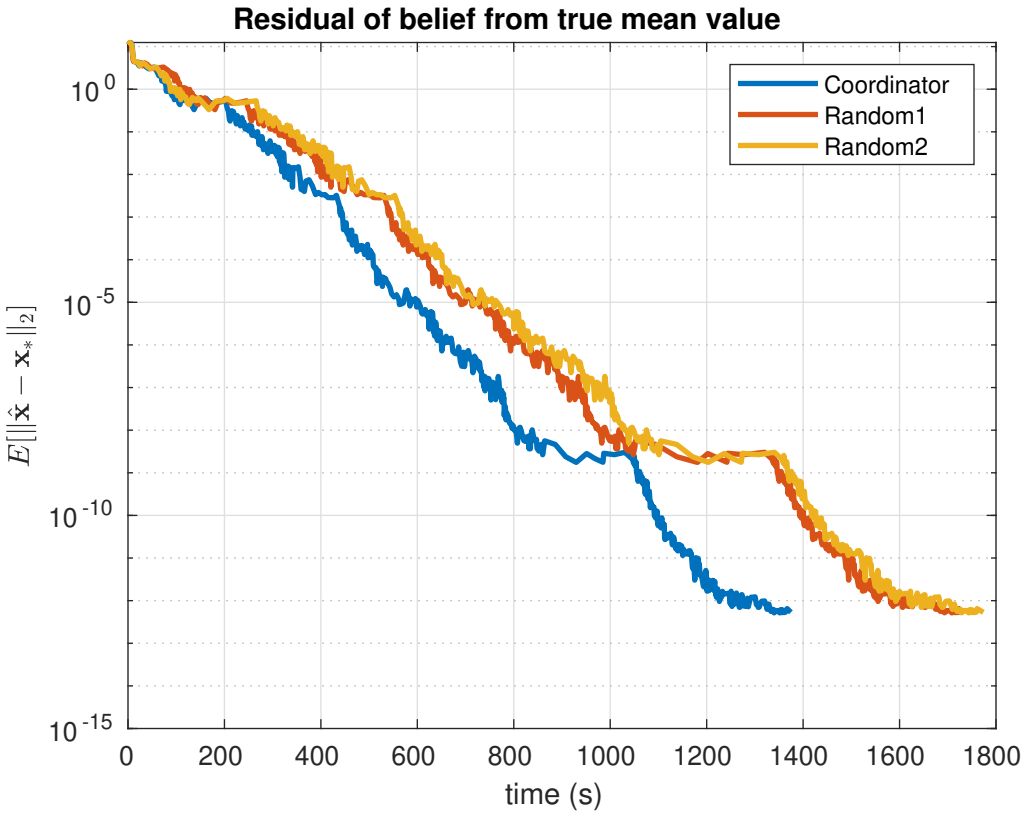


Figure 3.7: Convergence rate of GBP under non-i.i.d. asynchronous scheduling with 4 WSN terminals.

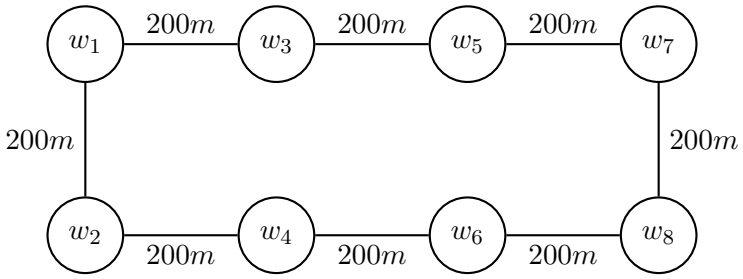


Figure 3.8: Placement of 8 WSN terminals.

consider a WSN with 8 terminals and each w_i is assigned the GBP messages sent from the i-th factor; i.e., $\{m_{g_i \to x_j} : 1 \leq j \leq 8\}$. With this assignment of terminals and because the factor graph is complete, every terminal needs to know all the other GBP messages in order to update all of its local values, so when w_i is active it transmits all its GBP messages to all the other terminals. That is,

$$\mathsf{M}_{\text{WSN}} = \begin{bmatrix} 0 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 0 & 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 0 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 0 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 0 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 0 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 & 0 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 0 \end{bmatrix} \tag{3.56}$$

Fig. 3.8 shows how the terminals are placed. For

$$\mathbf{p} = [0.1271 \quad 0.5314 \quad 0.6938 \quad 0.3454 \quad 0.3567 \quad 0.5896 \quad 0.4729 \quad 0.4234],$$

we have $\rho\left(\mathbf{P}\left(\mathbf{A}-\mathbf{I}\right)+\mathbf{I}\right) = 0.9473 < 1$, so the message means should converge. The parameters of the communication model are the same as before with the only difference being that now we have 3 RBs with $I_r = [0.02 \quad 0.025 \quad 0.03]$mW and $F_c^3 = 1920$MHz.

We select w_1 to be the coordinator. Every terminal selects uniformly at random a RB so as to send its values to w_{cord}. At every iteration, the coordinator contacts the active terminals using RB 1. Every 10 GBP iterations, new channel estimations are obtained and sent to the coordinator.

Fig. 3.9 shows the results of the simulation, where for the values of $\mathbb{E}[\|\hat{\mathbf{x}} - \mathbf{x}_*\|_2]$, 20 independent experiments were conducted. Because the communication burden became greater, the time needed for convergence also increased compared to Fig. 3.7. We can also see that the coordinator scheme still outperforms the other two baselines.

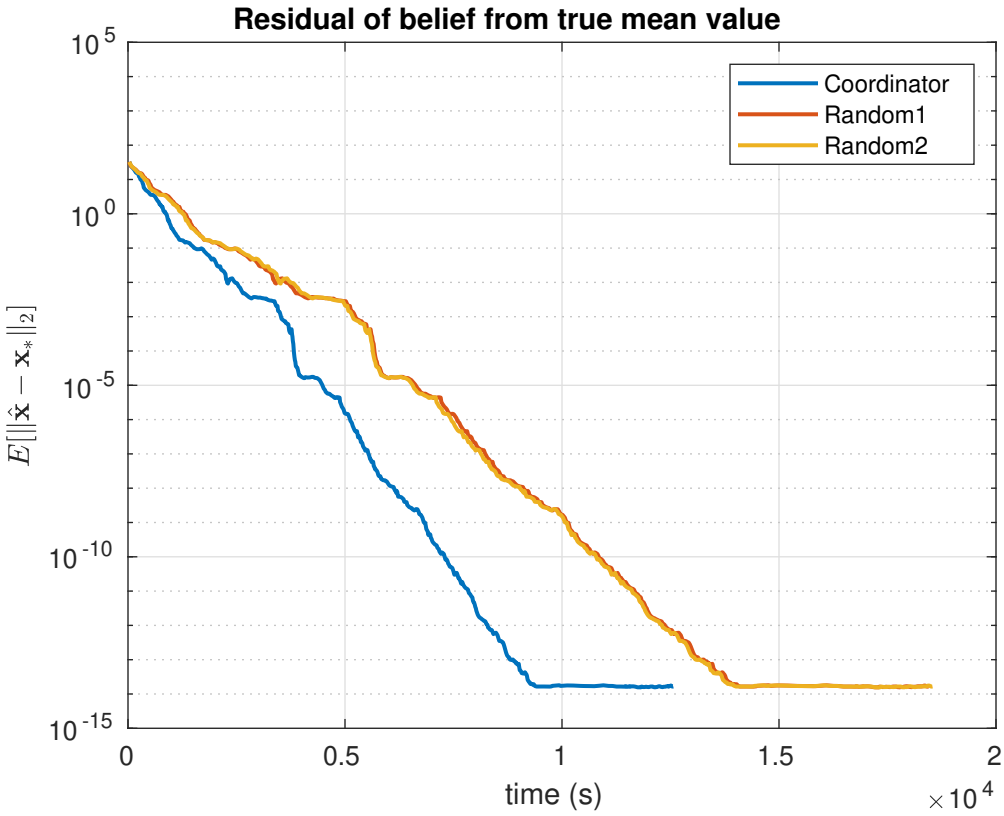


Figure 3.9: Convergence rate of GBP under non-i.i.d. asynchronous scheduling with 8 WSN terminals.

Chapter 4

The Bottleneck Assignment Problem

In this chapter we study one of the resource allocation problems of Sec. 3.5.1 (Eq. 3.44), which is known in the literature as Bottleneck Assignment Problem (BAP). Surprisingly, even though the BAP has been heavily studied, only two distributed algorithms have been proposed [7, 8]. Before providing its definition and presenting some of the most well-known concepts used in BAP algorithms, we discuss how we can use Belief Propagation (BP), which is inherently distributed, for solving combinatorial optimization problems. Then, we study the only line of work which uses BP for the BAP and offer some improvements. We provide new expressions, with reduced complexity for the BP messages and an asynchronous variant of the algorithm with convergence and correctness guarantees.

4.1 Belief Propagation for Combinatorial Optimization

Max-product BP is a popular iterative, message-passing algorithm for finding the *maximum a posteriori* (MAP) assignment of a joint probability distribution represented by a graphical model (GM). It is known to yield the correct solution when the GM has a tree structure [2]. In some applications that entail GMs with loops, the max-product algorithm is sometimes found to be exact or offer good approximations, yet providing theoretical guarantees for the loopy case in general remains a challenge. Its inherently distributed characteristics, the ease of implementation and strong parallelization potential render it a powerful tool. It turns out that certain combinatorial optimization problems can be solved by max-product. For the remaining of this section we will try to provide some intuition about the max-product algorithm on such problems, focusing on the Maximum Weight Matching (MWM) problem.

First we state and formulate the MWM problem. Consider an undirected weighted complete bipartite graph $K_{n,n} = (V_1, V_2, E)$, where $V_1 = \{\alpha_1, \ldots, \alpha_n\}$ can be thought of as a set of agents and $V_2 = \{\beta_1, \ldots, \beta_n\}$ as a set of tasks. Let each edge $(\alpha_i, \beta_j) \in E$, $\forall 1 \leq i, j \leq n$, have weight $w_{ij} \in \mathbb{R}$. The objective is to find a set of n edges so that the total sum of the corresponding weights is maximized, while satisfying the constraint that the selected edge set is a perfect matching; i.e., each agent selects exactly one task and each task is allocated to

exactly one agent. Assigning a decision variable $x_{ij} \in \{0,1\}$ to each edge in E, we can express the MWM problem as

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{1 \le i,j \le n} x_{ij} w_{ij} \\ \text{subject to} \quad & x_{ij} \in \{0,1\}, \qquad \forall\, 1 \le i,j \le n \\ & \sum_{1 \le i \le n} x_{ij} = 1, \qquad \forall\, 1 \le j \le n \\ & \sum_{1 \le j \le n} x_{ij} = 1, \qquad \forall\, 1 \le i \le n \end{aligned} \tag{4.1}$$

where $\mathbf{x} \in \{0,1\}^{|E|}$ and $\mathbf{x} = [x_{11}, x_{12}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{n1}, \ldots, x_{nn}]^T$.

Apparently, we have expressed the MWM problem as an integer programming problem. If we relax the integrality constraint; i.e., $0 \le x_{ij} \le 1$ for all $1 \le i,j \le n$, then we obtain a linear programming (LP) problem. The LP may yield a non-integer solution in general. However, we know that if an optimal solution to the relaxation is feasible for the integer programming problem, it is also an optimal solution to the latter, and that always holds for Eq. 4.1 and its relaxation [16]. Consequently, solving Eq. 4.1 is as hard as solving a LP problem.

In order to use max-product for solving the MWM problem we need to specify the graphical model as well as the joint distribution under consideration. In [17] the initial bipartite graph $K_{n,n}$ of the MWM is chosen as the graphical model. Additionally, the random variables $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ correspond to the vertices of $K_{n,n}$ and take values from $\{1, 2, \ldots, n\}$. Their joint probability distribution, $p(\overline{X} = (x_1, \ldots, x_n); \overline{Y} = (y_1, \ldots, y_n))$, is of the form:

$$p(\overline{X}, \overline{Y}) = \frac{1}{Z} \prod_{i,j} \psi_{\alpha_i \beta_j}(x_i, y_j) \prod_i \phi_{\alpha_i}(x_i)\, \phi_{\beta_i}(y_i)\,, \tag{4.2}$$

where the pairwise potentials $\psi(\cdot,\cdot)$ are defined as

$$\psi_{\alpha_i \beta_j}(r,s) = \begin{cases} 0, & r = j \text{ and } s \ne i \\ 0, & r \ne j \text{ and } s = i \\ 1, & \text{otherwise,} \end{cases} \tag{4.3}$$

the potentials at the nodes $\phi(\cdot)$ are defined as

$$\phi_{\alpha_i}(r) = e^{w_{ir}}, \quad \phi_{\beta_j}(r) = e^{w_{rj}}, \quad \forall 1 \le i,j,r,s \le n, \tag{4.4}$$

and Z is a normalization constant.

The idea is that, once the algorithm converges, every random variable X_i (or Y_i) indicates a neighbor of α_i (β_i) to be selected for the perfect matching. Each node has exactly n

neighbors, thus the random variables take values from $\{1, \ldots, n\}$ so that it is possible to choose any neighbor.

It is important that we encode all the available information of the optimization problem in the joint distribution, namely its constraints and cost function. More specifically, we utilize the pairwise potentials for the constraints and the node potentials for the cost function.

How do we ensure that the MAP assignment of the joint probability satisfies the constraints of the optimization problem? We define the probability distribution so that its support is restricted to perfect matchings. Therefore, every non-perfect matching assignment will have zero probability, hence will not be a MAP solution. By examining Eq 4.3 we see that for any $(\overline{X}, \overline{Y})$ with positive probability: a) If node α_i is matched to node β_j (i.e., $X_i = j$), then node β_j must be match to node α_i (i.e., $Y_j = i$). b) If node α_i is not matched to β_j (i.e., $X_i \neq j$), then node β_j must not be matched to node α_i (i.e., $Y_j \neq i$). For instance, assume that $n = 2$ and consider the configuration $(X_1 = 1, X_2 = 1; Y_1 = 1, Y_2 = 2)$, that is, agents α_1 and α_2 both select the task β_1, whereas tasks β_1 and β_2 are matched to agents α_1 and α_2 respectively. It is easy to see that this is not a perfect matching. The resulting pairwise potentials are: $\psi_{\alpha_1\beta_1}(1,1) = 1$, $\psi_{\alpha_1\beta_2}(1,2) = 1$, $\psi_{\alpha_2\beta_1}(1,1) = 0$ and $\psi_{\alpha_2\beta_2}(1,2) = 0$, thus $p(X_1 = 1, X_2 = 1; Y_1 = 1, Y_2 = 2) = 0$, as expected.

Now that we have discussed the role of the pairwise potentials, we can thing of the MAP assignment as

$$\arg \max_{(\overline{X},\overline{Y})} p(\overline{X}, \overline{Y}) = \arg \max_{(\overline{X},\overline{Y})} \prod_i \phi_{\alpha_i}(x_i)\, \phi_{\beta_i}(y_i)\,, \tag{4.5}$$

with the edge potentials guaranteeing that the optimal assignment yields a perfect matching. Looking at the cost functions of Eqs 4.1 and 4.5, it becomes obvious why the node potentials of Eq. 4.4 were chosen. Observe that for a perfect matching the cost function of Eq 4.5 is equal to $e^{2\sum_i w_{ix_i}}$.

So far we have seen how to encode the information of the MWM problem as a joint probability distribution so that its MAP assignment solves the initial problem. Nevertheless, one should not ignore the fact the GM under consideration has many loops, hence max-product's convergence and correctness must be proved. In particular, in [17], they prove the following theorem.

Theorem 2. For any weighted complete bipartite graph $K_{n,n}$ with unique MWM, the max-product algorithm when applied to the corresponding GM, converges to the correct MAP assignment or the MWM within $\lceil \frac{2nw^*}{\epsilon} \rceil$ iterations.

where w^* is the maximal value of edge weight and ϵ is the difference between the weight of the MWM and the second best MWM.

The uniqueness of the MWM solution can be easily guaranteed by adding some small random noise to the edge weights [17]. It is also worth mentioning that the maximum number of iterations needed for the algorithm to converge is not strictly polynomial in n, since it depends both on w^* and ϵ. However, in practice, max-product converges much faster than its theoretical upper bound. This is clearly illustrated in Fig. 4.1, where for different values

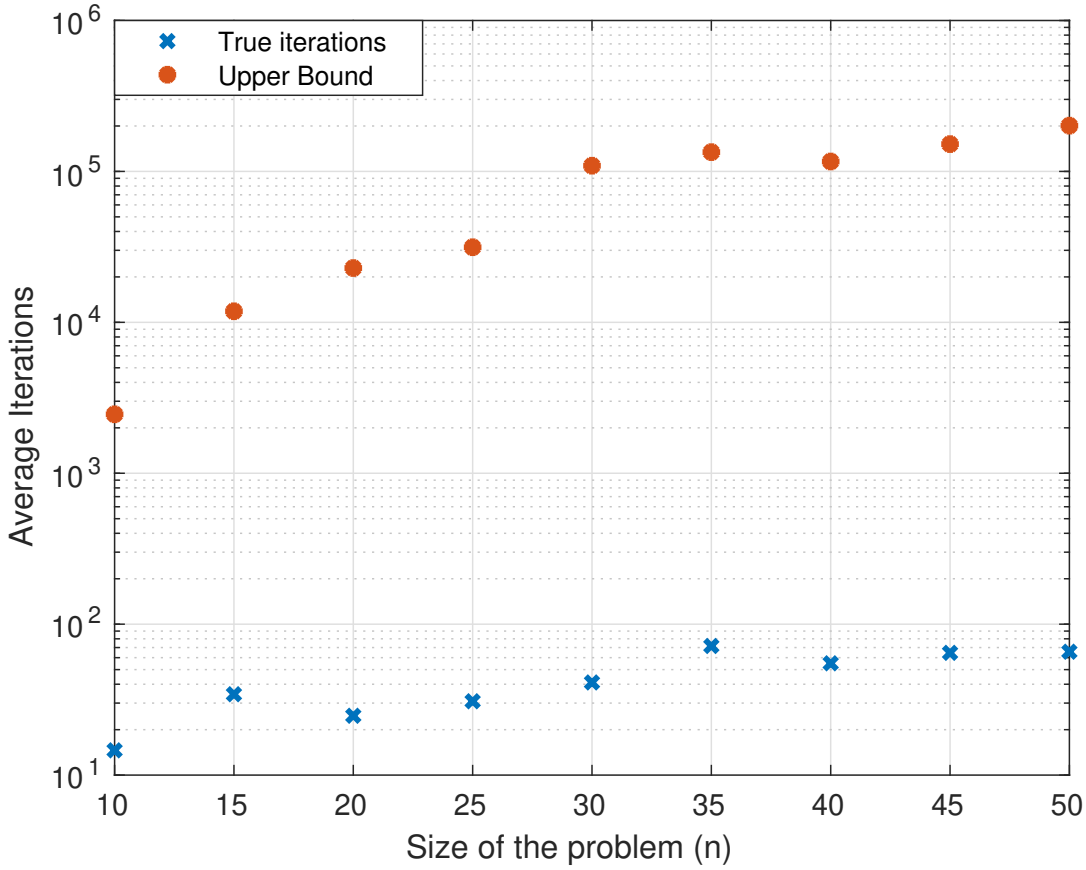


Figure 4.1: Max-product performance for the MWM problem.

of n, we solve 100 MWM problems, each with edge weights chosen uniformly at random from $[1, 1000]$, and then compute the average number of iterations until convergences as well as the average upper bound.

Maximum weight matching, shortest path, traveling salesman, cycle packing, vertex/edge cover and network flow can all be solved using BP [18]. In [18], the proof technique used in [17], for the convergence and correctness of max-product BP, is generalized and generic criteria about its applicability to LP problems are provided. What is more, in [11], they prove that BP can solve the weighted b-Matching problem on arbitrary graphs under synchronous as well as asynchronous scheduling.

4.2 Formulation and polynomial time algorithms

We have seen in Sec. 3.5.1 how the Bottleneck Assignment Problem (BAP) can be formulated as a 0/1 integer programming problem. In this section we are going to provide its formulation as a matching problem and discuss some polynomial time algorithms. In the sequel we also refer to the BAP as MiniMax Weight Matching Problem (MMWMP).

Consider an undirected weighted bipartite graph $G = (T, S, E)$ where $T = \{T_1, \ldots, T_n\}$ can be thought of a set of agents and $S = \{S_1, \ldots, S_n\}$ as a set of tasks. Let $m = |E|$ be the number of undirected edges and, for every edge $(T_i, S_j) \in E$, $w_{ij} \in \mathbb{R}$ be its corresponding weight. The objective is to find a perfect matching such that the costliest agent-task pairing is minimized. More formally, if $\mathsf{p} = \{\mathsf{p}(1), \mathsf{p}(2), \ldots, \mathsf{p}(n)\}$ is a permutation of $\{1, 2, \ldots, n\}$, then the collection of n edges $\left\{(T_1, S_{\mathsf{p}(1)}), (T_2, S_{\mathsf{p}(2)}), \ldots, (T_n, S_{\mathsf{p}(n)})\right\}$ is a perfect matching. Let W_{p} be the costliest pairing of a perfect matching:

$$W_{\mathsf{p}} = \max_{1 \leq i \leq n} w_{i\mathsf{p}(i)}. \tag{4.6}$$

The solution of the BAP, or MiniMax Weight Matching (MMWM), p^* is a perfect matching that satisfies

$$\mathsf{p}^* = \arg\min_{\mathsf{p}} W_{\mathsf{p}}. \tag{4.7}$$

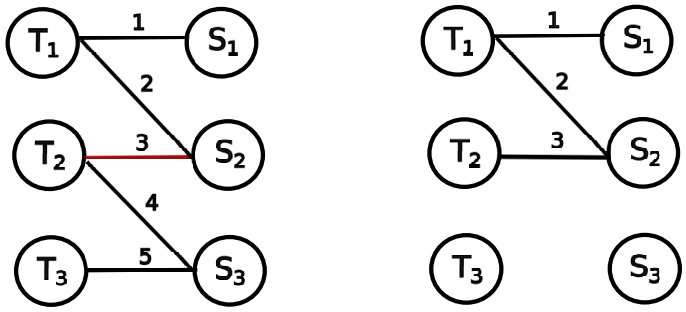
The BAP was first introduced in 1953 [19] and since then many polynomial algorithms have been proposed. One important family of such algorithms are the so-called *threshold algorithms*. A threshold algorithm alternates between two phases. In the first phase an edge $(T_i, S_j) \in E$, is selected and all edges of G with weight greater than w_{ij} are removed. This results in a subgraph, $\mathcal{G}_{ij} = (V, \mathcal{E}_{ij})$, with $\mathcal{E}_{ij} = \{(T_k, S_l) \in E : w_{kl} \leq w_{ij}\}$. In the second phase we check whether a perfect matching, denoted as p_{ij}, exists in $\mathcal{G}_{ij}$. If so, then we know that $W_{\mathsf{p}_{ij}} = w_{ij}$; i.e., the maximum weight of the perfect matching p_{ij} is that of the edge (T_i, S_j). Because we want to minimize the costliest pairing, the edge with the smallest weight for which the induced subgraph contains a perfect matching constitutes an optimal solution.

To better understand the above we present an example in Fig. 4.2. On the left you can see the graph G over which we want to solve (4.7), notice that there is only one possible perfect matching: $\mathsf{p}^* = \{(T_1, S_1), (T_2, S_2), (T_3, S_3)\}$. With red we mark the selected edge of phase one and with blue we mark the edges of perfect matching. On the right side you see the resulting subgraph for each case, which we examine for a perfect matching. In the first scenario (Fig. 4.2a) the selected edge does not result in a subgraph with perfect matching, thus next we should consider an edge with a higher weight. It easy to see that the smallest (and only) edge that induces a subgraph which contains a perfect matching is (T_3, S_3), Additionally, notice that the perfect matching of the subgraph (Fig. 4.2b) corresponds to p^*.

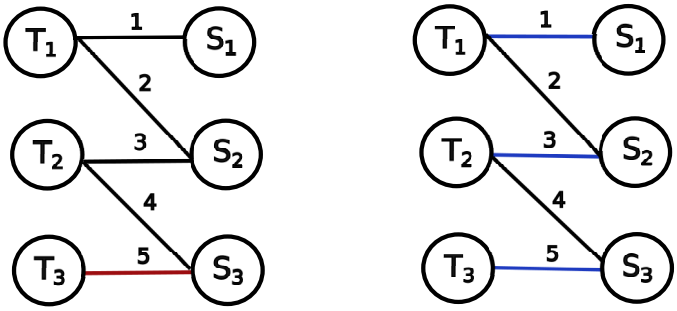
Regarding the complexity of threshold algorithms. Let $T(n)$ be the time complexity needed to find a perfect matching in a bipartite graph. Then, assuming we perform a binary search on the set of edge weights, the total complexity is $\mathcal{O}(T(n) \log n)$[1]. Concerning $T(n)$ refer to section 1 of [6].

Another concept used in many BAP algorithms is that of *augmenting path*. Define a matching $\mathcal{M}$ on the bipartite graph G to be a set of edges, with $|\mathcal{M}| \leq n$, so that each node in G has at most one of its edge in $\mathcal{M}$; we note that if $|\mathcal{M}| = n$, then M is a perfect matching. Next we define an *alternating path* P, relative to a matching $\mathcal{M}$, as a path that joins two nodes T_i and S_j and whose edges are alternatively free; i.e., not in $\mathcal{M}$, and matched; i.e., in

[1] In the worst case $m = n^2$.



(a) Induced subgraph does not have a perfect matching.



(b) Induced subgraph has a perfect matching.

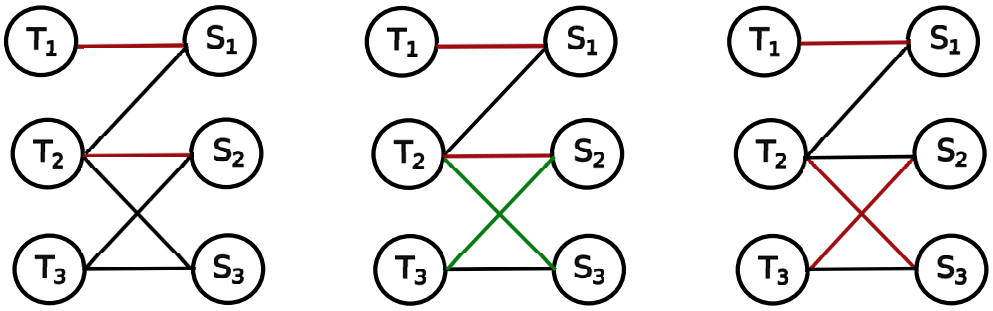
Figure 4.2: Threshold algorithm example.

$\mathcal{M}$. When the nodes T_i and S_j are free, then the alternating path joining them is called an augmenting path. This is important because by exchanging the free and matched edges in the path we get a new matching with an extra pair of nodes matched.

In Fig. 4.3 we provide an example. On the left side, you see a matching which comprises the edges $\mathcal{M} = \{(T_1, S_1), (T_2, S_2)\}$. We construct an alternating path in order to obtain a matching with cardinality $|\mathcal{M}| + 1$. In the middle of Fig. 4.3, you see the augmenting path $P = \{(T_3, S_2), (S_2, T_2), (T_2, S_3)\}$ connecting the unmatched nodes T_3 and S_3. Observe that the edges of the path alternate between free edges and edges of the matching, represented by green and red respectively. As a final step, we exchange the edges of the augmenting path and obtain the matching $\mathcal{M}' = \{(T_1, S_1), (T_2, S_3)(T_3, S_2)\}$, with $|\mathcal{M}'| = |\mathcal{M}| + 1$, as depicted on the right side.

According to [6], an algorithm with the best time complexity[2] is obtained by combining the threshold approach with augmenting paths, and the worst case complexity is $\mathcal{O}(m\sqrt{n \log n})$. In the special case where the edge weights are independent and identically distributed, [20] proposed an algorithm with expected running time $\mathcal{O}(n^2)$. For a comprehensive survey of the BAP and other assignment problems, refer to [6].

[2]The bibliography on this subject is vast and we did not find an algorithm with lower complexity.



(a) Initial matching. (b) Augmenting path. (c) New matching.

Figure 4.3: Augmenting path example.

4.3 Belief Propagation for the BAP

Surprisingly, even though the BAP is a well-studied problem, with several algorithms that solve it in polynomial time, there have not been proposed many distributed algorithms. After reviewing the literature, we found [7], from 2019, which claims to be the first line of work to address this problem. A year later, the same authors proposed a second distributed algorithm [8]. Both works discuss how augmenting paths can be found distributedly. Motivated by the inherently distributed characteristics of BP and by its capability to solve certain combinatorial optimization problems, we research its applicability to the BAP. In this section we explain why solving the BAP with a BP algorithm using the results of [18] is not possible. Then, we study the only line of work that provides a BP algorithm for the BAP [9]. Finally, we make some improvements; we propose new BP message equations with reduced complexity, and provide discussion on convergence and correctness guarantees for the asynchronous scheduling. To the best of our knowledge, this is the only asynchronous result for the BAP.

First we restate the BAP, for a $n \times n$ weighted complete bipartite graph, as an integer programming problem:

$$\begin{aligned} \min_{\mathbf{x}} \max_{i,j} \quad & x_{ij} w_{ij} \\ \text{subject to} \quad & x_{ij} \in \{0,1\}, \qquad \forall\, 1 \leq i,j \leq n \\ & \sum_{1 \leq i \leq n} x_{ij} = 1, \qquad \forall\, 1 \leq j \leq n \\ & \sum_{1 \leq j \leq n} x_{ij} = 1, \qquad \forall\, 1 \leq i \leq n \end{aligned} \tag{4.8}$$

where $\mathbf{x} = [x_{11}, x_{12}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{n1}, \ldots, x_{nn}]^T$.

Observe that the cost function of Eq. 4.8 is not linear; [18] requires the cost function to be linear. Introducing a variable q such that $q \geq x_{i,j} w_{i,j}$, $\forall\, 1 \leq i,j \leq n$, we can rewrite Eq. 4.8 as

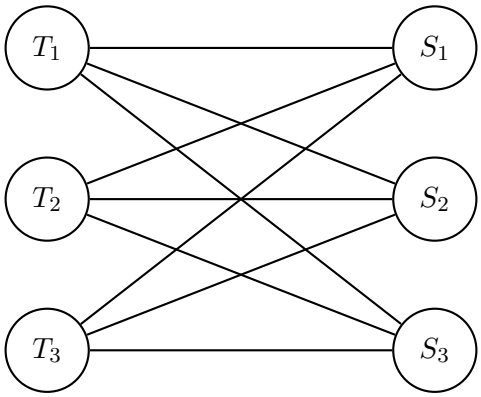


Figure 4.4: Original graph.

$$\min_{(\mathbf{x},q)} \quad q$$

$$\begin{aligned} \text{subject to} \quad & x_{ij} \in \{0,1\}, && \forall\, 1 \le i,j \le n \\ & \sum_{1\le i\le n} x_{ij} = 1, && \forall\, 1 \le j \le n \\ & \sum_{1\le j\le n} x_{ij} = 1, && \forall\, 1 \le i \le n \\ & q \ge x_{i,j} w_{i,j}, && \forall\, 1 \le i,j \le n \end{aligned} \tag{4.9}$$

This is a Mixed Integer Linear Programming (MILP) problem [16]. The linear relaxation of Eq. 4.9 does not always have integer solution. Hence, the results of [18] can not be utilized to obtain a BP algorithm for the BAP (or MMWMP).

To the best of our knowledge, there is only one line of work that provides a BP algorithm for the MMWMP. In [9], instead of defining a probability distribution and then derive the BP messages, they first propose the message equations and then prove the correctness of the algorithm using computation trees. The assumption made is that *the solution is unique*, i.e., there's only one perfect matching that minimizes the costliest pairing. In the rest of this section we present the work of [9].

In [9] the same formulation for the BAP is used as that of Section 4.2 (see Eq. 4.7). Fig. 4.4 shows a 3 by 3 original bipartite graph and the associated computation tree, of depth two, is depicted in Fig. 4.5. Notice that this computation tree is different to that of Sec. 2.3; instead of a root node here we have a root edge, thus we have two nodes at level 0. As we will see later, in this algorithm we are interested in the edge beliefs (not node beliefs). All the other rules, mentioned in Sec. 2.3, regarding the construction of a computation tree, hold.

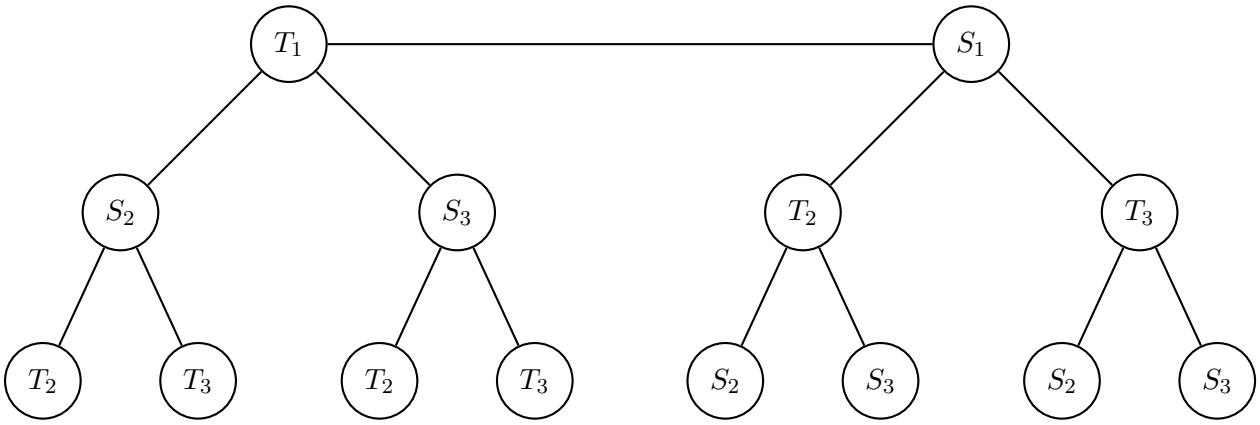


Figure 4.5: Computation tree of depth two for the original graph.

For each edge e on the computation tree, the following messages are defined.

Definition 3. $m(e)$ is the maximum weight of the MinMax Weight Matching (MMWM) on the subtree below e **with** e included.

Definition 4. $m'(e)$ is the maximum weight of the MinMax Weight Matching (MMWM) on the subtree below e but **without** e included.

Note a perfect tree matching is just one case of the general graph matchings. That is, each vertex except some leaves, must attach to one and only one associated edge. The main idea of the algorithm is to 1) first let every edge separately make its own decision about itself, to be chosen or not in the optimal tree matching, and 2) at the end gather all these individual decisions to form a global optimal matching in the original graph.

The belief of a root edge, e_r, is obtained by examining the difference $m(e_r) - m'(e_r)$. Intuitively, a root edge e_r, like (T_1, S_1) in Fig. 4.5, is not part of the optimal solution if $m(e_r) - m'(e_r) > 0$, because choosing e_r results in a perfect matching whose maximum weight is not minimized; according to $m'(e_r)$, we can discard e_r and obtain a MMWM with smaller maximum weight. On the contrary, when $m(e_r) - m'(e_r) < 0$, we select the edge e_r.

At the root edge we should combine the messages passed up from each of its endpoints, left (m_l from T_1, in Fig. 4.5) and right (m_r from S_1, in Fig. 4.5):

$$m(e_r) - m'(e_r) = \max\{m_l(e_r), m_r(e_r)\} - \max\{m'_l(e_r), m'_r(e_r)\}. \tag{4.10}$$

Now consider the updating rules at each node. For a $n \times n$ graph, each node, except the leaves, will have $n-1$ children in the corresponding computation tree. Refer to Fig. 4.6 for visualization. Mark the edges between the node and all its children as $i_1, i_2, \ldots, i_p$ and the edge between this node and its father as o. Let $w(e)$ return the weight of edge e and with $\mathbf{W}$ denote the weight matrix. Then

$$m(o) = \max\{w(o), \max_{1\le m\le p} m'(i_m)\}, \tag{4.11}$$

$$m'(o) = \min_{1\le m\le p}\{\max\{m(i_m), \max_{n\neq m} m'(i_n)\}\}. \tag{4.12}$$

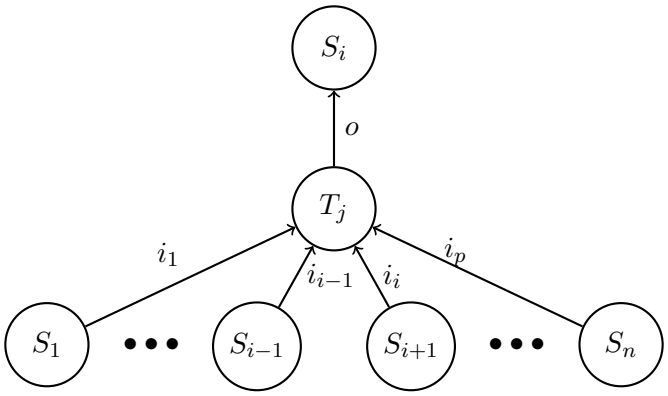


Figure 4.6: Messages passing through T_j.

Eq. 4.11 computes the message for the maximum weight of the minimax tree matching including edge o. In order to form a valid matching, if o is already included, all the $n-1$ children must be excluded. That is why the second maximization in Eq. 4.11 is taken on all the $m'(i_m)$'s, not the $m(i_m)$'s. Eq. 4.12 updates the message for the maximum weight of the minimax tree matching without edge o. Every node, except the leaves, must belong to one edge in the perfect tree matching. Since o is not chosen in this case, we thus have to pick up one edge from the $n-1$ children, while abandoning all that child's $n-2$ brothers. In order to find the minimax matching at the end, we take a minimization of all the maximum weights in Eq. 4.12.

Here is the entire algorithm:

Algorithm BP for MMWM

---

(1) Initialization:

$$m^{(0)}_{T_i \to S_j} = w_{i,j}, \tag{4.13}$$

$$m'^{(0)}_{T_i \to S_j} = \min(\mathbf{W}), \tag{4.14}$$

$$m^{(0)}_{S_j \to T_i} = w_{i,j}, \tag{4.15}$$

$$m'^{(0)}_{S_j \to T_i} = \min(\mathbf{W}). \tag{4.16}$$

(2) Messages at the kth iteration:

$$m^{(k)}_{T_i \to S_j} = \max\{w_{ij}, \max_{m \neq j} m'^{(k-1)}_{S_m \to T_i}\}, \tag{4.17}$$

$$m'^{(k)}_{T_i \to S_j} = \min_{m \neq j}\{\max\{m^{(k-1)}_{S_m \to T_i}, \max_{n \neq m,j} m'^{(k-1)}_{S_n \to T_i}\}\}, \tag{4.18}$$

$$m^{(k)}_{S_j \to T_i} = \max\{w_{ij}, \max_{m \neq i} m'^{(k-1)}_{T_m \to S_j}\}, \tag{4.19}$$

$$m'^{(k)}_{S_j \to T_i} = \min_{m \neq i}\{\max\{m^{(k-1)}_{T_m \to S_j}, \max_{n \neq m,i} m'^{(k-1)}_{T_n \to S_j}\}\}. \tag{4.20}$$

(3) Decisions at the end of kth iteration:

$$D^{(k)}_{ij} = \max\{m^{(k)}_{T_i \to S_j}, m^{(k)}_{S_j \to T_i}\} - \max\{m'^{(k)}_{T_i \to S_j}, m'^{(k)}_{S_j \to T_i}\}. \tag{4.21}$$

If $D^{(k)}_{ij} < 0$, choose the edge (T_i, S_j); otherwise, not.

Regarding the computational complexity of the messages: Eqs. 4.17 and 4.19 require us to find the maximum over n values and then compare it to w_{ij}, so $\mathcal{O}(n)$ comparisons are needed. In Eqs. 4.18 and 4.20, we find the minimum of $n-1$ values, for each of those we need $n-2$ comparisons for the inner maximization and one more for the outer maximization, in total $\mathcal{O}(n^2)$ comparisons are needed.

According to [9], the following theorem holds:

Theorem 3. For an n by n weighted complete bipartite graph, if the minimax weight matching is unique, the algorithm converges to the optimal solution in n iterations.

Let K_α be the number of iterations needed for the convergence of the algorithm, and K_m the iterations needed for the convergence of the messages. What can be derived directly is:

$$K_\alpha \leq K_m. \tag{4.22}$$

Intuitively, when the messages converge, the decision matrix D will stay the same, which leads to the convergence of the algorithm. However, the convergence of the algorithm, i.e., all the signs of the cells in the decision matrix will never change anymore, does not necessarily ensure the convergence of the messages. Another question thus arises: Will the messages converge? An intuitive answer is yes, because both $m(e_r)$ and $m'(e_r)$ are nondecreasing and bounded by $\max(\mathbf{W})$.

Regarding the complexity of the algorithm: a complete bipartite graph of size n has n^2 edges. In every iteration, for each edge, we compute 2 messages that need $\mathcal{O}(n^2)$ comparisons, thus executing the algorithm serially we have a computational complexity of $\mathcal{O}(n^4)$ per iteration. Since we have at most n iterations, the total computational complexity is $\mathcal{O}(n^5)$. Because this is a message passing algorithm, in a distributed

implementation assuming that every edge can compute and pass the messages simultaneously within one unit time in each iteration, the solution can be found in $\mathcal{O}(n)$ unit times.

4.3.1 New BP messages

Here we provide new expressions, with reduced complexity, for the messages of the BP algorithm for the MMWM problem. Refer to Fig. 4.6.

$$m'_{new}(o) = \min_{1 \le m \le p} \{m(i_m)\}. \tag{4.23}$$

Eq. 4.23 has complexity $\mathcal{O}(n)$ and we argue that it can be used instead of Eq. 4.12 which has complexity $\mathcal{O}(n^2)$.

For notational convenience, denote the computation tree below an edge e with e included as $CT\{e\}$, and the computation tree below an edge e but without e included as $\overline{CT}\{e\}$.

Lemma 1. For a 2 by 2 complete bipartite graph, Eq. 4.23 satisfies Definition 4.

Proof of Lemma 1:
Each node of the computation tree has $n-1$ children, so in this case, where $n = 2$, each node has exactly one child.

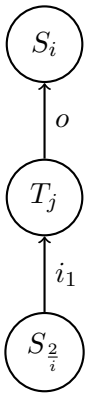


Figure 4.7: Messages passing through T_j when $n = 2$.

Refer to Fig. 4.7; we are interested in finding an expression for the $\overline{CT}\{o\}$ that satisfies Definition 4. In the perfect tree matching, every node of $\overline{CT}\{o\}$, except some leaves, must belong to one edge. Additionally, in $\overline{CT}\{o\}$ the edge o is not considered, thus the node T_j belongs to the edge i_1. Therefore, the maximum weight of the MMWM on the $\overline{CT}\{o\}$ is obtained by $m(i_1)$.

Now, it suffices to show that $m'_{new}(o) = m(i_1)$ when $n = 2$. From Eq. 4.23:

$$m'_{new}(o) = \min_{1 \le m \le p} \{m(i_m)\} \overset{p=1}{=} m(i_1). \tag{4.24}$$

□

Now we provide an example to explain why Eq. 4.23 may not satisfy Definition 4 when the problem is of size $n \geq 3$ and the depth of the computation tree is less than 3. Consider a 3×3 complete bipartite graph and the following computation tree

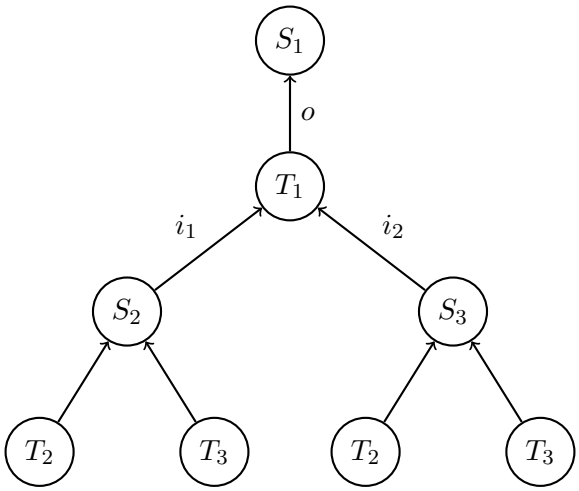


Figure 4.8

We want $m'(o)$ to return the maximum weight of the MMWM on $\overline{CT}\{o\}$. Eq. 4.23 examines each subtree $CT\{i_m\}$, $m \in \{1, 2\}$, independently, ignoring the rest of the perfect tree matching on the other neighboring subtree. Specifically, in Fig. 4.8, Eq. 4.23 can only "see" the maximum weights of the following tree matchings: $\{(T_1, S_2)$ and $\{(T_1, S_3)\}$, while the perfect tree matchings for $\overline{CT}\{o\}$ are: $\{(T_1, S_2), (T_2, S_3)\}$, $\{(T_1, S_2), (T_3, S_3)\}$, $\{(T_1, S_3), (T_2, S_2)\}$ and $\{(T_1, S_3), (T_3, S_2)\}$. However, this is not a problem once the computation tree has depth at least 3 and the previous messages have been computed correctly.

Lemma 2. For a n by n weighted complete bipartite graph, with $n \geq 3$, given the correct messages of the third iteration, Eq. 4.23 satisfies Definition 4.

Proof of Lemma 2:
First we prove the correctness of Eq. 4.23 for the forth iteration. Refer to Fig. 4.9, by assumption we know the correct values of $m(i_m)$ and $m'(i_m)$, $\forall 1 \leq m \leq p$. Each node of the computation tree has $n - 1$ children and the computation tree has depth 3. We want $m'_{new}(o)$ to return the maximum weight of the MMWM on $\overline{CT}\{o\}$. In the perfect tree matching, every node of $\overline{CT}\{o\}$, except some leaves, must belong to one edge. Additionally, in $\overline{CT}\{o\}$ the edge o is not considered, thus the node T_j belongs to one edge i_m, $1 \leq m \leq n$. Now we assume that examining each subtree $CT\{i_m\}$ separately

is sufficient; i.e., we don't need to search at neighboring subtrees $\overline{CT}\{i_\lambda\}$, $\lambda \neq m$, in order to complete the perfect tree matching. Under this assumption, Definition 4 is satisfied by selecting the smallest of the $m(i_m)$ values,[3] which is exactly what Eq. 4.23 does.

The aforementioned assumption holds because each $CT\{i_m\}$ has depth 3, and by the construction of the computation tree, all different nodes of the original (bipartite) graph $(T \cup S)$ are encountered in each $CT\{i_m\}$; see Fig. 4.9 for visualization. Perfect tree matchings on subtrees $CT\{i_m\}$ include pairings for all different nodes of the original graph and thus searching the neighboring subtrees is redundant.

For the next iterations exactly the same arguments hold, with the only difference being that the $CT\{i_m\}$ has depth at least 4 or more. □

Using Eq. 4.23 instead of Eq. 4.12, the new modified step (2) of the algorithm BP for MMWM is:

(2) Messages at the kth iteration:

$$m_{T_i \to S_j}^{(k)} = \max\{w_{ij}, \max_{m \neq j} m'^{(k-1)}_{S_m \to T_i}\}, \tag{4.25}$$

$$m'^{(k)}_{T_i \to S_j} = \min_{m \neq j}\{m^{(k-1)}_{S_m \to T_i}\}, \tag{4.26}$$

$$m_{S_j \to T_i}^{(k)} = \max\{w_{ij}, \max_{m \neq i} m'^{(k-1)}_{T_m \to S_j}\}, \tag{4.27}$$

$$m'^{(k)}_{S_j \to T_i} = \min_{m \neq i}\{m^{(k-1)}_{T_m \to S_j}\}. \tag{4.28}$$

From Lemma 2 we know that the messages of Eq. 4.23 are correct after the third iteration, assuming that the previous ones have been computed correctly. We also explained why Eq. 4.23 may fail to satisfy Def. 4 in the first 3 iterations. At first glance, these may lead someone to believe that the algorithm can not converge to the correct solution, provided that it is unique, using the proposed equations. However, after numerous simulations we have not been able to find such an example. Intuitively, the reason for this is that as the depth of the computation tree grows, the influence of the first messages decreases.

Our experimental results indicate that messages of Eq. 4.12 and 4.23 eventually converge to the same correct values, provided that the solution of the BAP is unique. In the following experiments we use the subscript *new* for messages that stem from Eq. 4.23, while l and r are used for messages $T \to S$ and $S \to T$, respectively. For instance, $\mathbf{m}'^{(k)}_{l\,new}$ is a vector constructed by stacking all messages $m'^{(k)}_{T \to S}$ computed by Eq. 4.26, of iteration k, whereas $\mathbf{m}'^{(k)}_{l}$ is constructed similarly using messages $m'^{(k)}_{T \to S}$ computed by Eq. 4.18.

[3] Recall that $m(i_m)$ returns the maximum weight of the MMWM on $CT\{i_m\}$.

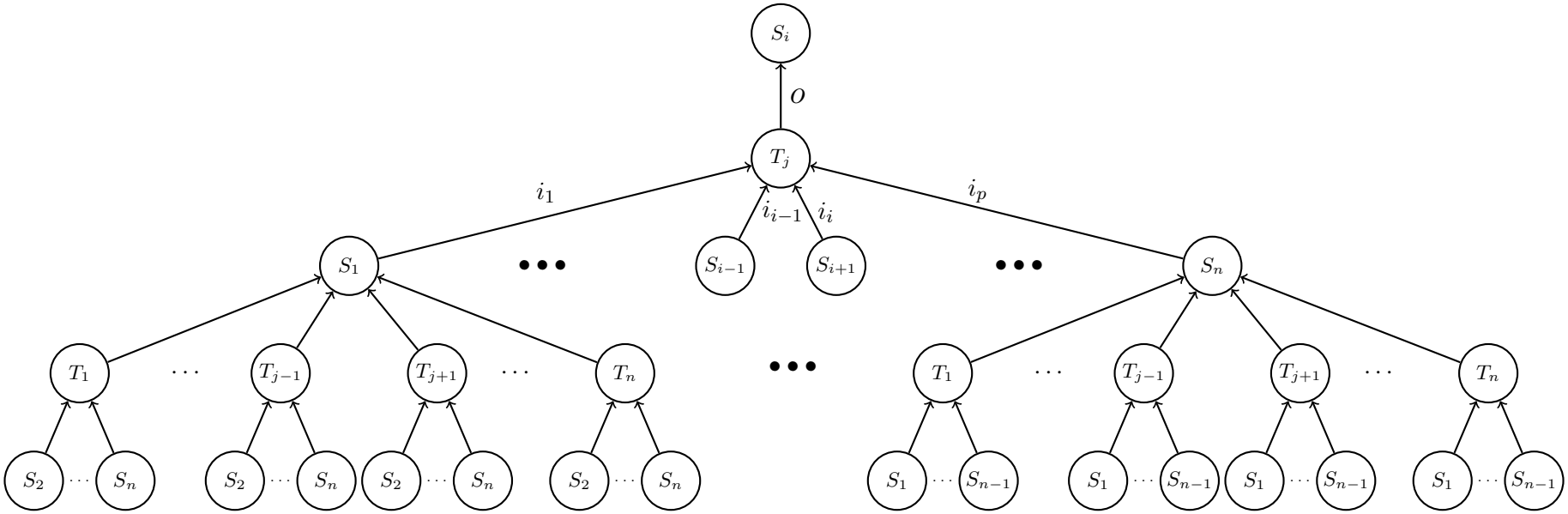


Figure 4.9: Messages passing through T_j, with depth of $CT\{i_m\}$ equal to 3.

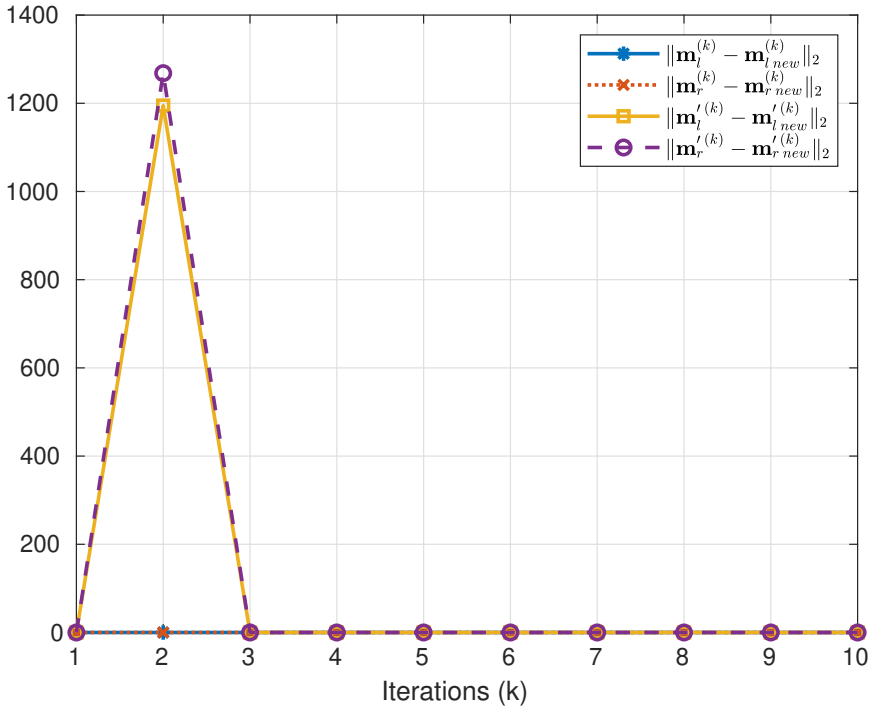


Figure 4.10: The different message expressions converge in less than $n = 5$ iterations.

Consider the following weight matrix

$$\mathbf{W} = \begin{bmatrix} 400 & 292 & 107 & 952 & 423 \\ 527 & 432 & 373 & 921 & 548 \\ 417 & 16 & 199 & 53 & 943 \\ 657 & 985 & 490 & 738 & 418 \\ 628 & 168 & 340 & 270 & 984 \end{bmatrix} \tag{4.29}$$

Fig. 4.10 illustrates that the different message expressions return the same values after the second iteration. We note that the decision matrices, from the different message equations, are the same in every iteration, i.e., we have different messages but same beliefs.

Interestingly, the different messages do not always converge before the nth iteration. For

$$\mathbf{W} = \begin{bmatrix} 321 & 530 & 453 & 525 & 851 \\ 512 & 830 & 753 & 973 & 912 \\ 61 & 859 & 110 & 711 & 640 \\ 726 & 790 & 110 & 312 & 256 \\ 557 & 318 & 270 & 292 & 89 \end{bmatrix} \tag{4.30}$$

as we can see in Fig. 4.11, the message converge to the same values after the sixth iteration, while the problem size is $n = 5$. However, similarly to the previous case, the decision matrices are identical in every iteration.

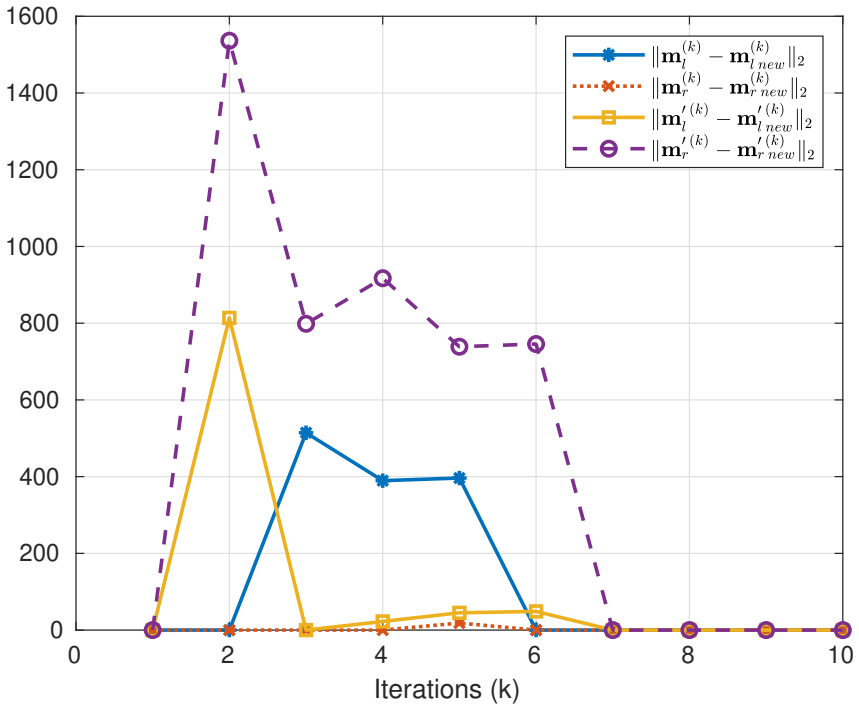


Figure 4.11: The different message expressions converge after more than $n = 5$ iterations.

4.3.2 Asynchronous variant

In this section we show that the algorithm BP for MMWM, converges to the optimal solution (assuming that it is unique) under totally asynchronous scheduling, which is the most general form of asynchronous scheduling. For the theoretical guarantees we use the technique introduced in section 6 of [11].

Consider a $n \times n$ complete weighted bipartite graph $G = (T, S, E)$. The update equations are exactly analogous to the synchronous version, but at each time only a subset of the edges are updated in an arbitrary order. Denote with $\overrightarrow{E}$ the set of all directed edges in G; i.e. $\overrightarrow{E} = \{(T_a \to S_b) : 1 \leq a, b \leq n\} \cup \{(S_b \to T_a) : 1 \leq a, b \leq n\}$. Let A be a sequence $\overrightarrow{E}(1), \overrightarrow{E}(2), \ldots$ of subsets of the set $\overrightarrow{E}$. Then, the asynchronous BP algorithm corresponding to the sequence A can be obtained by modifying only the step (2) of the algorithm BP for MMWM.

(2) Messages in iteration t:

$$m_{T_i\to S_j}^{(t)} = \begin{cases} \max\{w_{ij}, \max_{m\neq j} m_{S_m\to T_i}^{\prime(t-1)}\}, & \text{if } (T_i \to S_j) \in \overrightarrow{E}(t) \\ m_{T_i\to S_j}^{(t-1)}, & \text{otherwise} \end{cases} \tag{4.31}$$

$$m_{T_i\to S_j}^{(t)} = \begin{cases} \min_{m\neq j}\{m_{S_m\to T_i}^{(t-1)}\}, & \text{if } (T_i \to S_j) \in \overrightarrow{E}(t) \\ m_{T_i\to S_j}^{(t-1)}, & \text{otherwise} \end{cases} \tag{4.32}$$

$$m_{S_j\to T_i}^{(t)} = \begin{cases} \max\{w_{ij}, \max_{m\neq i} m_{T_m\to S_j}^{\prime(t-1)}\}, & \text{if } (S_j \to T_i) \in \overrightarrow{E}(t) \\ m_{S_j\to T_i}^{(t-1)}, & \text{otherwise} \end{cases} \tag{4.33}$$

$$m_{S_j\to T_i}^{\prime(t)} = \begin{cases} \min_{m\neq i}\{m_{T_m\to S_j}^{(t-1)}\}, & \text{if } (S_j \to T_i) \in \overrightarrow{E}(t) \\ m_{S_j\to T_i}^{\prime(t-1)}, & \text{otherwise} \end{cases} \tag{4.34}$$

We assume that the sequence A of the updates does not have redundancies. That is, no edge direction $(i \to j) \in \overrightarrow{E}$ is re-updated before at least one of its incoming edge directions is updated. More formally, let $N(i)$ be the set of neighbors of node i, then if $(i \to j) \in \overrightarrow{E}(t) \cup \overrightarrow{E}(t+s)$ and $(i \to j) \notin \cup_{r=1}^{s-1}\overrightarrow{E}(t+r)$, then at least for one $l \in N(i)\backslash\{j\}$, we should have $(l \to i) \in \cup_{r=1}^{s-1}\overrightarrow{E}(t+r)$.

We denote the above algorithm by Async-BP for MMWM. Let $u(t)$ be the minimum number of times that an edge direction of the graph G appears in the sequence $\overrightarrow{E}(1), \ldots, \overrightarrow{E}(t)$:

$$u(t) = \min_{(i\to j)\in\overrightarrow{E}} \left(\left| \{l : \text{ s.t. } 1 \leq l \leq t \text{ and } \quad (i \to j) \in \overrightarrow{E}(l)\} \right| \right). \tag{4.35}$$

We claim that the following result holds.

Theorem 4. For a $n \times n$ weighted complete bipartite graph, if the minimax weight matching is unique, the algorithm Async-BP for MMWM converges to the optimal solution after at most t iterations, provided $u(t) \geq n$.

Because of the asynchronous scheduling, we can not utilize the computation tree of an edge which assumes that at each iteration all messages are updated. Instead, we use the generalized computation tree (GCT), introduced in [11], which takes into account the asynchronous scheduling.

For any $(i \to j) \in \overrightarrow{E}(t)$, define $R_{i\to j}^{t}$ to be the computation branch of i to j at time t, which is a rooted tree (not necessarily balanced) and recursively defined according to the following rules:

a) The root has label j.

b) The root has only one child which has label i.

c) If $t = 0$, then the child i has no child ($R^0_{i\to j}$ is just a single edge (i, j))

d) For $t \geq 0$, if $(i \to j) \notin \overrightarrow{E}(t)$ then $R^t_{i\to j} = R^{t-1}_{i\to j}$. Otherwise, the child i has $n-1$ children which have all of labels in the set $N(i)\backslash\{j\}$ and for any child r of i the subtree that consists of all descendants of r and the edge $(r \to i)$ is $R^{t-1}_{r\to i}$.

For any edge $(T_a, S_b) \in E$ and any t, the GCT $R^t_{(T_a,S_b)}$ is a tree with root edge (T_a, S_b) (i.e., the nodes T_a and S_b are connected at level 0) and with computation branches $R^t_{S_r\to T_a}$, for $r = 1, \ldots, a-1, a+1, \ldots, n$, and $R^t_{T_k\to S_b}$, for $k = 1, \ldots, b-1, b+1, \ldots, n$. Here we have slightly modified the definition of GCT in order to have a root edge instead of a root node.

Since the GCT is not necessarily balanced, we define the depth of $R^t_{(T_a,S_b)}$ to be the length of the shortest path from T_a or S_b to a leaf and we denote it by $d(R^t_{(T_a,S_b)})$.

Proof of Theorem 4. First we associate the depth of $R^t_{(T_a,S_b)}$ with $u(t)$ using the following lemma.

Lemma 3. For any vertex $i \in T \cup S$ and any t, the depth of any computation branch at time t is at least $u(t)$; i.e., $d(R^t_{i\to j}) \geq u(t)$ for $(i \to j) \in \overrightarrow{E}$.

Proof of Lemma 3. [11] □

This tells us that when $u(t) \geq n$, $d(R^t_{(T_a,S_b)}) \geq n$; i.e., the shortest computation branch of the GCT, which is constructed under totally asynchronous scheduling, has depth at least n, that is, all computation branches of the GCT have depth at least n after t iterations.

The rest of the proof is similar to the proof of Theorem 1 from [9], with the only difference being that we use the GCT of depth $u(t)$ instead of the computation tree constructed under synchronous scheduling. □

4.4 Numerical Results

Now we present some experimental results. We note that the new proposed messages of Sec. 4.3.1 were used. First, for $n = 10$, we generate a random weight matrix $\mathbf{W}$ with integer elements sampled uniformly at random from $[1, 1000]$.

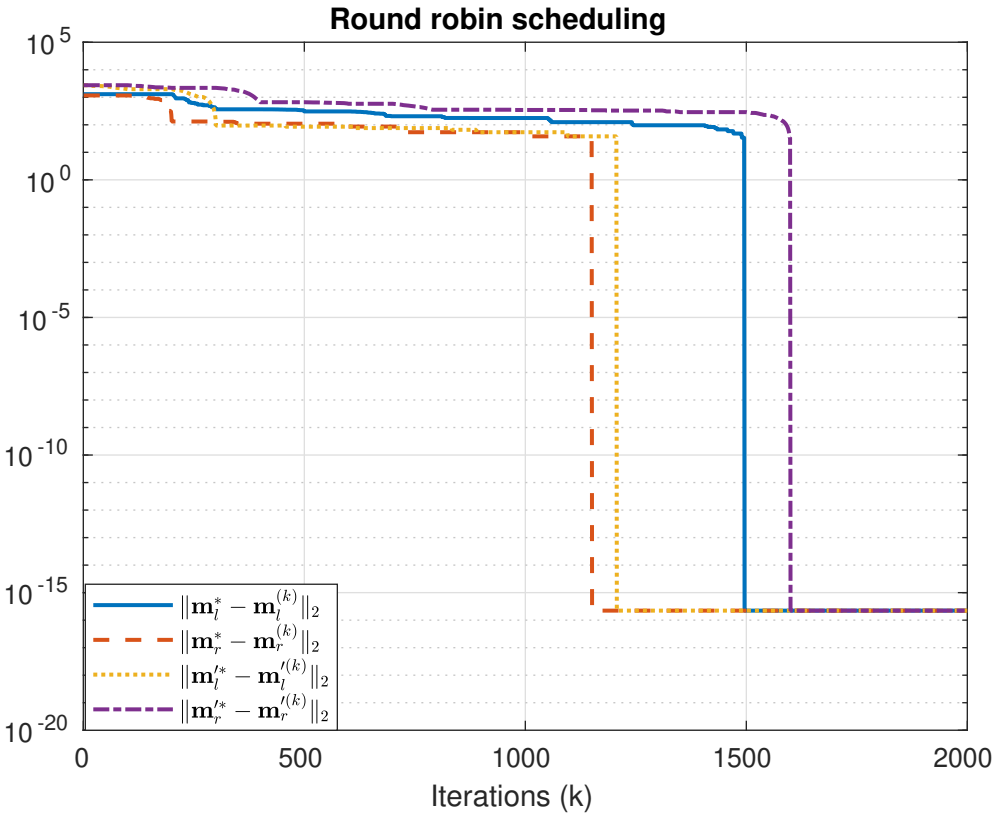


Figure 4.12: Convergence to optimal solution under round robin asynchronous scheduling.

$$\mathbf{W} = \begin{bmatrix} 312 & 595 & 86 & 964 & 38 & 107 & 31 & 183 & 60 & 660 \\ 924 & 263 & 263 & 547 & 886 & 654 & 745 & 240 & 682 & 519 \\ 431 & 603 & 802 & 522 & 914 & 495 & 501 & 887 & 43 & 973 \\ 185 & 712 & 30 & 232 & 797 & 780 & 480 & 29 & 72 & 649 \\ 905 & 222 & 929 & 489 & 99 & 716 & 905 & 490 & 522 & 801 \\ 980 & 118 & 731 & 625 & 262 & 904 & 610 & 168 & 97 & 454 \\ 439 & 297 & 489 & 680 & 336 & 891 & 618 & 979 & 819 & 433 \\ 112 & 319 & 579 & 396 & 680 & 335 & 860 & 713 & 818 & 826 \\ 259 & 425 & 238 & 368 & 137 & 699 & 806 & 501 & 723 & 84 \\ 409 & 508 & 459 & 988 & 722 & 198 & 577 & 472 & 150 & 134 \end{bmatrix} \tag{4.36}$$

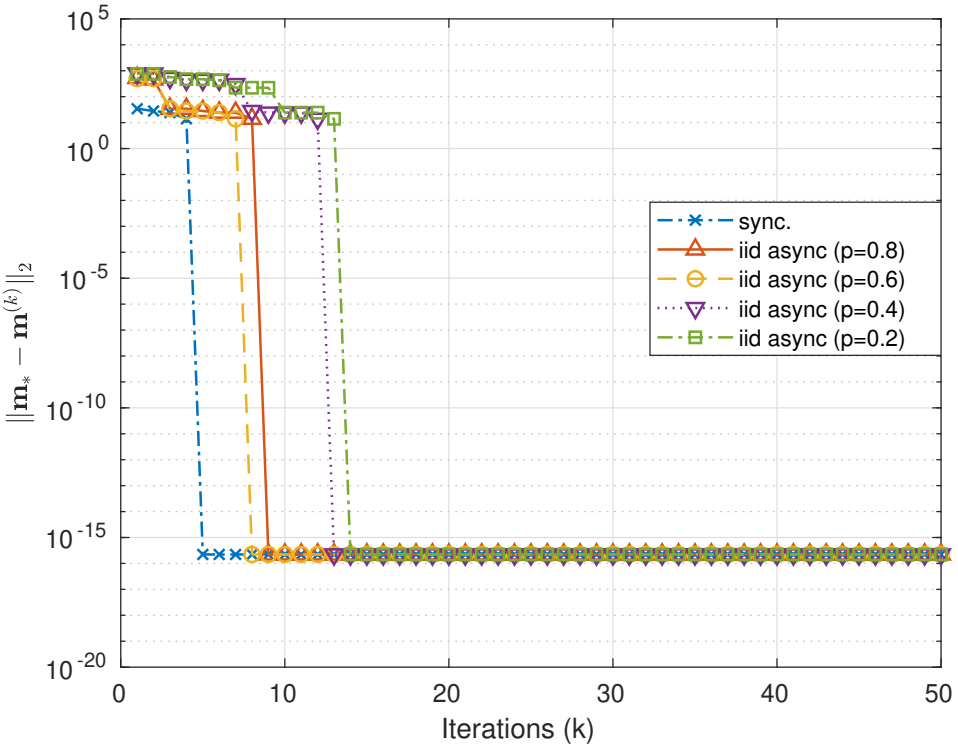
Then, we use Async-BP for MMWM to solve the corresponding BAP under round robin asynchronous scheduling. Specifically, for a fixed ordering of the directed edges, in each iteration we update only one directed edge; i.e., one node transmits to one of its neighbors ($m_{T_i \to S_j}$ and $m'_{T_i \to S_j}$ or $m_{S_j \to T_i}$ and $m'_{S_j \to T_i}$ are sent). We stack all $m^{(k)}_{T \to S}$ and $m'^{(k)}_{T \to S}$ to obtain the vectors $\mathbf{m}_l^{(k)}$ and $\mathbf{m}_l'^{(k)}$ correspondingly, we do the same for $m^{(k)}_{S \to T}$ and $m'^{(k)}_{S \to T}$ to construct the vectors $\mathbf{m}_r^{(k)}$ and $\mathbf{m}_r'^{(k)}$. In this particular instance, we have 200 directed edges and in every iteration we update one of them, thus an edge is updated every 200 iterations. According to Theorem 4, by the time every edge has been updated at least $n = 10$ times, the algorithm must have converged. For the last directed edge of the ordering, $200 \cdot 10 = 2000$ iterations need to take place in order to be updated $n = 10$ times, so the algorithm is expected to converge after at most 2000 iterations. In Fig. 4.12 we can see that this holds, since the messages converge to their optimal values in less than 2000 iterations (Recall that messages require at least as many iterations as beliefs need for convergence).

Next we consider the i.i.d. asynchronous scheduling. In every iteration, each node in the undirected bipartite graph samples its Bernoulli random variable with parameter p, independently from the other nodes, and based on the outcome decides whether to send the messages to all of its neighbors or not transmit at all. This asynchronous scheduling is a subclass of totally asynchronous scheduling, and because each directed edge is updated with probability $p > 0$, we expect the algorithm to converge to the optimal solution.

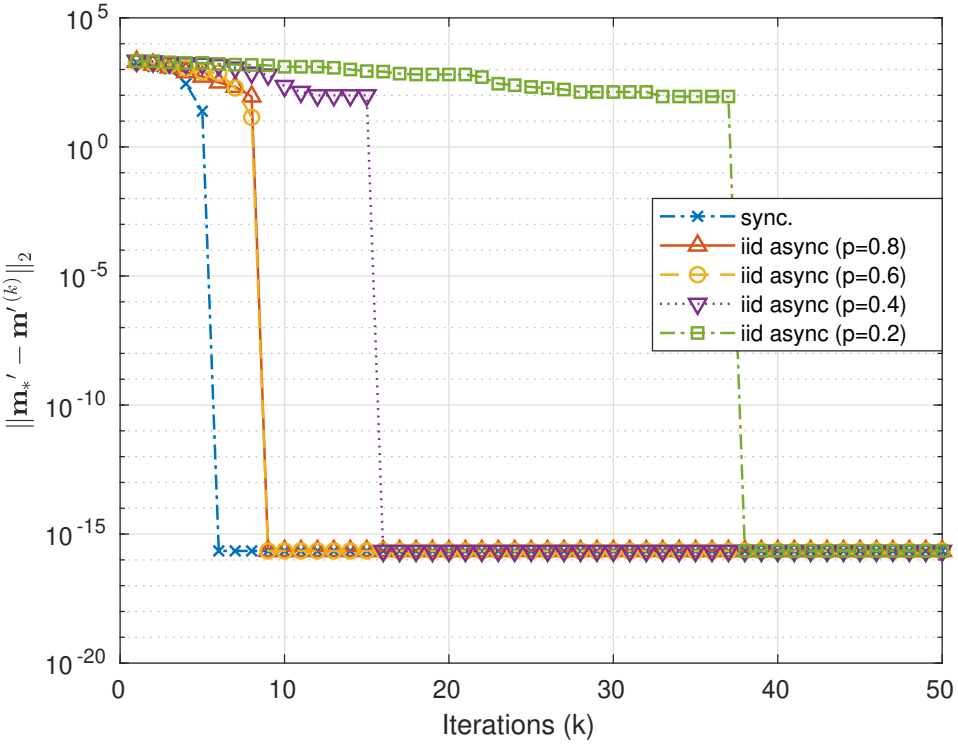
As before, we generate a random weight matrix

$$\mathbf{W} = \begin{bmatrix} 461 & 895 & 323 & 860 & 995 & 527 & 37 & 63 & 107 & 451 \\ 516 & 88 & 780 & 598 & 207 & 247 & 64 & 407 & 999 & 578 \\ 272 & 540 & 336 & 655 & 608 & 543 & 323 & 464 & 867 & 75 \\ 232 & 429 & 620 & 916 & 348 & 781 & 99 & 203 & 616 & 58 \\ 900 & 618 & 993 & 434 & 718 & 522 & 171 & 870 & 27 & 301 \\ 909 & 559 & 649 & 290 & 28 & 932 & 372 & 598 & 323 & 522 \\ 604 & 226 & 540 & 632 & 67 & 148 & 40 & 24 & 464 & 562 \\ 366 & 105 & 233 & 296 & 928 & 417 & 710 & 900 & 100 & 242 \\ 599 & 10 & 740 & 623 & 88 & 281 & 642 & 453 & 571 & 913 \\ 669 & 60 & 889 & 48 & 333 & 599 & 175 & 59 & 326 & 826 \end{bmatrix} \tag{4.37}$$

We stack all messages $m^{(k)}_{T\to S}$ and $m^{(k)}_{S\to T}$ and form the vector $\mathbf{m}^{(k)}$, similarly we construct $\mathbf{m'}^{(k)}$ from $m'^{(k)}_{T\to S}$ and $m'^{(k)}_{S\to T}$. We experiment with four different values for the parameter p $(0.2, 0.4, 0.6, 0.8)$. As we can see from Fig. 4.13a and 4.13b the algorithm does indeed converge in all cases, as we expected.



(a)



(b)

Figure 4.13: Convergence to optimal solution under i.i.d. asynchronous scheduling.

Chapter 5

Conclusions and Future Work

In this work we studied how the Gaussian Belief Propagation (GBP) algorithm can be utilized by a Wireless sensor network (WSN) so as to solve linear systems of equations. After reviewing the theoretical properties of the algorithm under asynchronous scheduling, we showed that the non-i.i.d. asynchronous scheduling can not be directly used by a WSN. The induced transmission delays combined with intermittent operations, render it challenging for the WSN terminals to know when each iteration of the algorithm should be executed. As a consequence, in practice, the algorithm may diverge even though it can converge in theory. To address this issue, we proposed the coordinator scheme and formulated two 0/1 integer programming problems, for the wireless resources to be allocated optimally. Through simulations it was shown that the proposed changes result in shorter convergence time (approximately 20%) compared to other scenarios. It would be interesting to test this idea in a real WSN and see if the simulation results are verified. We note that, reviewing the literature, we did not find polynomial time algorithms for the integer programming problem of Eq. 3.48, thus efficient algorithms, possibly heuristic, need to be developed given the limited computation power of WSN terminals.

Next, we considered the Bottleneck Assignment Problem (BAP) with a focus on solving it with Belief propagation (BP). An existing line of work was examined and some improvements were offered: new BP messages with computational complexity $\mathcal{O}(n)$, instead of $\mathcal{O}(n^2)$, and convergence and correctness guarantees for the asynchronous variant. The main limitation of BP in this case is the fact that the solution of the BAP needs to be unique; adding small random noise as in the Maximum Weight Matching (MWM) problem is not sufficient. Therefore, an interesting future direction would be to research under which conditions the problem has a unique solution. Additionally, since there exist only three distributed algorithms for the BAP, a detailed comparison between the algorithm BP for MMWM and the algorithms of [7, 8] would be interesting (we expect that the BP approach will require less algorithmic iterations at the expense of increased communication burden).

Bibliography

[1] Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. Complexity of inference in graphical models. *arXiv preprint arXiv:1206.3240*, 2012.

[2] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[3] Bin Li and Yik-Chung Wu. Convergence analysis of gaussian belief propagation under high-order factorization and asynchronous scheduling. *IEEE Transactions on Signal Processing*, 67(11):2884–2897, 2019.

[4] Vasileios Papageorgiou, Athanasios Nichoritis, Panagiotis Vasilakopoulos, Georgios Vougioukas, and Aggelos Bletsas. Towards ambiently powered inference on wireless sensor networks: Asynchrony is the key! In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 458–465. IEEE, 2021.

[5] Vasileios Papageorgiou. Asynchronous inference in ambiently powered wireless sensor networks. Diploma thesis, Technical University of Crete, 2021.

[6] Rainer E Burkard and Eranda Cela. Linear assignment problems and extensions. In *Handbook of combinatorial optimization*, pages 75–149. Springer, 1999.

[7] Mitchell Khoo, Tony A Wood, Chris Manzie, and Iman Shames. Distributed algorithm for solving the bottleneck assignment problem. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1850–1855, 2019.

[8] Mitchell Khoo, Tony A Wood, Chris Manzie, and Iman Shames. A distributed augmenting path approach for the bottleneck assignment problem. *arXiv preprint arXiv:2011.09606*, 2020.

[9] Mindi Yuan, Shen Li, Wei Shen, and Yannis Pavlidis. Belief propagation for minimax weight matching. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 37–45. Springer, 2015.

[10] Devavrat Shah. Algorithms for inference (6.438). MIT OpenCourseWare, 2014. Lecture notes.

[11] Mohsen Bayati, Christian Borgs, Jennifer Chayes, and Riccardo Zecchina. Belief propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. *SIAM Journal on Discrete Mathematics*, 25(2):989–1011, 2011.

[12] Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[13] Danny Bickson. *Gaussian belief propagation: Theory and aplication*. PhD thesis, Hebrew University of Jerusalem, 2008.

[14] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[15] Athanasios P. Liavas. Wireless communications. Technical University of Crete, 2019. Lecture notes.

[16] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*, volume 13. Dynamic Ideas Belmont, 2005.

[17] Mohsen Bayati, Devavrat Shah, and Mayank Sharma. Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Transactions on information theory*, 54(3):1241–1251, 2008.

[18] Sejun Park and Jinwoo Shin. Convergence and correctness of max-product belief propagation for linear programming. *SIAM Journal on Discrete Mathematics*, 31(3):2228–2246, 2017.

[19] DR Fulkerson. *A production line assignment problem*. Rand Corporation, 1953.

[20] Ulrich Pferschy. The random linear bottleneck assignment problem. *RAIRO-Operations Research*, 30(2):127–142, 1996.