TECHNICAL UNIVERSITY OF CRETE
ELECTRONIC AND COMPUTER ENGINEERING DEPARTMENT
TELECOMMUNICATIONS DIVISION

# Factor Graphs: Theory and Applications

by

Panagiotis Alevizos

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

ELECTRONIC AND COMPUTER ENGINEERING

September 2012

THESIS COMMITTEE

Assistant Professor Aggelos Bletsas, *Thesis Supervisor*
Assistant Professor George N. Karystinos
Professor Athanasios P. Liavas

# Abstract

*Factor graphs (FGs) represent graphically the factorization of a global function into a product of local sub-functions. The global function is usually a multi-variable probability density function (pdf), where the calculation of a marginal pdf is usually intractable. The sum-product algorithm (SPA) is applied on the FG through message-passing, i.e. exchange of functions, between the FG nodes in a distributed way; the output is a marginal pdf with respect to a variable of interest. Factor graph theory has several applications in many interdisciplinary fields, such as error correction coding theory, detection and estimation, wireless networking, artificial intelligence and many others.*

*This thesis provides the basic theoretical background in a tutorial way, from first principles. Furthermore, specific FG applications found in the literature are presented. Specifically, coding problems (LDPC, convolutional and parallel concatenated Turbo codes), Bayesian estimation (in the context of network localization) and wireless multi-hop networking (in the context of time scheduling) are analyzed within the FG framework. In all cases, the respective graph, the associated SPA, the message-passing scheduling and the final output are thoroughly presented.*

*The power of FGs as a distributed inference tool is vividly demonstrated.*

# Acknowledgements

Thanks to my parents for the psychological support and for their unconditional love.

I would like to thank a lot my thesis supervisor, Aggelos Bletsas, for his patience, guidance, ideas, support, inducement in research and especially for his contribution in my knowledge. Many thanks to Professor Athanasios Liavas who provided his experience and knowledge whenever needed and to Assistant Professor George Karystinos who provided me the motivation to study about communication theory.

Special thanks to my girlfriend for her patience, her support, her love and especially for the most beautiful moments in my life. I apologize whenever I was steep and nervous.

Finally, I would like to thank a lot my friends for the adorable moments I had with them and for their support.

# Table of Contents

# List of Figures

# List of Abbreviations

| | |
|---|---|
| GLD | generalized distributed law |
| LDPC | low density parity check |
| FG | factor graph |
| MAP | maximum a-posteriori |
| pdf | probability density function |
| pmf | probability mass function |
| SPA | sum-product algorithm |

# Preface

The introduction of new terminology will be *emphasized* when it appears for first time. Definitions, theorems, lemmas corollaries and examples share the same index within each chapter. The symbol □ stands for the end of proof of theorem, or lemma, or corollary. The symbol ■ denotes the end of an example. The symbol ▼ denotes the end of a definition.

| | |
|---|---|
| $x$ | a variable |
| $\mathbf{x}$ | a vector |
| $\mathbf{A}$ | a matrix |
| $\mathbf{A}^\top$ | transpose of $\mathbf{A}$ |
| $\mathbf{I}_n$ | $n \times n$ identity matrix |
| $\lvert\mathcal{C}\rvert$ | the cardinality of a set $\mathcal{C}$ |
| $\lvert\lvert\mathbf{x}\rvert\rvert_p$ | the $p$ norm of a vector $\mathbf{x}$ |
| $f(x) \propto g(x)$ | the function $g(x) = kf(x)$ for some some scalar $k$ |
| $\mathbb{R}$ | the set of real numbers |
| $\mathbb{N}$ | the set of natural numbers |
| $\mathbb{B}$ | the set of binary numbers |
| $\mathbb{F}_q$ | the $q-$ary field |
| $\delta(\cdot)$ | delta Dirac function or delta Kronecker function. Every time it is used, it will be clarified which of two function is |
| $\arg\max_{\mathbf{x}}\{f(\mathbf{x})\}$ | the argument that maximizes $f(\mathbf{x})$ |

The vector $\mathbf{x}_{k:l}$ follows a MATLAB-like notation that denotes the sub-vector of $\mathbf{x}$ that consist of the $k$-th up to $l$-th elements of it.

# Chapter 1

# Introduction

## 1.1 Motivation

The sum-product algorithm (also known as belief propagation algorithm) is a powerful tool of graphical models with several extensions in many communication fields such as signal processing, coding theory, artificial intelligence and many others. Factor graphs model several problems in communication field and provide a unified manner to resolve them.

## 1.2 Related Work

The emanation of factor graphs originates from coding theory and more specifically from Robert Gallagher in 1960 [1] who introduced low density parity check (LDPC) codes and expressed them via factor graphs in his PhD thesis. Tanner in [2] explicitly introduced graphs to describe LDPC codes, presented the fundamentals of iterative decoding on graphs and also defined block and convolutional error-correcting codes in terms of *bipartite* graphs, which consist of codeword bit nodes and parity check constraints. Finally factor graphs provide the way of representing graphically the factorization of a function; such factorization will help us explore powerful algorithms for digital communications and signal processing in the sequel of this thesis.

The literature on graphical models and their applications is vast. Kschischang *et al.* in [3] formalized factor graph, showing that belief propagation and many algorithms used in digital communications and signal processing are all representations of a more general message passing algorithm, the sum-product algorithm. These graphs were used for code description, construction and decoding. Wiberg *et al.* in [4], introduced "hidden" state variables and also proposed applications on coding. Markov random fields [5] and Bayesian networks have closed connections with factor graphs and graphical representation [6]. These graphical models, like factor graphs, are usually factorization of pmf (probability mass functions) of several random variables. Pearl's "belief propagation" (BP) algorithm [7], operates in a factor graph expressing the same factorization as the above, which can

be viewed as an instance of sum product algorithm. Forney in [8] shows how to create a "normal" realization of a factor graph, demonstrating how some classes of codes can be represented as realizations on such graphs. A set of applications for the FG framework is discussed in [9]. Finally Yedidia *et al.* in [10] expresses sum-product algorithm (SPA) in terms of free-energy functions of statistical physics, showing that inference problems in FGs can be represented as a constraint optimization problems of such functions.

## 1.3 Thesis Outline

The thesis is organized as follows :

- Chapter 2 introduces the notion of factor graphs, describing how variables of a global function are grouped together locally, constructing a graph. Such graphs have some useful properties, which will be examined therein.
- Chapter 3 introduces linear block, LDPC, convolutional and Turbo codes from a factor graph perspective, demonstrating how decoding algorithms can be performed via sum-product algorithm on factor graphs.
- Chapter 4 describes localization as Bayesian estimation, and how factor graphs are connected with it. It will be shown that after expressing the topology of the network in a graph, it can be derived a factor graph based algorithm, in order to localize all nodes of the network efficiently.
- Chapter 5 considers the application of FGs in the problem of time scheduling in multi-hop wireless networks. Given a packet radio network (PRN) we can express it in terms of FGs, and applying SPA we derive a consistent as well as near-optimal (in terms of channel utilization) time scheduling.
- The final Chapter 6 consist of the conclusions about factor graphs and their applications.

# Chapter 2

# Factor Graph Theory: Basics

## 2.1 Graphs and factors

The first subsection begins with some preliminaries about graph theory, followed by an introduction of (global) functions of many variables which can be factorized as a product of smaller functions with domain, a subset of the global function's domain. The first part consists of auxiliary material providing definitions about graph theory which is useful in many fields, including signal processing, artificial intelligence, statistical inference etc. The second part offers some basic definitions about functions of many variables with finite domain, their factorization in smaller functions and basic definitions about marginalization with respect to a single or more variables.

### 2.1.1 Introduction to graphs

In this section we will provide some basic definitions about graphs that will be useful in the rest of this thesis. These definitions can be also found in [24] and [26].

**Definition 2.1** (**Graph** [24])**.** A graph consists of a pair of sets $G(V, E)$ such that $E \subseteq V \times V$, i.e. the elements in $E$ are two-element subsets of $V$; $V$ represents the set of vertices of the graph, while $E$ denotes the set of edges. ▼

**Definition 2.2** (**Adjacency** [24])**.** Given an edge $e \in E$, there are two vertices in $V$, namely $v_1$, $v_2$, such that $e = (v_1, v_2)$. We say that $v_1$ and $v_2$ are adjacent and the set of vertices adjacent to vertex $v$ is denoted $\mathcal{N}_G(v)$. Given a vertex $v \in V$, there are two edges $e_1$, $e_2$ such that $e_1 = (v, v_1)$ and $e_2 = (v, v_2)$ for some $v_1, v_2 \in V$. We say that $e_1$ and $e_2$ are adjacent. The set of edges adjacent to edge $e$ is denoted $\mathcal{N}_G(e)$. ▼

**Definition 2.3** (**Subgraphs** [26])**.** A graph $H(V_H, E_H)$ is a subgraph of a graph $G(V_G, E_G)$, denoted by $H \subseteq G$, if $V_H \subseteq V_G$ and $E_H \subseteq E_G$. A subgraph $H \subseteq G$ spans $G$ (and $H$ is a spanning subgraph of $G$), if every vertex of $G$ is in $H$, i.e., $V_H = V_G$. ▼

**Definition 2.4** (**Degree** [24])**.** The degree of a vertex $v$ is the number of its adjacent vertices and denoted $d_G(v)$, i.e. $d_G(v) = |\mathcal{N}_G(v)|$. ▼

**Definition 2.5** (**Minimum degree, Maximum degree** [26])**.** The minimum and the maximum degree of a graph $G(E, V)$ are defined as:

$$\delta(G) = \min_{v \in G} \{d_G(v)\}, \quad \Delta(G) = \max_{v \in G} \{d_G(v)\}.$$

▼

**Definition 2.6** (**Isomorphism** [26])**.** Two graphs $G$ and $H$ are isomorphic, denoted by $G \cong H$, if there exists a mapping $\alpha : V_G \mapsto V_H$ such that

$$(v_1, v_2) \in E_G \Longleftrightarrow (\alpha(v_1), \alpha(v_2)) \in E_H$$

for all $v_1,\ v_2 \in V_G$. ▼

**Definition 2.7** (**Incidence** [24])**.** Given an edge $e \in E$, there are two vertices $v_1$, $v_2$ such that $e = (v_1, v_2)$. We say that $v_1$ and $v_2$ are incident with $e$ and at the same time $e$ is said to be incident with $v_1$ and $v_2$. The set of vertices incident with an edge $e$ is denoted $\mathcal{I}_G(e)$. Similarly, the set of edges incident with a vertex $v$ is denoted $\mathcal{I}_G(v)$. Obviously, $v \in \mathcal{I}_G(e) \Leftrightarrow e \in \mathcal{I}_G(v)$ ▼

**Definition 2.8** (**Walk** [26])**.** Let $e_i = (v_i, v_{i+1}) \in G(V, E)$ be edges of a graph $G$ for $i = 1, ..., N$. The sequence $\{e_1, e_2, ..., e_N\}$ is a walk of length $N$ from $v_1$ to $v_{N+1}$ if $e_i$ is adjacent to $e_{i+1}$ for all $i = 1, ..., N$. ▼

We write more informally,

$$W : v_1 \to v_2 \to ... \to v_{N+1}, \quad \text{or } W : v_1 \xrightarrow{N} v_{N+1}.$$

We write $v \xrightarrow{\star} u$ to say that there is a walk of some length from $v$ to $u$. We use this notation because sometimes we don't care about the edges $e_i$ of the walk $W$. The length of the walk is denoted by $|W|$.

**Definition 2.9** (**Closed, Path, Cycle** [26])**.** Let $W = \{e_1, e_2, ..., e_N\}$ with $e_i = (v_i, v_{i+1})$ be a walk.
We say $W$ is closed if $v_1 = v_{N+1}$.
We say $W$ is a path, if $v_i \neq v_j$ for all $i \neq j$.

We say $W$ is cycle, if it is closed, and $v_i \neq v_j$ for all $i \neq j$ except $v_1 = v_{N+1}$. The length of the cycle is $N + 1$. ▼

An example of cyclic graph is depicted in figure 2.1. The degree of vertex $v_1$ is 1, the degree of vertices $v_3, v_4, v_5$ is 2 and the degree of vertex $v_2$ is 3. The cycle $(v2 \to v3 \to v4 \to v5 \to v2)$ has length 4, since the number of edges of the cycle is equal to 4.

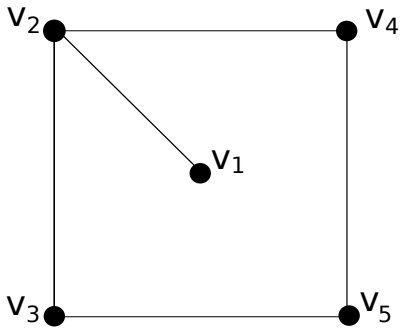



Figure 2.1: A cyclic graph with five vertices and five edges. The degree of vertex $v_1$ is 1, the degree of vertices $v_3, v_4, v_5$ is 2 and the degree of vertex $v_2$ is 3.

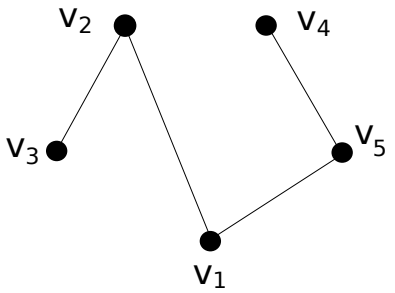



Figure 2.2: A graph that is tree with five vertices and four edges. The degree of vertices $v_3$ and $v_4$ is 1, while the degree of vertices $v_1, v_2$ and $v_5$ is 2.

**Definition 2.10** (**Connected graph** [26])**.** Suppose there exists a walk from $v$ to $w$ in

$G$, then the following quantity

$$d_G(v, w) = \min \{k \mid v \xrightarrow{k} w\}$$

is called the distance between $v$ and $w$. If there are no walks $v \xrightarrow{\star} w$, we write $d_G(v, w) = \infty$. A graph $G$ is connected, if $d_G(v, w) < \infty$ for all $v,\ w \in G$; otherwise, it is disconnected. ▼

Figure 2.2 illustrates an acyclic graph. The distance between vertices $v_3$ and $v_1$ is 2, while the distance from vertex $v_3$ to $v_4$ is equal to 4.

**Definition 2.11** (**Connected components** [26])**.** The maximal connected subgraphs of $G$ are its connected components, denoted as

$$c(G) = \textit{the number of connected components of } G.$$

If $c(G) = 1$, then $G$ is connected. ▼

In figures 2.1, 2.2, 2.3 there is only one connected component, the whole graph, therefore $c(G) = 1$. In figure 2.4 there are 2 connected components, hence $c(G) = 2$.

**Definition 2.12** (**Girth**)**.** The minimum cycle length of a graph is called the girth of the graph, provided that there is a non-trivial cycle (cycle-length $\geq 3$). If the graph has no cycles its girth is equal to 2. ▼

For example in figure 2.1 the girth of this graph is 4 since the smallest cycle (and the only one) has length 4.

**Definition 2.13** (**Bipartite graph**)**.** A bipartite graph is a graph whose vertices can be divided into two disjoint sets $U$ and $V$ such that every edge connects a vertex in $U$ to a vertex in $V$; that is, $U$ and $V$ are independent sets. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles. ▼

**Corollary 2.14.** *The minimum girth of a cyclic bipartite graph is 4.*

*Proof.* Since bipartite graphs have even-length cycles and the cycles of length 2 are trivial, i.e. they are not considered as cycles since they traverse the same edge, the minimum even number after the number 2 is the number 4. Therefore, cyclic bipartite graphs have minimum girth equal to 4. □

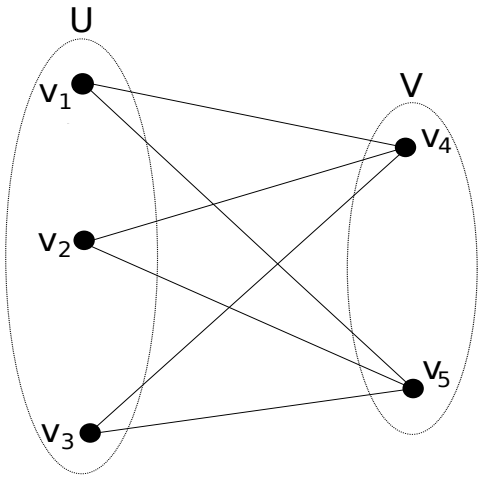



Figure 2.3: A bipartite graph with five vertices and six edges. The reader easily can note that this family of graphs has even-length cycle paths, since the traversal from one side to other and then the return back, consist of even number of edges on graph.

An example of a bipartite graph is given in figure 2.3. It can be seen that a bipartite graph contains only even number of cycles since one node must pass across the other side and then an edge must return back.

**Definition 2.15** (**Bridge** [26])**.** An edge $e \in G$ is a bridge of the graph $G$, if $G - e$ has more connected components than $G$, i.e. if $c(G - e) > c(G)$. In particular, an edge $e$ in a connected graph $G$ is called bridge if and only if $G - e$ is disconnected. We note that, for each edge $e \in G$, $e = (v, u)$ is a bridge $\iff u$, $v$ belong at different connected components of $G - e$. ▼

In figure 2.4 we show a cyclic graph with girth 3 and two connected components (attention this graph is not a bipartite graph, since it does not satisfies the preconditions of a bipartite graph). The edges which are bridges are the edges $(v_2, v_3)$, $(v_3, v_4)$, $(v_4, v_5)$ and $(v_6, v_7)$, since if we delete anyone of them, then the number of connected components of the resulting graph will be increased. The maximum and minimum degree of the graph is three and one, respectively.

**Definition 2.16** (**Tree** [26])**.** A graph is called acyclic, if it has no cycles. An acyclic graph is also called a forest. A tree is a connected acyclic graph. ▼

**Corollary 2.17** ([26])**.** *A connected graph is a tree if and only if all its edges are bridges.*

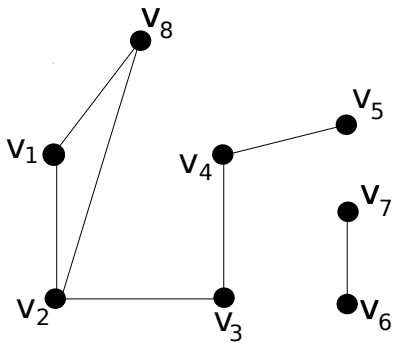



Figure 2.4: We present a cyclic graph with girth 3 and two connected components. The edges which are bridges are $(v_2, v_3)$, $(v_3, v_4)$, $(v_4, v_5)$ and $(v_6, v_7)$. The maximum and minimum degree of the graph is three and one respectively.

### 2.1.2 Introduction to factors

Through this thesis we deal with functions of several variables. Let $X_1, X_2, ..., X_n$ be a set of variables, in which for each $i$, $X_i$ takes values in some finite *domain* $\mathcal{X}_i$. Let $f(X_1, X_2, ..., X_n)$ be a real valued function of these variables, i.e. a function with domain

$$\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n, \tag{2.1}$$

and range the set of real numbers $\mathbb{R}$. The domain $\mathcal{X}$ of $f$ is called *configuration space* for the given set of variables $\{X_1, ..., X_n\}$, and each element of $\mathcal{X}$ is a particular configuration of the variables, i.e. an assignment of a value for each input of $f$. Knowing that the set of real numbers is closed over summation, we will associate $n$ *marginal functions*[1] associated with function $f(X_1, ..., X_n)$, denoted as $g_{X_i}(x_i)$ for every $i$. For each $x_i \in \mathcal{X}_i$, the value $g_{X_i}(x_i)$ is obtained by summing the value of $f(X_1, ..., X_n)$ over all configurations of the input variables that have $X_i = x_i$.

**Definition 2.18** (**Marginal** [24])**.** The marginal of $f(X_1, ..., X_n)$ with respect to variable $X_i$ is a function from $\mathcal{X}_i$ to $\mathbb{R}$ which is denoted $g_{X_i}(x_i)$, and it is obtained by summing over all other variables. More specifically, the marginal with respect to variable $x_i \in \mathcal{X}_i$ is given by

$$g_{X_i}(x_i) = \sum_{x_1 \in \mathcal{X}_1} \cdots \sum_{x_{i-1} \in \mathcal{X}_{i-1}} \sum_{x_{i+1} \in \mathcal{X}_{i+1}} \cdots \sum_{x_n \in \mathcal{X}_n} f(X_1 = x_1, ..., X_i = x_i, ..., X_n = x_n). \tag{2.2}$$

[1]Through this thesis we will consider the terms marginal and marginal function as equivalent terms

For notational convenience, instead of indicating the variables being summed over, we indicate those variable not being summed over and we will use the following shorthand

$$g_{X_i}(x_i) = \sum_{\sim\{x_i\}} f(x_1, ..., x_i, ..., x_n) \tag{2.3}$$

$$= \sum_{x_1,...,x_{i-1},x_{i+1},...,x_n} f(x_1, ..., x_i, ..., x_n). \tag{2.4}$$

▼

Following the approach of [24], let $f(X_1, ..., X_n)$ factors into a product of several *local* functions, each having some subset of $\{X_1, ..., X_n\}$ as arguments, specifically, it is assumed that $(X_1, ..., X_n)$ can be factorized into $K$ factors, namely,

$$f(X_1, ..., X_n) = \prod_{k=1}^{K} f_k(S_k), \tag{2.5}$$

where $S_k \subseteq \{X_1, ..., X_n\}$ is the subset of variables associated with the real-valued local factor $f_k$, i.e. its configuration space. Such factorization is not unique. Function $f(\cdot)$ itself is a trivial factorization, since it consist of 1 factor. Consider the following example.

**Example 2.1.** Consider the function of $n = 6$ variables:

$$f(X_1, X_2, X_3, X_4, X_5, X_6) = f_1(X_2, X_5) f_2(X_1, X_3, X_6) f_3(X_1, X_4),$$

with $S_1 = \{X_2, X_5\},\ S_2 = \{X_1, X_3, X_6\}$ and $S_3 = \{X_1, X_4\}$. The marginal with respect to variable $X_4$ is described as follows:

$$\begin{aligned} g_{X_4}(x_4) &= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_5} \sum_{x_6} f(x_1, x_2, x_3, x_4, x_5, x_6) \\ &= \sum_{x_1,x_2,x_3,x_5,x_6} f(x_1, x_2, x_3, x_4, x_5, x_6) \\ &= \sum_{x_1,x_2,x_3,x_5,x_6} f_1(x_2, x_5) f_2(x_1, x_3, x_6) f_3(x_1, x_4), \end{aligned}$$

or equivalently can be written as:

$$\begin{aligned} g_{X_4}(x_4) &= \sum_{\sim\{x_4\}} f(x_1, x_2, x_3, x_4, x_5, x_6) \\ &= \sum_{\sim\{x_4\}} f_1(x_2, x_5) f_2(x_1, x_3, x_6) f_3(x_1, x_4), \end{aligned}$$

As we said the factorizations are not unique, since by joining together the first two factors (i.e. $\tilde{f}_1(X_1, X_2, X_3, X_5, X_6) = f_1(X_2, X_5) f_2(X_1, X_3, X_6)$) we take another factorization, that is

$$f(X_1, X_2, X_3, X_4, X_5, X_6) = \tilde{f}_1(X_1, X_2, X_3, X_5, X_6) f_3(X_1, X_4),$$

with $\tilde{S}_1 = \{X_1, X_2, X_3, X_5, X_6\}$ and $\tilde{S}_2 = S_3 = \{X_1, X_4\}$. ■

**Definition 2.19** (**Acyclic factorization** [24])**.** A factorization $\prod_{k=1}^{K} f_k(S_k)$ of a function $f(X_1, ..., X_n)$ contains a cycle of length $L \geq 2$, if there exists a list of $L$ distinct couples $\{X_{i_1}, X_{i_2}\},\ \{X_{i_2}, X_{i_3}\}, ..., \{X_{i_L}, X_{i_1}\}$, and $L$ distinct variable subsets $S_{k_1}, ..., S_{k_L}$, such that $\{X_{i_l}, X_{i_{l+1}}\} \subseteq S_{k_l}$, for all $l \in \{1, ..., L\}$. We say that a factorization of a function is acyclic if it contains no cycles of any length $L \geq 2$. ▼

As we will see in latter sections, if a factorization has not cycles the algorithms that we will apply will converge to the exact marginal for every variable, in contrast with the case were a factorization has cycles, where the solution is not always exact. It is noted that in the case of a cyclic FG, the solutions of the FG algorithms are sometimes very close to the exact marginal. In example 2.1, both factorizations had no cycles. Below we present an example with a cyclic factorization of a function $f$.

**Example 2.2.** Consider the function of $n = 4$ variables:

$$f(X_1, X_2, X_3, X_4) = f_1(X_1, X_3) f_2(X_1, X_2, X_4) f_3(X_2, X_3),$$

with $S_1 = \{X_1, X_3\},\ S_2 = \{X_1, X_2, X_4\}$ and $S_3 = \{X_2, X_3\}$. This factorization contains a cycle of length $L = 3$ since $\{X_1, X_3\} \subseteq S_1,\ \{X_1, X_2\} \subseteq S_2$ and $\{X_2, X_3\} \subseteq S_3$. ■

**Definition 2.20** (**Connected factorization** [24])**.** A factorization $\prod_{k=1}^{K} f_k(S_k)$ of a function $f(X_1, ..., X_n)$ is said to be disconnected when we can group factors $\prod_{k=1}^{K} f_k(S_k) = f_A(S_A) f_B(S_B)$ such that $S_A \cup S_B = \{X_1, ..., X_n\}$ and $S_A \cap S_B = \emptyset$. When no such grouping is possible, the factorization is said to be connected. ▼

**Example 2.3.** Consider the function of $n = 4$ variables:

$$f(X_1, X_2, X_3, X_4) = f_1(X_1, X_3) f_2(X_2, X_4),$$

with $S_1 = \{X_1, X_3\}$ and $S_2 = \{X_2, X_4\}$. This factorization is disconnected since the intersection of the domains of local functions $f_1$ and $f_2$ is the empty set, i.e. $S_1 \cap S_2 = \emptyset$. ■

## 2.2 Factor graphs

In this section the provided definitions refer to global factorisable functions of input variables with finite domain. When we deal with continuous variables we utilize integrals instead of sums. Factor graphs (FGs) provide an efficient way (in terms of concurrency) to compute the marginals of a factorisable function using the sum-product algorithm (SPA). We will see that FGs can be simply constructed, while the SPA is more conceivable than other mathematical algorithms.

### 2.2.1 Introduction to factor graphs

Factor graphs are bipartite graphs that represent the factorization of a global function to smaller local functions, e.g. as in expression 2.5. More formally, we provide the definition below:

**Definition 2.21** (**Factor graph**)**.** Let $f(X_1, ..., X_n)$ be a decomposable function with $K$ factors, namely

$$f(X_1, ..., X_n) = \prod_{k=1}^{K} f_k(S_k).$$

The factor graph $G(V, E)$ corresponding to global function $f$ is a bipartite graph, where for every variable $X_i$, there is a variable node denoted with a solid circle, and for every factor $f_j$, there is a factor node denoted with a non-solid square. Furthermore, if variable $X_i$ is in the domain of factor $f_j$ an edge is created among them, namely $e_{ij} = (X_i, f_j)$. It is more convenient to write $X_i \in \mathcal{N}_G(f_j)$ or equivalently $f_j \in \mathcal{N}_G(X_i)$ to denote that variable $X_i$ is argument of factor $f_j$ or in "graph" words, variable node $X_i$ is adjacent with factor node $f_j$. $S_k$ stands for the subset of the variables of global function $f$ associated with local function $f_k$. ▼

For every factorization of function $f(X_1, ..., X_n)$ there is a unique factor graph $G(V, E)$ and vice versa, since the mapping between factorizations and factor graphs is one-to-one [24]. Since factor graphs are graphs, they may have cycles or they may have tree structure. This plays significant role for the convergence of the sum-product algorithm, the algorithm which we will encounter at the next section. Consider the following examples for clarification.

**Example 2.4.** Consider the function $f$ of the example 2.1, recall $f$ is given by

$$f(X_1, X_2, X_3, X_4, X_5, X_6) = f_1(X_2, X_5)f_2(X_1, X_3, X_6)f_3(X_1, X_4).$$

The corresponding factor graph is illustrated in figure 2.5. ■

**Example 2.5.** Let $f$ be a function of 3 variables,

$$f(X_1, X_2, X_3) = f_1(X_1, X_2)f_2(X_1, X_3)f_3(X_2, X_3).$$

The corresponding (cyclic) factor graph is depicted in figure 2.6. ■

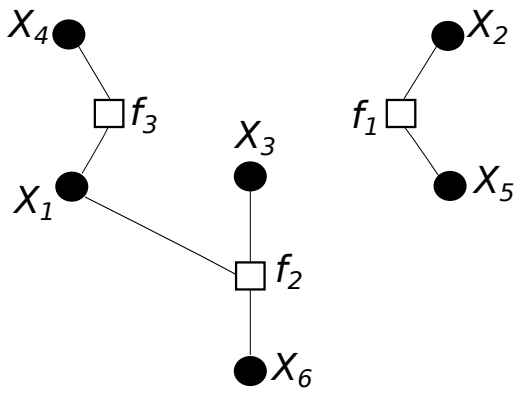



Figure 2.5: A disconnected, acyclic factor graph corresponding to global function of example 2.4. Notice that this graph satisfies the preconditions of a bipartite graph, since it has no odd-length cycles. The minimum cycle length is equal to 2, since it does not contains cycles (a graph with maximum cycle length equal to 2 is not cyclic, since a length 2 cycle consists of a single edge, i.e. is a trivial cycle).

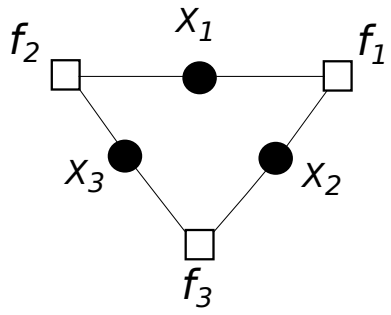



Figure 2.6: A cyclic factor graph corresponding to the global function of example 2.5.

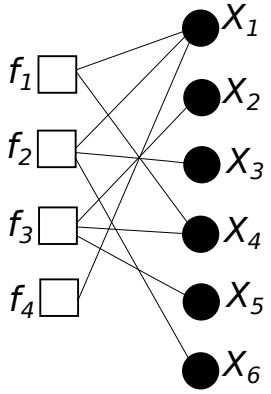



Figure 2.7: A factor graph which corresponds to the function of the example 2.6.

### 2.2.2 Marginals and distributive law

The computation of marginals of a function $f$, which is factorisable into smaller factors is fundamental. In high dimensional spaces the computation of a marginal function becomes intractable, especially when a few variables are continuous. *Generalized distributive law* (GLD) is a key tool which exploits the tree structure of the functions and computes the marginals efficiently. One simple example is the following:

$$(a+b)c = ac + bc \ , \tag{2.6}$$

where $a$, $b$, and $c$ are elements of an arbitrary field $\mathbb{F}$. The right-hand side of equation 2.6 involves three arithmetic operations (one addition and two multiplications), whereas the left-hand side needs only two.

Distributive law is examined from McEllie *et al.* in [11], which provides a general

framework for the marginalization of global function via graphical models. The following example illustrates the importance of GLD.

**Example 2.6.** Consider the acyclic and connected factorization of a function $f$ of 6 variables with identical (finite) domain, i.e. $\mathcal{X}_i = \mathcal{X},\ i = 1, ..., 6$:

$$f(X_1, X_2, X_3, X_4, X_5, X_6) = f_1(X_1, X_4) f_2(X_1, X_3, X_6) f_3(X_2, X_4, X_5) f_4(X_1).$$

The corresponding factor graph is given in figure 2.7. If we want to compute the marginal with respect to variable $X_3$, $g_{X_3}(x_3)$, following the definition 2.18, we have

$$g_{X_3}(x_3) = \sum_{\sim\{x_3\}} f_1(x_1, x_4) f_2(x_1, x_3, x_6) f_3(x_2, x_4, x_5) f_4(x_1). \tag{2.7}$$

The computation of the marginal above requires $\mathcal{O}(|\mathcal{X}|^6)$ operations since we sum over all possible values of the variables $X_1,\ X_2,\ X_4,\ X_5,\ X_6$, that is $\mathcal{O}(|\mathcal{X}|^5)$ operations for summations and $\mathcal{O}(|\mathcal{X}|)$ operations for the evaluation of the function $g_{X_3}(x_3)$ for every possible value of variable $X_3$, where $|\mathcal{X}|$ denotes the cardinality of set $\mathcal{X}$. If we apply the distributive law to the above expression, the resulting expression is

$$g_{X_3}(x_3) = \sum_{x_1} \left( f_4(x_1) \left( \sum_{x_6} f_2(x_1, x_3, x_6) \right) \left( \sum_{x_4} \left( f_1(x_1, x_4) \sum_{x_2, x_5} f_3(x_2, x_4, x_5) \right) \right) \right),$$

which is equivalent with

$$g_{X_3}(x_3) = \sum_{\sim\{x_3\}} \left( f_4(x_1) \left( \sum_{\sim\{x_1\}} f_2(x_1, x_3, x_6) \right) \left( \sum_{\sim\{x_1\}} f_1(x_1, x_4) f_3(x_2, x_4, x_5) \right) \right).$$

The last term in parenthesis requires $\mathcal{O}(|\mathcal{X}|^3)$ operations since it is a sum over variables $X_2, X_5$ and $X_4$. Therefore, accounting the first sum (over $X_1$), we totally take $\mathcal{O}(|\mathcal{X}|^4)$ arithmetic operations. Finally we want to evaluate marginal $g_{X_3}(x_3)$ for all possible values of $X_3$, which are $|\mathcal{X}|$ values. All these require $\mathcal{O}(|\mathcal{X}|^5)$ operations. Similarly the computation of the marginal with respect to variable $X_1$, $g_{X_1}(x_1)$, following the definition 2.18, is given by

$$g_{X_1}(x_1) = \sum_{\sim\{x_1\}} f_1(x_1, x_4) f_2(x_1, x_3, x_6) f_3(x_2, x_4, x_5) f_4(x_1),$$

if we apply the distributive law we take

$$g_{X_1}(x_1) = f_4(x_1)\left(\sum_{x_4} f_1(x_1, x_4)\left(\sum_{x_2,x_5} f_3(x_2, x_4, x_5)\right)\right)\left(\sum_{x_3,x_6} f_2(x_1, x_3, x_6)\right)$$

the equivalent expression is

$$g_{X_1}(x_1) = \left(\sum_{\sim\{x_1\}} f_4(x_1) f_1(x_1, x_4)\left(\sum_{\sim\{x_4\}} f_3(x_2, x_4, x_5)\right)\right)\left(\sum_{\sim\{x_1\}} f_2(x_1, x_3, x_6)\right)$$

which reduces the operations from $\mathcal{O}(|\mathcal{X}|^6)$ to $\mathcal{O}(|\mathcal{X}|^4)$. The conclusion is that the adjustment of the distributive law reduces significantly the computational complexity of marginals. ■

As an another example consider an arbitrary probability density function (pdf) in high-dimensional space, where we wish to efficiently calculate its marginals.

When a function via its factorization has a tree structure we are able to compute exactly a marginal with respect to any variable. Firstly, we construct a tree of the expression of the desired marginal, e.g. expression such as Eq. 2.7. That tree has as root the variable node corresponding to the desired marginal. An example of such a tree is depicted in figure 2.8. Notice that every internal node (either variable or factor) of such trees has $\geq 1$ child nodes or zero child nodes if it is leaf node. Moreover, notice that every node (either variable or factor) has only one parent node. Then a bottom-up procedure starts from the leaf nodes (variable nodes or factor nodes) by sending their messages to their parent nodes. In sequel, the latter send their messages to their parent nodes and so forth. When the root variable node is reached, the procedure terminates.

When the term "message" is referred, denotes the information passing along the edge corresponding to two neighboring nodes. Consider a node $i$ with $N-1$ child nodes and 1 parent node. The calculation of the outgoing message from node $i$ to its parent node $j$ requires the incoming messages from all child nodes arrived to node $i$.

The calculation of the outgoing messages of variable nodes differs from the calculation of the outgoing messages of factor nodes. Consider a variable node $X_i$ with $N$ adjacent (factor) nodes and the factor node $f_j$ which is the only parent node of variable node $X_i$ and obviously adjacent to it (i.e. $f_j \in \mathcal{N}(X_i)$). Then the outgoing message from variable

node $X_i$ to its parent factor node $f_j$, denoted by $\mu_{X_i \longrightarrow f_j}(x_i)$, and is given by

$$\mu_{X_i \longrightarrow f_j}(x_i) = \prod_{f_k \in \mathcal{N}(X_i) \setminus \{f_j\}} \mu_{f_k \longrightarrow X_i}(x_i), \tag{2.8}$$

i.e. it is the product of all incoming messages from all child factor nodes. Consider a factor node $f_j$ which has $M$ adjacent variable nodes $X_1, ..., X_M$, and a variable node $X_i$ which is the only parent variable node of factor node $f_j$ and obviously adjacent to it (i.e. $X_i \in \mathcal{N}(f_j) = \{X_1, ..., X_M\}$). Then the outgoing message from factor node $f_j$ to its parent variable node $X_i$, denoted by $\mu_{f_j \longrightarrow X_i}(x_i)$, and is given by

$$\mu_{f_j \longrightarrow X_i}(x_i) = \sum_{\sim \{x_i\}} f_j(x_1, ..., x_i, ..., x_M) \prod_{X_l \in \mathcal{N}(f_j) \setminus \{X_i\}} \mu_{X_l \longrightarrow f_j}(x_l), \tag{2.9}$$

i.e. it is the sum over all the configurations of the arguments of factor node $f_j$ for $X_i = x_i$, multiplied by the product of incoming messages from all variable child nodes.

Regarding leaf nodes, the message from a leaf variable node to its parent factor node is the constant 1, while the message from a leaf factor node to its parent variable node is equal to the value of its local function. This process is called *single sum-product algorithm*, since it computes the marginal of a single variable. An outgoing message from any node to its parent node can be computed only if all incoming messages from its child nodes are available.

To avoid the confusion of this procedure we provide the following example to make it clear.

**Example 2.7.** Consider the function $f(X_1, X_2, X_3, X_4, X_5)$ which is factorisable, and let variables $X_1, X_2, X_3, X_4, X_5$ having finite domain $\mathcal{X}_i,\ \forall i = 1, ..., 5$. More specifically,

$$f(X_1, X_2, X_3, X_4, X_5) = f_1(X_1, X_3) f_2(X_3, X_4, X_5) f_3(X_2, X_4) f_4(X_2),$$

the tree corresponding to the expression above is illustrated in figure 2.8. The marginal with respect to variable $X_2$ after the operation of distributive law follows:

$$g_{X_2}(x_2) = \sum_{x_1, x_3, x_4} \left( f_4(x_2) f_1(x_1, x_3) f_3(x_2, x_4) \left( \sum_{x_5} f_2(x_3, x_4, x_5) \right) \right). \tag{2.10}$$

Edges whose messages have been sent are depicted with purple color. Every message corresponding to a distinct step is illustrated with an arrow and an index inside a circle

associated with that step. In figure 2.9(a), the process starts from leaf nodes $f_4$, $X_1$ and $X_5$ which send their messages to their parent nodes $X_2$, $f_1$ and $f_2$, respectively. The leaf variable nodes $X_1$ and $X_4$ send to their parent factor nodes ($f_1$, $f_2$) the constant message 1, whereas the leaf factor node $f_4$ sends to its parent variable node $X_2$ the message of its factor $f_4(x_2)$. Variable node $X_2$ stays idle since the message from its child factor node $f_3$ is not available yet. Furthermore, the factor node $f_1$ is ready to send its message to its parent node $X_3$. Following the Eqs. 2.8 and 2.9 for variable nodes and factor nodes respectively, the upward messages calculated for all nodes of the tree. Figure 2.9 illustrates the message-passing for every distinct step. The process terminates when the root variable node $X_2$ has available all messages from its child factor nodes $f_4$ and $f_3$. The marginal with respect to (root) variable $X_2$ is depicted in figure 2.10. ■

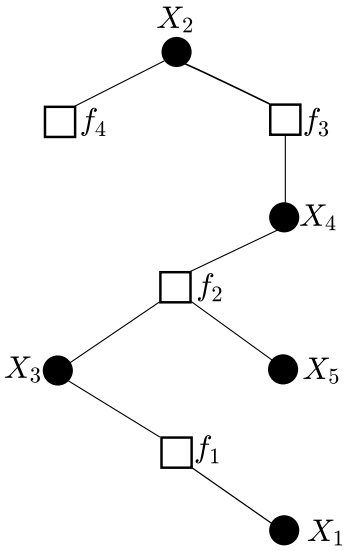



Figure 2.8: A factor graph which corresponds to the function of the example 2.7. This factor graph does not contain cycles therefore it has tree structure.

From the above example, it must be noted that when a factor node has degree 1, i.e. it is leaf factor node, the expression "sum over not a variable $X_i$ the factor $f_j$" (factor $f_j$ has only as argument the variable $X_i$, i.e. $f_j(X_i)$) is equivalent with the expression "$f_j(X_i)$". For the above example we have

$$f_4(x_2) = \sum_{\sim\{x_2\}} f_4(x_2).$$

Therefore we conclude that the leaf factor nodes send their functions to their parent variable nodes.

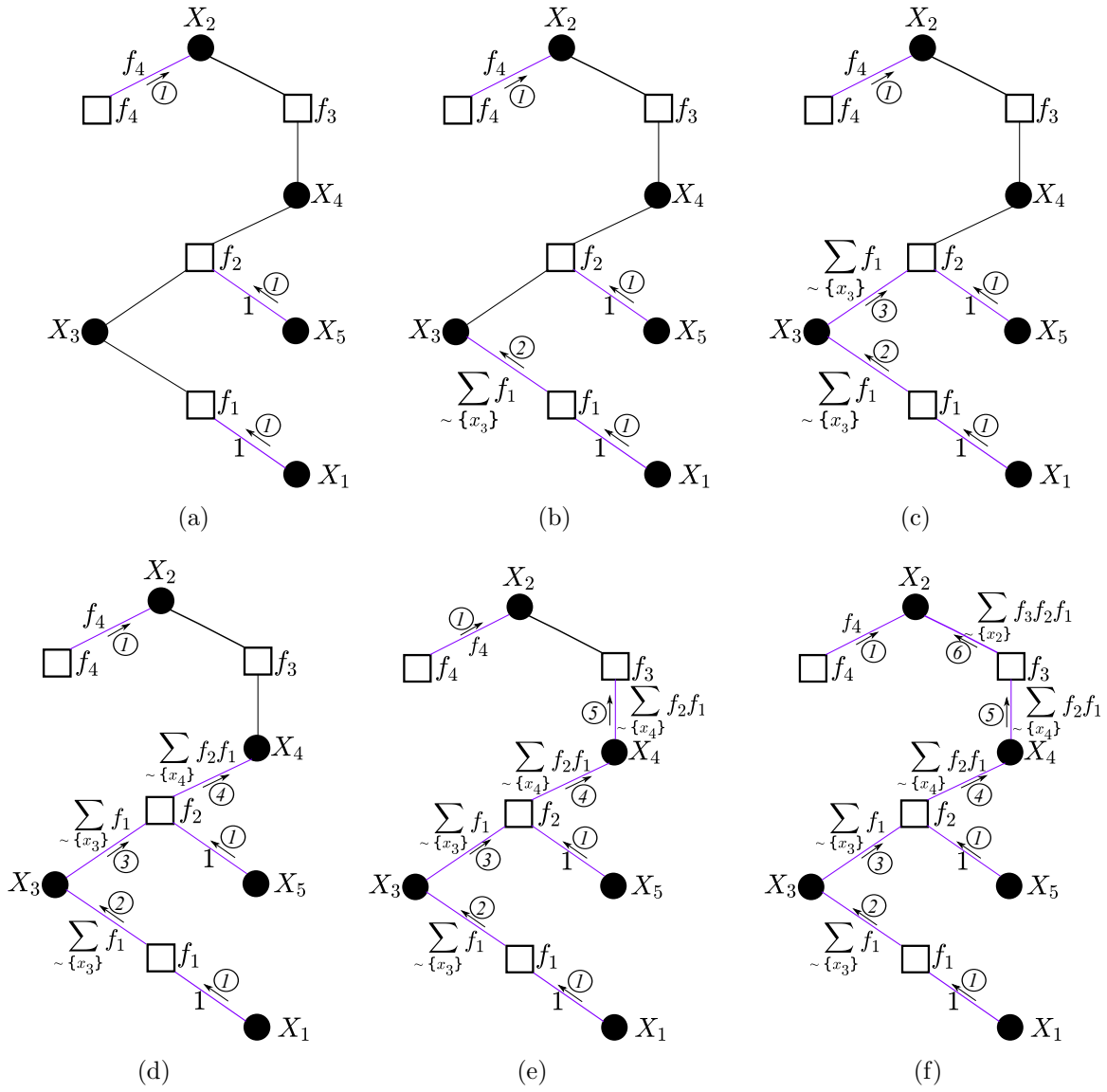



Figure 2.9: The message passing schedule of the tree expression of Eq. 2.10 .

This detailed example showed that the schedule for the calculation of the messages via message-passing process begins from the leaf nodes and continuous gradually until we reach the target variable node. When the messages from all child nodes arrive to a node, it is able to compute the external message to its parent node. When we reach the target root variable node we perform a simple multiplication of its incoming messages, in order

to calculate the desired marginal.

The process which described in the above example is utilized for the calculation of the marginal function with respect to a single variable. Usually we are interested to find all the marginal functions for all variables of the global function; this fact lead us to the following subsection which describes the sum-product algorithm, i.e. the concurrent calculation of all marginals of the global function's variables.

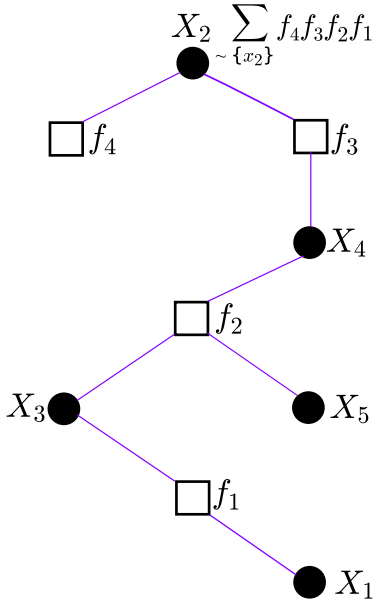



Figure 2.10: The product of the incoming incident messages from child nodes of variable node $X_2$ equals to the marginal function of variable $X_2$.

### 2.2.3 Sum-product algorithm

#### Discrete variables

This subsection provides a powerful algorithm for the efficient computation of the marginals of a global factorisable function, which consist of variables with finite domain. This algorithm is known as the *sum-product algorithm* (SPA).

In many circumstances we are interested in calculating the marginal of more than one variables. One possible solution for such calculation would be the creation of different trees for each variable and application of single sum-product algorithm. However, that approach would be very computation-expensive and thus, slow, especially for large number of variables.

The simultaneous computation of all marginal functions could be completed if we had computed all the messages passing across the edges of FG. The computation of the outgoing messages for SPA follows the same idea of the single sum-product algorithm. However, none of the nodes is considered as a root node, so there is not parent/child relationship among neighboring nodes. The sum-product algorithm terminates when all messages have been passed along the edges of the FG.

As we mentioned above, there are two kind of nodes, the factor nodes and the variable nodes. Suppose we have a variable $X_i$ with finite domain and a local function $f_j$.[2] Let $\mu_{X_i \longrightarrow f_j}(x_i)$ denotes the message from a variable node $X_i$ to a neighboring factor node $f_j$, and $\mu_{f_j \longrightarrow X_i}(x_i)$ denotes the message from factor node $f_j$ to variable node $X_i$. According to the sum-product algorithm the messages update rules have the following form:

- **variable node to local function node update rule :**

$$\mu_{X_i \longrightarrow f_j}(x_i) = \prod_{f_k \in \mathcal{N}(X_i) \backslash \{f_j\}} \mu_{f_k \longrightarrow X_i}(x_i), \tag{2.11}$$

- **local function node to variable node update rule :**

$$\mu_{f_j \longrightarrow X_i}(x_i) = \sum_{\sim \{x_i\}} \left( f_j(S_j = s_j) \prod_{X_l \in \mathcal{N}(f_j) \backslash \{X_i\}} \mu_{X_l \longrightarrow f_j}(x_l) \right), \tag{2.12}$$

where $S_j$ denotes the set of variables which are arguments in the local function $f_j$, i.e. $S_j = \{X_l : X_l \in \mathcal{N}(f_j)\}$. The backslash operator denotes the expression "except", namely the expression $X_l \in \mathcal{N}(f_j) \backslash \{X_i\}$ means "all variable nodes $X_l$ which are adjacent with factor node $f_j$ except variable node $X_i$". Similarly, $f_k \in \mathcal{N}(X_i) \backslash \{f_j\}$ are all neighboring to $X_i$ factors nodes, except $\{f_j\}$.

The initializing phase is the same with the single-SPA, that is, for every factor node $f_j$ which is leaf node, the message to its neighboring variable node ($X_m \in \mathcal{N}(f_j)$) is $\mu_{f_j \longrightarrow X_m} = f_j(x_m)$, similarly, the message from every leaf variable node $X_i$ to its neighboring factor node ($f_l \in \mathcal{N}(X_i)$) is $\mu_{X_i \longrightarrow f_l}(x_i) = 1$.

Every marginal $g_{X_i}(x_i)$ of a variable $X_i$ is the product of all incoming messages incident

[2]The term local function is equivalent with the term factor and the term local factor.

to variable node $X_i$ (if all the latter are available), i.e.

$$g_{X_i}(x_i) = \prod_{f_k \in \mathcal{N}(X_i)} \mu_{f_k \longrightarrow X_i}(x_i). \tag{2.13}$$

Similarly, if we want to find the marginal with respect to a cluster of variables $S_j$ ($S_j$ is a subset of variables which are arguments of the global function) corresponding to a factor $f_j$ (i.e. $S_j = \{X_l : X_l \in \mathcal{N}(f_j)\}$), we must multiply the incoming messages incident to the factor node $f_j$ with the factor itself ($f_j$), i.e.

$$g_{S_j}(s_j) = f_j(s_j) \prod_{X_l \in \mathcal{N}(f_j)} \mu_{X_l \longrightarrow f_j}(x_l). \tag{2.14}$$

Figure 2.12 depicts the sum-product algorithm update rule for factor nodes, while the figure 2.11 illustrates the sum-product algorithm update rule for variable nodes. Finally, 2.13 shows the calculation of the marginal for a single variable. The calculation of marginal applied at every FG node at the last step of SPA (after the calculation of all messages on the edges of FG). The calculation of the messages at every step of the SPA operates concurrently [3]. The concurrent operation of SPA will be shown with an example, subsequently.

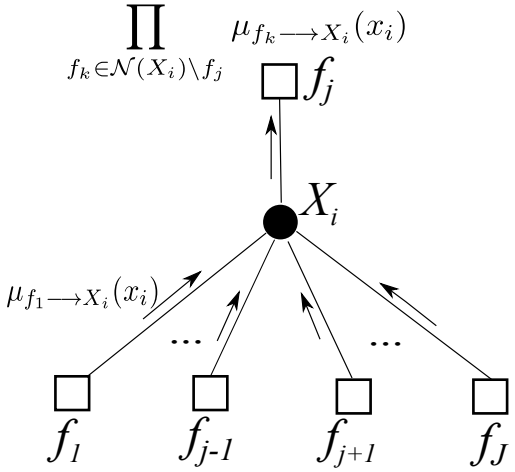



Figure 2.11: Update rule of the sum-product algorithm for a variable node. The variable node has $J$ neighboring nodes. The external message for factor node $f_j$ is equal with the product of the incoming messages incident to variable node $X_i$.

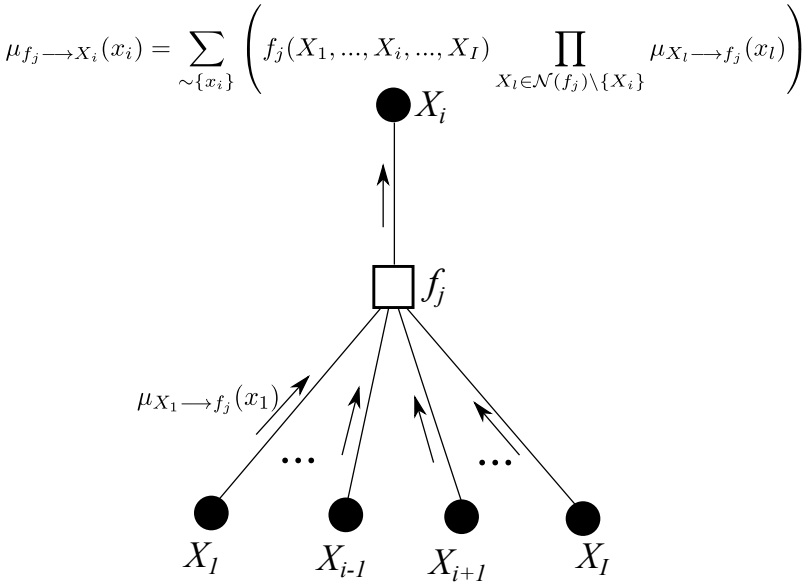



Figure 2.12: Update rule of the sum-product algorithm for a factor node. The factor node has $I$ neighboring nodes. The external (outgoing) message of the factor node $f_j$ corresponds to the variable node $X_i$, therefore the summation is over not variable $X_i$.

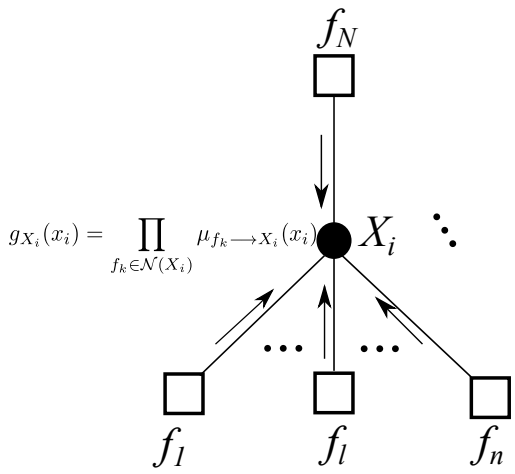



Figure 2.13: The marginal function for a variable node $X_i$ with $N$ adjacent factor nodes. The computation of the marginal with respect to variable $X_i$ is the product of all incoming messages from $N$ factor nodes to variable node $X_i$.

Two important observations are, firstly, that every variable node of degree $D$ must perform $D-1$ multiplications and secondly, that the degree of leaf nodes is 1. Furthermore,

note that a variable node of degree 2 simply propagates its incoming message to the factor node corresponding to the outgoing edge.

Below, we present a detailed example in order to clarify the SPA rules and the message-passing schedule.

**Example 2.8.** Consider the factorization of function $f$ of example 2.7, i.e.

$$f(X_1, X_2, X_3, X_4, X_5) = f_1(X_1, X_3) f_2(X_3, X_4, X_5) f_3(X_2, X_4) f_4(X_2),$$

the corresponding factor graph for the above factorization is given in figure 2.8 and it has tree structure. If we perceive this factor graph as a graph without cycles and not as a tree with root, then the factor graph has the structure depicted in figure 2.14. For convenience, we will apply sum-product algorithm in distinct steps. At every step we will derive the corresponding messages.

- Step 1:

$$\begin{aligned}
\mu_{f_4 \longrightarrow X_2}(x_2) &= \sum_{\sim\{x_2\}} f_4(x_2) = f_4(x_2), \\
\mu_{X_1 \longrightarrow f_1}(x_1) &= 1, \\
\mu_{X_5 \longrightarrow f_2}(x_5) &= 1.
\end{aligned}$$

  The messages of this step are depicted in figure 2.15(a).

- Step 2:

$$\begin{aligned}
\mu_{f_1 \longrightarrow X_3}(x_3) &= \sum_{\sim\{x_3\}} f_1(x_1, x_3) \mu_{X_1 \longrightarrow f_1}(x_1), \\
\mu_{X_2 \longrightarrow f_3}(x_2) &= \mu_{f_4 \longrightarrow X_2}(x_2).
\end{aligned}$$

  Note that factor node $f_2$ has not received the incoming message from variable node $X_3$, in order to compute the outgoing message for variable nodes $X_5$ and $X_4$. Similarly, the incoming message from variable node $X_4$ is not available yet, therefore, $f_2$ can not send an outgoing message to variable nodes $X_5$ and $X_3$. Consequently, factor node $f_2$ at this step remains idle. The messages of this step are showed in figure 2.15(b).

- Step 3:

$$\mu_{X_3 \longrightarrow f_2}(x_3) = \mu_{f_1 \longrightarrow X_3}(x_3),$$
$$\mu_{f_3 \longrightarrow X_4}(x_4) = \sum_{\sim\{x_4\}} f_3(x_2, x_4)\mu_{X_2 \longrightarrow f_3}(x_2).$$

Factor node $f_2$ remains silent since its incoming messages have not arrived yet. Figure 2.15(c) illustrates the messages of this step.

Step 4:

$$\mu_{X_4 \longrightarrow f_2}(x_4) = \mu_{f_3 \longrightarrow X_4}(x_4),$$
$$\mu_{f_2 \longrightarrow X_4}(x_4) = \sum_{\sim\{x_4\}} f_2(x_3, x_4, x_5)\big(\mu_{X_5 \longrightarrow f_2}(x_5)\mu_{X_3 \longrightarrow f_2}(x_3)\big).$$

At this step factor node $f_2$ can calculate the outgoing message to variable node $X_4$, since the corresponding incoming messages from variables nodes $X_5, X_3$ have arrived (at step 1 and at step 3, respectively). The outgoing messages for variable nodes $X_3$ and $X_5$ are not ready to be computed yet. Consequently these 2 variable nodes must wait one step more. The messages of this step are depicted in figure 2.15(d).

Step 5:

$$\mu_{X_4 \longrightarrow f_3}(x_4) = \mu_{f_2 \longrightarrow X_4}(x_4),$$
$$\mu_{f_2 \longrightarrow X_3}(x_3) = \sum_{\sim\{x_3\}} f_2(x_3, x_4, x_5)\big(\mu_{X_5 \longrightarrow f_2}(x_5)\mu_{X_4 \longrightarrow f_2}(x_2)\big),$$
$$\mu_{f_2 \longrightarrow X_5}(x_5) = \sum_{\sim\{x_5\}} f_2(x_3, x_4, x_5)\big(\mu_{X_4 \longrightarrow f_2}(x_4)\mu_{X_3 \longrightarrow f_2}(x_3)\big).$$

All the outgoing messages from factor node $f_2$ are calculated, due to all its incoming messages are available. The messages corresponding to this step are illustrated in figure 2.15(e).

Step 6:

$$\mu_{X_3 \longrightarrow f_1}(x_3) = \mu_{f_2 \longrightarrow X_3}(x_3),$$
$$\mu_{f_3 \longrightarrow X_2}(x_2) = \sum_{\sim\{x_2\}} f_3(x_2, x_4)\mu_{X_4 \longrightarrow f_3}(x_4).$$

The messages of this step are depicted in figure 2.15(f).

Step 7:

$$\mu_{X_2 \longrightarrow f_4}(x_2) = \mu_{f_3 \longrightarrow X_2}(x_2),$$
$$\mu_{f_1 \longrightarrow X_1}(x_1) = \sum_{\sim\{x_1\}} f_1(x_1, x_3)\mu_{X_3 \longrightarrow f_1}(x_3).$$

The messages of the final step are illustrated in figure 2.15(g).

Termination:

$$g_{X_1}(x_1) = \mu_{f_1 \longrightarrow X_1}(x_1),$$
$$g_{X_2}(x_2) = \mu_{f_4 \longrightarrow X_2}(x_2)\mu_{f_3 \longrightarrow X_2}(x_2),$$
$$g_{X_3}(x_3) = \mu_{f_1 \longrightarrow X_3}(x_3)\mu_{f_2 \longrightarrow X_3}(x_3),$$
$$g_{X_4}(x_4) = \mu_{f_2 \longrightarrow X_4}(x_4)\mu_{f_3 \longrightarrow X_4}(x_4),$$
$$g_{X_5}(x_5) = \mu_{f_2 \longrightarrow X_5}(x_5).$$

In the last step we calculate concurrently the marginals with respect to all variables of function $f$. The calculation of the marginal is just the product of all the incoming messages for every variable node. ■

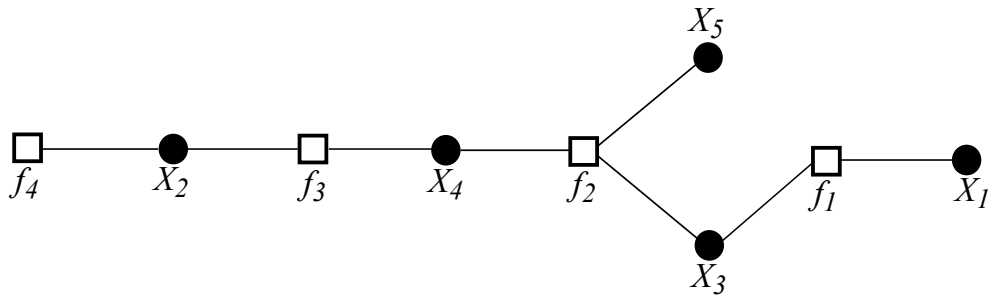
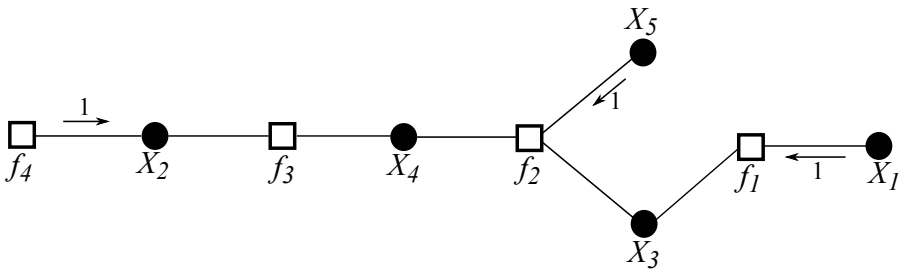
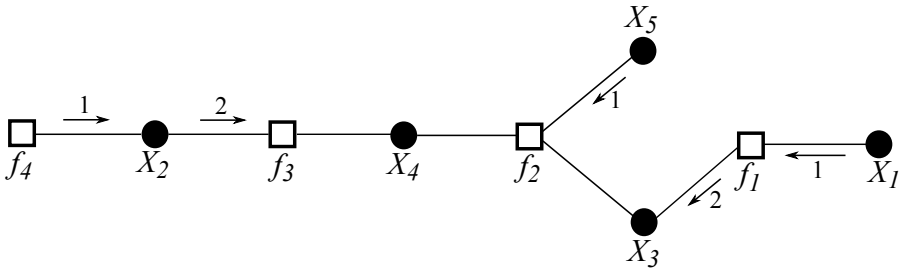


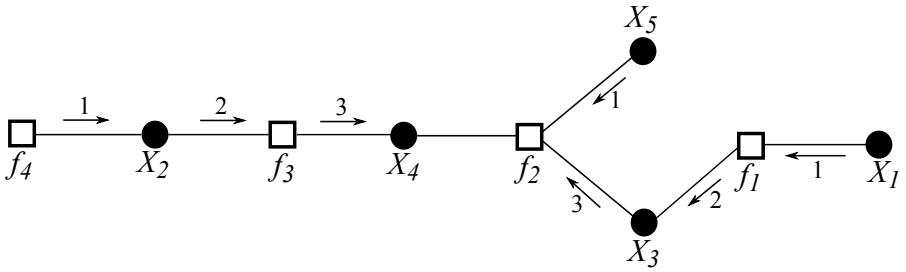

Figure 2.14: Factor graph of the global function of example 2.8. This factor graph does not contain cycles. The leaf nodes of the graph are the variable nodes $X_1, X_5$ and the factor node $f_4$. This factor graph is equivalent with the factor graph of figure 2.8.


$X_5$
1
$f_4$
$X_2$
$f_3$
$X_4$
$f_2$
$f_1$
1
$X_1$
$X_3$

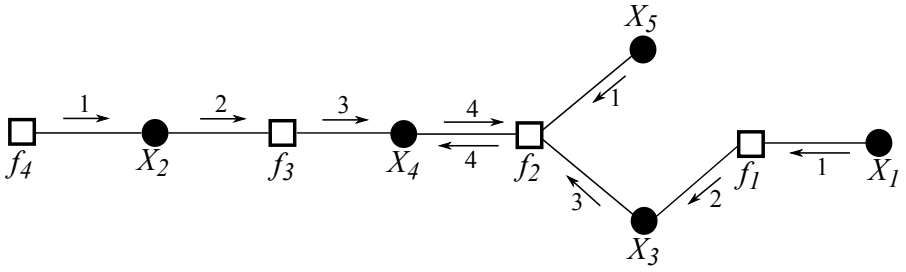
(a)


$X_5$
1
2
$f_4$
$X_2$
$f_3$
$X_4$
$f_2$
$f_1$
1
$X_1$
2
$X_3$

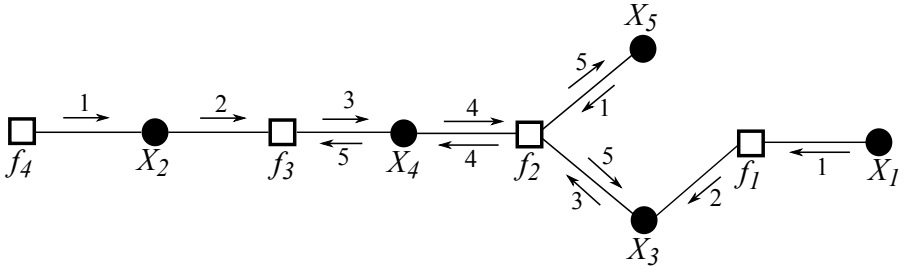
(b)


$X_5$
1
2
3
$f_4$
$X_2$
$f_3$
$X_4$
$f_2$
$f_1$
1
$X_1$
3
2
$X_3$

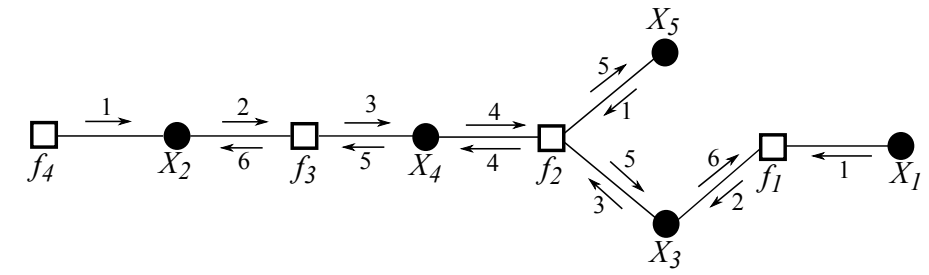
(c)


$X_5$
1
2
3
4
$f_4$
$X_2$
$f_3$
$X_4$
4
$f_2$
$f_1$
1
$X_1$
3
2
$X_3$

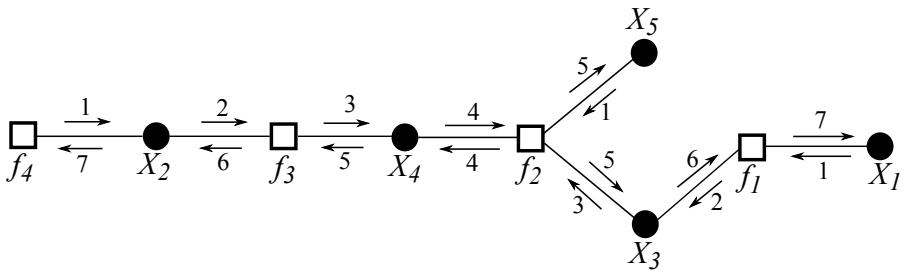
(d)


$X_5$
5
1
2
3
4
$f_4$
$X_2$
$f_3$
5
$X_4$
4
$f_2$
5
$f_1$
1
$X_1$
3
2
$X_3$

(e)




(f)




(g)

Figure 2.15: Sum-product algorithm, message schedule.

The above example clarifies the message-passing scheduling of the sum-product algorithm and gives intuition of how messages are exchanged as the algorithm execution progresses.

### Continuous variables

When we are dealing with functions whose variables are defined over continuous domain, the only change the sum-product algorithm must perform is the utilization of integrals instead of sums. More precisely, if we consider that we have a factorisable function $f$ which has arguments continuous variables, the update rules of the sum-product algorithm are the following:

- **variable node to local function node update rule :**

$$\mu_{X_i \longrightarrow f_j}(x_i) = \prod_{f_k \in \mathcal{N}(X_i) \setminus \{f_j\}} \mu_{f_k \longrightarrow X_i}(x_i), \tag{2.15}$$

- **local function node to variable node update rule :**

$$\mu_{f_j \longrightarrow X_i}(x_i) = \int \left( f_j(S_j = s_j) \prod_{X_l \in \mathcal{N}(f_j) \setminus \{X_i\}} \mu_{X_l \longrightarrow f_j}(x_l) \right) \{\sim \mathrm{d}x_i\}, \tag{2.16}$$

  where the symbol $\{\sim \mathrm{d}x_i\}$ denotes that we integrate over all variables which are arguments in factor $f_j$ except variable $X_i$.

The calculation of the marginal with respect to a variable is the same with the case of discrete variables, namely

$$g_{X_i}(x_i) = \prod_{f_k \in \mathcal{N}(X_i)} \mu_{f_k \longrightarrow X_i}(x_i), \tag{2.17}$$

where $g_{X_i}(x_i)$ stands for the marginal function associated with the variable $X_i$. Finally the calculation of the marginal with respect to a subset of variables which are arguments of a local factor $f_j$ is given:

$$g_{S_j}(s_j) = f_j(s_j) \prod_{X_l \in \mathcal{N}(f_j)} \mu_{X_l \longrightarrow f_j}(x_l), \tag{2.18}$$

where $S_j$ denotes the variables associated with the variable nodes which are adjacent with factor node $f_j$, namely $S_j = \{X_l : X_l \in \mathcal{N}(f_j)\}$.

### 2.2.4 Normal factor graphs

Normal factor graphs were introduced by Forney in [8] and they are an equivalent graph expression of a factorisable function, like the FGs use have studied so far. The only difference between normal factor graphs and (simple) factor graphs, is that in normal factor graphs there are no specific nodes for variables, whereas the edges of the graph correspond to the variables of the global function; in normal factor graphs all nodes correspond to factors. There are 2 kinds of factors, the factors of the factorization of the global function and the equality factors which equate the edges of the variables with cardinality more than 2. When a given factor graph is asked to be transformed to the equivalent normal factor graph, we follow the rules below:

- for every variable node of degree 1, we create an edge and we remove that node. See figure 2.16;

- for every variable node of degree 2, we create an edge among the 2 factor nodes, and we remove that node. See figure 2.17;
- for every variable node $X_i$ of degree 3 or more we create an equality factor node which is denoted $f_=$ (same notation as in [9]) and signifies that any incident (variable) edge is equal to the variable $X_i$. Therefore if we have a variable $X_i$ with degree $N > 2$ we replace the variable node with the factor node $f_=(X_i^{(1)}, ..., X_i^{(N)})$. The variables $X_i^{(1)}, ..., X_i^{(N)}$ are said dummy variables. See figure 2.17.

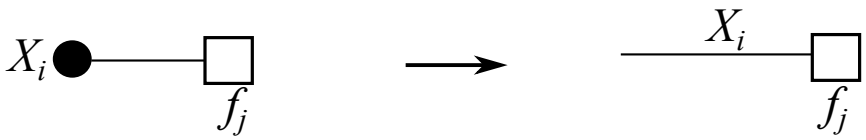



Figure 2.16: Convert a degree 1 variable node into the corresponding normal factor graph edge variable.

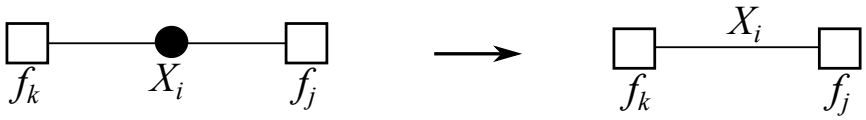



Figure 2.17: Convert a degree 2 variable node into the corresponding normal factor graph edge variable.

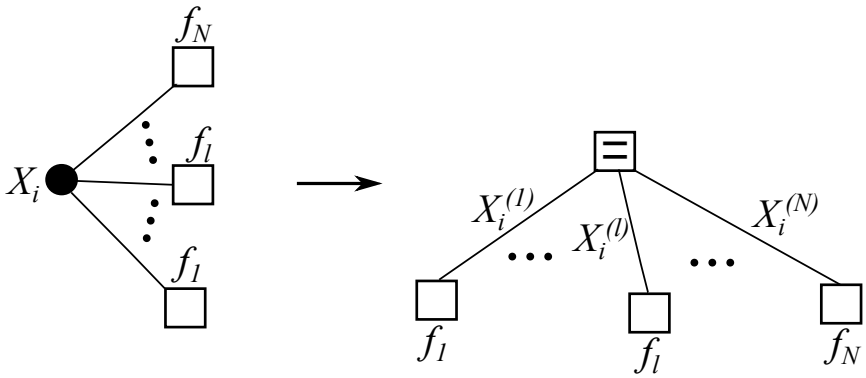



Figure 2.18: Convert a degree $N$ ($> 2$) variable node into the corresponding normal factor graph equality factor node.

More specifically, the equality factor of $N$ variables is defined as:

$$f_{=}(X_1, ..., X_N) = \prod_{k=1}^{N-1} \delta(X_{k+1} - X_k) \tag{2.19}$$

where $\delta(\cdot)$ stands for the Dirac delta function when the variables $X_i$ are continuous, while in the case where variables $X_i$ are discrete, denotes the Kronecker delta function.

Two examples below follow:

**Example 2.9.** Consider the factorization of function of the example 2.8, namely

$$f(X_1, X_2, X_3, X_4, X_5) = f_1(X_1, X_3) f_2(X_3, X_4, X_5) f_3(X_2, X_4) f_4(X_2),$$

the factor graph of the above expression is showed in figure 2.14. Following the rules above, the resulting normal factor graph illustrated in figure 2.19. ■

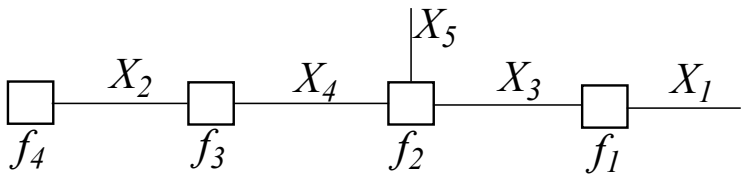



Figure 2.19: The normal factor graph of the factorization of function of the example 2.9.

**Example 2.10.** Consider the following factorization of a function $f$:

$$f(X_1, X_2, X_3, X_4, X_5) = f_1(X_1, X_3, X_5) f_2(X_1, X_2) f_3(X_1, X_3, X_4) f_4(X_3),$$

suppose variables $X_i,\ i = 1, ..., 5$, have finite domain. The factor graph of the above expression is illustrated in figure 2.20, while the corresponding normal factor graph is illustrated in figure 2.21. Function $f$ can be factorized in terms of normal factor graphs as:

$$\begin{aligned} f(X_1, X_2, X_3, X_4, X_5) =& f_1(X_1, X_3^{(1)}, X_5) f_2(X_1^{(2)}, X_2) f_3(X_1^{(1)}, X_3, X_4) f_4(X_3^{(2)}) \times \\ & \times \underbrace{\delta(X_1 - X_1^{(1)})\delta(X_1^{(1)} - X_1^{(2)})}_{f_{=}^5(X_1, X_1^{(1)}, X_1^{(2)})} \underbrace{\delta(X_3 - X_3^{(1)})\delta(X_3^{(1)}, X_3^{(2)})}_{f_{=}^6(X_3, X_3^{(1)}, X_3^{(2)})}, \end{aligned}$$

where $\delta(\cdot)$ denotes the delta Kronecker function. ■

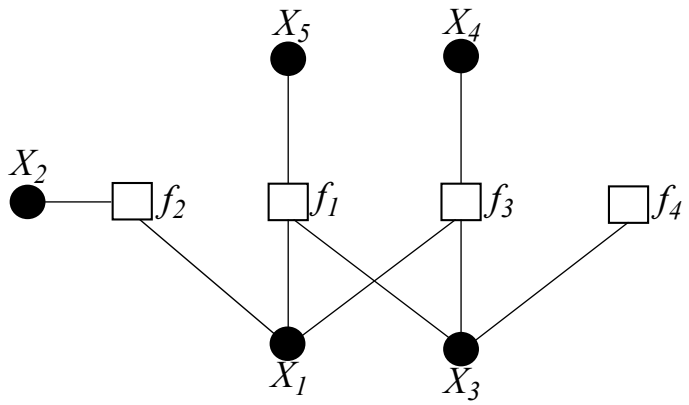



Figure 2.20: The factor graph of example 2.10.

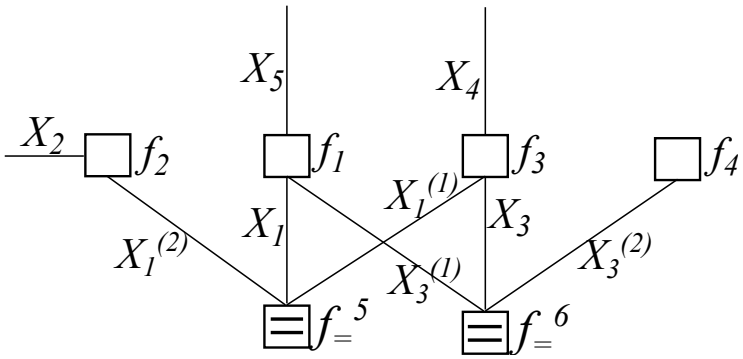



Figure 2.21: The normal factor graph of the example 2.10.

### 2.2.5 Sum-product algorithm for normal factor graphs

Starting from equality nodes, suppose an equality node $f^k_=(X_1, ..., X_i, ..., X_N)$ with degree $N$, then the outgoing message along the edge corresponding to the variable $X_i$ is given by

$$\begin{aligned}\mu_{f^k_= \longrightarrow X_i}(x_i) &= \sum_{\sim\{x_i\}} f^k_=(X_1, ..., X_i, ..., X_N) \prod_{X_n \in \mathcal{N}(f^k_=)\setminus X_i} \mu_{X_n \longrightarrow f^k_=}(x_n) \\ &= \prod_{X_n \in \mathcal{N}(f^k_=)\setminus X_i} \mu_{X_n \longrightarrow f^k_=}(x_i), \end{aligned} \tag{2.20}$$

Note that a message along an edge does not change. To see that, consider 2 factor nodes $f_k$ and $f_j$ which both of them have as argument the variable $X_i$, then $\mu_{f_k \longrightarrow X_i}(x_i) = \mu_{X_i \longrightarrow f_j}(x_i)$ and $\mu_{f_j \longrightarrow X_i}(x_i) = \mu_{X_i \longrightarrow f_k}(x_i)$. See figure 2.22.

The initialization of the algorithm is the same as in the case of factor graphs. Namely,

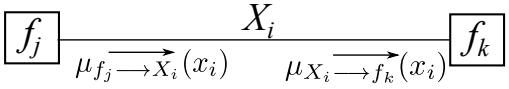



Figure 2.22: The message $\mu_{f_j \longrightarrow X_i}(x_i)$ does not change along the edge $X_i$.

- every variable $X_i$ of degree 1 (i.e. a half edge) which is adjacent with a factor node $f_j$ transmits along the edge $X_i$ the message 1 ($\mu_{X_i \longrightarrow f_j}(x_i) = 1$),
- every leaf node $f_j$ with argument the variable $X_i$ transmits along the edge $X_i$ the message $f_j(x_i)$ ($\mu_{f_j \longrightarrow X_i}(x_i) = f_j(x_i)$).

Since we have only factor nodes, the update rule for a factor node $f_j$ of degree $N$ is given by

$$\mu_{f_j \longrightarrow X_i}(x_i) = \sum_{\sim\{x_i\}} \left( f_j\left(S_j = s_j\right) \prod_{X_l \in \mathcal{N}(f_j)\setminus\{X_i\}} \mu_{X_l \longrightarrow f_j}(x_l) \right), \tag{2.21}$$

where $S_j$ stands for the set of variables which are arguments in the local function $f_j$, namely $S_j = \{X_l : X_l \in \mathcal{N}(f_j)\}$ and $|S_j| = N$. The messages $\mu_{X_l \longrightarrow f_j}(x_l)$ are from factor nodes which have in common with factor $f_j$ the variable $X_l$.

Finally, if we want to compute the marginal with respect to a variable $X_i$, we must perform a simple multiplication of the two opposite directional messages passing through edge $X_i$ from any neighboring factor node $f_j$ which contains as argument the variable $X_i$, consequently, the marginal of variable $X_i$ is given by

$$g_{X_i}(x_i) = \mu_{f_j \longrightarrow X_i}(x_i)\mu_{X_i \longrightarrow f_j}(x_i) \tag{2.22}$$

where $f_j$ is any factor which has argument the variable $X_i$. Consider the example 2.10, then the marginal of variable $X_1$ can be calculated (with several ways) as follows:

$$\begin{aligned} g_{X_1}(x_1) &= \mu_{f_2 \longrightarrow X_1^{(2)}}(x_1^{(2)}) \times \mu_{X_1^{(2)} \longrightarrow f_2}(x_1^{(2)}) \\ &= \mu_{f_3 \longrightarrow X_1^{(1)}}(x_1^{(1)}) \times \mu_{X_1^{(1)} \longrightarrow f_3}(x_1^{(1)}) \\ &= \mu_{f_1 \longrightarrow X_1}(x_1) \times \mu_{X_1 \longrightarrow f_1}(x_1) \\ &= \mu_{f_=^5 \longrightarrow X_1^{(1)}}(x_1^{(1)}) \times \mu_{X_1^{(1)} \longrightarrow f_=^5}(x_1^{(1)}). \end{aligned}$$

The careful reader can note that the message update rules of the normal factor graphs

are very similar with the case of the (simple) factor graphs.

### 2.2.6 Variations of sum-product algorithm

In this subsection, we will give a brief description of the notion of groups, fields and semi-rings. We will provide some basic definitions for the above topics, in order to better understand some variations of the sum-product algorithm.

**Definition 2.22** (**Group** [25])**.** A set of elements $\mathcal{A} = \{a, b, c, ...\}$ and an operator $\oplus$ form a group and the following axioms hold:

- Closure: for any $a \in \mathcal{A}$, $b \in \mathcal{A}$, the element $a \oplus b$ is in $\mathcal{A}$,
- Associativity: for any $a, b, c \in \mathcal{A}$, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$,
- Identity: there is an identity element 0 in $\mathcal{A}$ for which $a \oplus 0 = 0 \oplus a = a$ for all $a \in \mathcal{A}$
- Inverse: for each $a \in \mathcal{A}$ there is an inverse $(-a) \in \mathcal{A}$ such that $a \oplus (-a) = 0$.

▼

Furthermore, we will assume that for every group $\mathcal{A}$ and 2 elements of $\mathcal{A}$, $a, b$, the commutative property: $a \oplus b = b \oplus a$. If this axiom holds, we say that group $\mathcal{A}$ is an *abelian* group.

**Definition 2.23** (**Field** [25])**.** A field is a set $\mathbb{F}$ of at least two elements, with two operations $\oplus$ and $*$ ($\times$), for which the following axioms are satisfied:

- The set $\mathbb{F}$ forms an abelian group (whose identity is called 0) under the operation $\oplus$,
- The set $\mathbb{F}^* = \mathbb{F} - \{0\} = \{a \in \mathbb{F}, a \neq 0\}$ forms an abelian group (whose identity is called 1) under the operation $*$.
- Distributive law: For all $a, b, c \in \mathbb{F}$, $(a \oplus b) * c = (a * c) \oplus (b * c)$.

▼

The operation $\oplus$ is called addition and often denoted as $+$, while the operation $*$ is called multiplication. The multiplication has greater priority than addition. Ring has the same axioms as the field; the only difference is that rings have no multiplication inverse.

We define a triplet $(\mathbb{F}, \oplus, \otimes)$ and we say that this triplet forms a semi-ring if the following properties hold [24]:

- $\oplus$ is associative and commutative, i.e. there exist an identity element for $\oplus$, denoted as $0_{\oplus}$,
- $\otimes$ is associative and commutative, i.e. there exist an identity element for $\otimes$, denoted as $1_{\otimes}$,
- $\otimes$ is distributive over $\oplus$.

$\mathbb{F}$ can be any abstract field. In comparison to a field $\mathbb{F}$ we do not require the existence of an inverse with respect to either operation $(+, *)$. This small example below illustrates the max-product semi-ring.

**Example 2.11** ([24])**.** Consider the triplet $(\mathbb{R}, \max, *)$. This triplet forms a semi-ring since

- max is associative,

$$\max\{a, \max\{b, c\}\} = \max\{\max\{a, b\}, c\},$$

- max is commutative,

$$\max\{a, b\} = \max\{b, a\},$$

- There exist an identity element for max, $0_{\oplus}$, such that $\max\{x, 0_{\oplus}\} = x,\ \forall x \in \mathbb{R}$. The element that satisfies this condition is the $-\infty = 0_{\oplus}$,
- $*$ is associative,
- $*$ is commutative,
- There exist an identity element for $*$, $1_{\otimes}$, such that $1_{\otimes} * x = x,\ \forall x \in \mathbb{R}$. The element that satisfies this condition is the number 1.

■

Notice that the sum-product algorithm is associated with the semi-ring $(\mathbb{R}, +, *)$. One variation of sum-product algorithm is the max-product algorithm which is associated with the semi-ring $(\mathbb{R}, \max, *)$. Another one variation of sum-product algorithm is the min-sum algorithm which is associated with the semi-ring $(\mathbb{R}, \min, +)$. Consequently, we can derive the update rules of the max-product algorithm.

Consider a factorisable function $f$, where its arguments are variables with finite domain. Let $X_i$ be an arbitrary variable node of the factor graph corresponding to global function $f$, and let $f_j$ be an arbitrary factor node neighboring to variable node $X_i$. Then the update rules for max-product algorithm are the following:

- **variable node to local function node update rule:**

$$\mu_{X_i \longrightarrow f_j}(x_i) = \prod_{f_k \in \mathcal{N}(X_i) \setminus \{f_j\}} \mu_{f_k \longrightarrow X_i}(x_i), \tag{2.23}$$

- **local function node to variable node update rule:**

$$\mu_{f_j \longrightarrow X_i}(x_i) = \max_{\sim \{x_i\}} \left\{ f_j \left(S_j = s_j\right) \prod_{X_l \in \mathcal{N}(f_j) \setminus \{X_i\}} \mu_{X_l \longrightarrow f_j}(x_l) \right\}, \tag{2.24}$$

where $S_j$ stands for the set of variables which are arguments of the local function $f_j$, i.e. $S_j = \{X_l : X_l \in \mathcal{N}(f_j)\}$, and the symbol $\sim \{x_i\}$ under maximum denotes that we maximize over all variables $X_l \in S_j$ except variable $X_i$.

The initialization phase is a little different of that in sum-product algorithm, since variable leaf nodes send to their (unique) neighboring factor nodes the constant message $-\infty$. More specifically, the initialization rules for max-product algorithm are the following:

- for every factor node $f_j$ which is leaf node, the message to its neighboring variable node ($X_m \in \mathcal{N}(f_j)$) is equal to $\mu_{f_j \longrightarrow X_m} = f_j(x_m)$,
- the messages from every leaf variable node $X_i$ to its neighboring factor node ($f_l \in \mathcal{N}(X_i)$) is equal to $\mu_{X_i \longrightarrow f_l}(x_i) = -\infty$.

Every marginal $g_{X_i}(x_i)$ with respect to variable $X_i$ is the product of all incoming messages, if all the latter are available, i.e.

$$g_{X_i}(x_i) = \prod_{f_k \in \mathcal{N}(X_i)} \mu_{f_k \longrightarrow X_i}(x_i), \tag{2.25}$$

Similarly if we want to find the marginal with respect to a cluster of variables $S_j$ ($S_j$ is a subset of variables which are arguments of the global function) corresponding to a factor $f_j$, we are multiplying the incoming messages incident to that local factor node with the

local factor itself, i.e.

$$g_{S_j}(s_j) = f_j(s_j) \prod_{X_l \in \mathcal{N}(f_j)} \mu_{X_l \longrightarrow f_j}(x_l), \tag{2.26}$$

### 2.2.7 Cyclic factorizations

Finally, we must consider the case when a factor graph of a factorisable function has cycles, where the sum-product algorithm may fail to terminate, and when it terminates, usually, the final solution is not the exact one, i.e. the solution diverges from true marginal function.

That happens because the scheduling of messages for the sum-product algorithm for these graphs is difficult. Sometimes the dependencies on a cycle may lead the sum-product algorithm to avoid the initialization phase. Another issue which sum-product algorithm sometimes encounters, is that it may run infinitely. One solution for this problem is to terminate the algorithm after a predetermined number of iterations. However, the number of required iterations until the algorithm converges to a satisfiable solution, is usually unknown.

When it is given a factor graph with cycles and we desire to compute the marginals, we apply the sum-product algorithm, as described in above subsection, but we follow a schedule which usually depends on the problem and the structure of the graph. Even though the sum-product algorithm and its variations encounter difficulties in graphs with cycles, the most famous algorithms in signal processing and coding theory use iterative versions of sum-product algorithm on factor graphs with cycles [1], [12], [18].

# Chapter 3

# Factor Graph Applications: Coding Theory

The main purpose of coding theory field is to achieve the maximum rate $R$, at which information can be transmitted with arbitrarily small probability of error, at a given SNR ([16]). Factor graphs theory originated from low density parity check (LDPC) codes, which were invented by Gallagher [1]. This chapter provides a brief introduction in the coding theory field with a factor graph approach. First, we consider binary block linear codes, which simplify the analysis of LDPC codes, and then we study convolutional codes and Turbo codes. The goal of this chapter is to express the aforementioned codes in a factor graph representation, and apply decoding utilizing the sum-product algorithm (SPA).

We consider a stream of bits transmitted to a destination through an additive white Gaussian noisy AWGN channel. Our goal is to apply SPA on the graph corresponding to a code, in order to restore at the receiver the received (and possibly) corrupted bit-stream. An efficient way to achieve this, is by inserting redundant parity bits in the transmitted sequence before the latter pass through the channel; such procedure is called *encoding*. *Decoding* is the process in the receiver, during which the latter recovers the encoded originally transmitted sequence. During this chapter we will consider binary block codes, i.e. the transmitted coded symbols are binary symbols. Finally, the goals of this chapter are:

1. to understand how several classes of binary block codes are closely related to factor graphs,

2. given a binary block code, how to construct the equivalent factor graph taking into account the code's constraints,

3. how to apply the SPA, in order to perform decoding.

Through this chapter, we consider modulo 2 arithmetic since we are in binary field $\mathbb{B} \equiv \mathbb{F}_2 \equiv \{x \mid x \in \{0, 1\}\}$. A simple example of modulo 2 arithmetic is the addition and

the inner product of 2 arbitrary binary vectors $\mathbf{b}_1 = 010$ and $\mathbf{b}_2 = 110$. Their sum gives $\mathbf{b}_1 + \mathbf{b}_2 = 100$ and their inner product gives $\mathbf{b}_1\mathbf{b}_2^\top = 0 + 1 + 0 = 1$. Finally, every vector $\mathbf{x}$ which contains binary elements in this chapter is considered as a row vector. The interested reader on coding theory can see also [28], [29] and [31].

## 3.1 Linear block codes

We start the analysis from binary block linear codes because they constitute the simplest case of error-correcting codes.

Consider a binary information block sequence $\mathbf{b}$ with block length $k$. A block code is a linear mapping from every binary vector $\mathbf{b} \in \mathbb{B}^k$ to a binary vector $\mathbf{c} \in \mathbb{B}^n$, with $n > k$, and $n$ standing for the block length of the codeword. The set of all codewords $\mathbf{c}$ form a block code $\mathcal{C}$ and all possible codewords are at most $2^k$. The above process is associated with the *encoding*. Obviously, the set $\mathcal{C}$ must consist of distinct codewords $\mathbf{c}$, in order to identify uniquely each one. Every set $\mathcal{C}$ corresponds to a code $(n, k)$, if each codeword has length equal to $n$ and there are $2^k$ distinct codewords.

**Definition 3.1** (**Linear block code**)**.** A block code $(n, k)$ is said to be linear, if and only if, it forms a $k$-dimensional *subspace* of binary $n$-tuples ($\mathbb{B}^n$), namely $\mathcal{C} \subset \mathbb{B}^n$ with $dim\{\mathcal{C}\} = k$. Since it forms a subspace, it contains the all zero $n$-tuple, i.e. $\mathbf{0}_{1\times n} \in \mathcal{C}$. ▼

A subspace is a vector space which is closed over addition and multiplication with a scalar. A $k$-dimensional subspace $\mathcal{S}$ of binary $n$-tuples consist of binary vectors $\mathbf{x}_i \in \mathbb{B}^n$. Any *linear combination* of $\mathbf{x}_i$'s yield a vector space of dimension $k < n$, namely the subspace $\mathcal{S} \subset \mathbb{B}^n$. Consider any subspace $\mathcal{S}$ (of binary $n$-tuples), then the following statement is in force:

- if $\mathbf{x}_1 \in \mathcal{S}$ and $\mathbf{x}_2 \in \mathcal{S}$ then

$$a\mathbf{x}_1 + b\mathbf{x}_2 \in \mathcal{S}, \quad a, b \in \mathbb{B}. \tag{3.1}$$

The vector $\mathbf{y} = a\mathbf{x}_1 + b\mathbf{x}_2$ is a linear combination of vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. If the reader is not familiar with the definitions of linear algebra can see also [32].

**Definition 3.2** (**Rate**)**.** The following ratio:

$$r \triangleq \frac{k}{n}, \tag{3.2}$$

is called the rate of a $(n, k)$ code and denotes how many information bits correspond to the total transmitted bits per codeword. ▼

**Definition 3.3** (**Weight**)**.** The weight of a codeword $\mathbf{c}$ is the number of its non zero elements. ▼

**Definition 3.4** (**Minimum distance**)**.** The minimum distance of a code $\mathcal{C}$ is denoted with $d_{min}(\mathcal{C})$, and is defined as the minimum weight of any codeword (except the zero codeword). Specifically,

$$d_{min}(\mathcal{C}) = \min_{\mathbf{c}_i \in \mathcal{C} \setminus \mathbf{c}_i = \mathbf{0}} ||\mathbf{c}_i||_2 \,, \tag{3.3}$$

▼

**Definition 3.5** (**Orthogonality**)**.** Two non zero vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ are said to be orthogonal, if and only if $\mathbf{c}_i \mathbf{c}_j^\top = 0$. ▼

**Definition 3.6** (**Dual code**)**.** Consider a linear block code $\mathcal{C}$. The orthogonal or dual code $\mathcal{C}^\perp$ is defined as the code, which consist of $n$-tuples which are perpendicular to every codeword $\mathbf{c}$ of code $\mathcal{C}$, namely

$$\mathcal{C}^\perp = \{\mathbf{y} \in \mathbb{B}^n \mid \mathbf{y}\mathbf{c}^\top = 0, \ \forall \mathbf{c} \in \mathcal{C}\}. \tag{3.4}$$

▼

The following example explains the definitions above.

**Example 3.1.** Consider the binary code $\mathcal{C} = \{0000,\ 0110,\ 1001,\ 1111\}$. The code $\mathcal{C}$ forms a 2-dimensional subspace in $\mathbb{B}^4$ since it is closed over summation and multiplication with a binary number, namely:

$$\mathbf{c}_i \in \mathcal{C},\ \mathbf{c}_j \in \mathcal{C} \Longleftrightarrow a\mathbf{c}_i + b\mathbf{c}_j \in \mathcal{C}$$

with $a,\ b \in \mathbb{B}$. The rate of the code $\mathcal{C}$ is $r = k/n = 1/2$ and its minimum distance $d_{min}(\mathcal{C}) = 2$. Notice that any codeword of $\mathcal{C}$ is orthogonal to all other codewords in $\mathcal{C}$. Therefore $\mathcal{C} = \mathcal{C}^\perp$. The code $\mathcal{C}$ is called *self-dual*. ■

Since a linear code $\mathcal{C}$ forms a subspace of dimension $k$, there exist $k$ linearly independent binary vectors in $\mathbb{B}^n$, which yield a basis of dimension $k$. Specifically if we denote these $k$

linearly independent vectors as $\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_k$, then any linear combination of them generates a codeword in $\mathcal{C}$. Let us arrange these vectors in an $k \times n$ matrix:

$$\mathbf{G} \triangleq \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_k \end{bmatrix}, \tag{3.5}$$

this matrix is called *generator* matrix and all codewords in $\mathcal{C}$ can be generated from the rows of $\mathbf{G}$; consider an arbitrary information word $\mathbf{b}$, then any codeword $\mathbf{c}$, such that

$$\mathbf{c} = \mathbf{b}\mathbf{G}, \quad \mathbf{b} \in \mathbb{B}^k, \tag{3.6}$$

belongs in $\mathcal{C}$. Notice that generator matrix $\mathbf{G}$ is full row rank.

The utilization of generator matrix $\mathbf{G}$ is preferable, since it costs less storage complexity. To see this, consider the case when we store $2^k$ $n-$tuples for a code $\mathcal{C}$. Then, the encoder would demand storage $\mathcal{O}(2^k n)$. On the other hand, if we store every element of the generator matrix $\mathbf{G}$, then the encoder will require spatial complexity $\mathcal{O}(kn)$.

**Definition 3.7** (**Systematic code**)**.** Any linear block code which has generator matrix according to the following form:

$$\mathbf{G}_s = \begin{bmatrix} \mathbf{I}_k & \mathbf{P} \end{bmatrix}, \tag{3.7}$$

is called systematic linear block code. $\mathbf{I}_k$ denotes the $k \times k$ identity matrix, while the matrix $\mathbf{P}$ is a parity $k \times (n-k)$ binary matrix. A systematic block code has the property that the first $k$ bits of the coded word are equal with the $k$ transmitted information bits, and thus, information bits can be easily restored from the first $k$ bits of the codeword. To obtain a systematic generator matrix $\mathbf{G}_s$, we apply column permutations or row operations to the matrix $\mathbf{G}$. The matrices $\mathbf{G}$ and $\mathbf{G}_s$ are called equivalent, since their corresponding codes are equivalent. ▼

Every linear block code has an equivalent systematic code representation, since if we choose $k$ linear independent columns of generator matrix $\mathbf{G}$ we can yield the space $\mathbb{B}^k$. Nevertheless, the space $\mathbb{B}^k$ can be yield also, by the columns of $k \times k$ identity matrix. Therefore, if we transfer the $k$ linear independent columns of generator matrix $\mathbf{G}$ at the first $k$ columns of $\mathbf{G}$, after row operations we can create the $k \times k$ identity matrix.

**Definition 3.8** (**Parity check matrix**)**.** The parity check matrix is denoted by $\mathbf{H}$ and is

defined as the $(n-k)\times n$ matrix, whose all rows are orthogonal to the row space of matrix $\mathbf{G}$. That is, all rows of parity matrix $\mathbf{H}$ belong to the nullspace of generator matrix $\mathbf{G}$.

Any code $\mathcal{C}$ can be defined in terms of its parity check matrix $\mathbf{H}$; it can be shown that any codeword $\mathbf{c}$ belongs to $\mathcal{C}$, if and only if, it is perpendicular in any row of parity check matrix:

$$\mathbf{c}\in\mathcal{C}\iff\mathbf{c}\mathbf{H}^{\top}=\mathbf{0}. \tag{3.8}$$

▼

**Corollary 3.9.** *Consider any $(n,k)$ linear block code $\mathcal{C}$ with generator matrix $\mathbf{G}$, then the $(n,n-k)$ code, $\mathcal{C}'$, which has as generator matrix the $(n-k)\times n$ parity check matrix $\mathbf{H}$ of code $\mathcal{C}$, is the dual code of $\mathcal{C}$, i.e. $\mathcal{C}'=\mathcal{C}^{\perp}$.*

*Proof.* The proof arises from the fact that any row of parity check matrix $\mathbf{H}$ is orthogonal to any codeword in $\mathcal{C}$, therefore the codewords generated from matrix $\mathbf{H}$ belong to the dual code of $\mathcal{C}$. Consequently, we conclude $\mathcal{C}'=\mathcal{C}^{\perp}$. □

Any systematic generator matrix $\mathbf{G}_s$ has a corresponding systematic parity check matrix $\mathbf{H}_s$, that is,

$$\mathbf{H}_s=\left[\begin{array}{cc}\mathbf{P}^{\top} & \mathbf{I}_{n-k}\end{array}\right]. \tag{3.9}$$

Obviously $\mathbf{G}\mathbf{H}^{\top}=\mathbf{G}_s\mathbf{H}^{\top}=\mathbf{G}\mathbf{H}_s^{\top}=\mathbf{G}_s\mathbf{H}_s^{\top}=\mathbf{0}_{k\times(n-k)}$.

From now on we will consider $\mathbf{G}\equiv\mathbf{G}_s$ and $\mathbf{H}\equiv\mathbf{H}_s$, i.e. any generator matrix and its corresponding systematic one will have the same symbolism, similarly, for parity check matrix $\mathbf{H}$. We do this for simplicity in notation. The following example clarifies the above definitions.

**Example 3.2.** Consider a $(8,4)$ code $\mathcal{C}$, with the following generator matrix :

$$\mathbf{G}=\left[\begin{array}{cccccccc}0&0&0&0&1&1&1&1\\1&0&1&0&1&0&1&0\\0&0&1&1&0&0&1&1\\1&1&1&1&1&1&1&1\end{array}\right].$$

The rate $r$ of the code $\mathcal{C}$ is equal to $1/2$, while the minimum distance is $d_{min}(\mathcal{C})=4$ since the minimum non zero elements of any codeword of $\mathcal{C}$ is 4. Then if we permute the 5th

column with the 1st one, we take:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

if we add the 2nd row at the 4th row and then the 1st row at the 2nd one, we take

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Then we add the 3rd and 4th row and the result is added at the 2nd row, furthermore, we add the resulting 2nd row to the 4th row and we take:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Finally we add the 4th row at the 3rd row and we take the systematic form of the generator matrix, namely,

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_4 & \mathbf{P} \end{bmatrix}.$$

Since the generator matrix $\mathbf{G}$ has systematic form, we can extract easily the parity check matrix $\mathbf{H}$ as

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^\top & \mathbf{I}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Consider an arbitrary information bit sequence, e.g. $\mathbf{b} = [0 \;\; 1 \;\; 0 \;\; 1]$, then the encoded codeword according to definitions above is

$$\mathbf{c} = \mathbf{bG} = [0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1] = [\mathbf{c}_s \;\; \mathbf{c}_p],$$

where the first sub-codeword ($\mathbf{c}_s$) corresponds to the systematic part of the codeword, while the last sub-codeword ($\mathbf{c}_p$) corresponds to the parity part of the codeword. Obviously, $\mathbf{c}_s = \mathbf{b}$. Finally, if we want to check if $\mathbf{c}$ belongs to $\mathcal{C}$, it suffices to check if it is perpendicular to every row of parity check matrix $\mathbf{H}$, i.e. whether

$$\mathbf{cH}^\top = [0 \;\; 0 \;\; 0 \;\; 0].$$

Notice that every row of the parity check matrix corresponds to an equation that must be satisfied, e.g. the first row of matrix $\mathbf{H}$ imposes the modulo-2 sum of coded-bits $c_2$, $c_3$, $c_4$ and $c_5$ to be equal to zero and so forth. Every equation can be seen as indicator function of the sum of the variables (bits) involved in the corresponding function and they are also called *checks*. ■

Relying on the above example, we observe that the first process (multiplication of information word with generator matrix) corresponds to the encoding phase, while the second process (multiplication of the received coded word with the parity check matrix) verifies that the received codeword belongs in code $\mathcal{C}$. After a brief review on linear block codes, we examine the first class of capacity-approaching codes, the LDPC codes, under the scope of factor graphs.

## 3.2 Low density parity check (LDPC) codes

Low density parity check (LDPC) codes are linear block codes which are defined via their parity check matrix $\mathbf{H}$. As their name indicates, the basic property of their parity check matrix $\mathbf{H}$ is that it has sparse structure, i.e. it has low density of non-zero (ones) elements. As we explicitly stated above, all codewords $\mathbf{c}$ of a linear block code $\mathcal{C}$ with parity check matrix $\mathbf{H}$ must satisfy $\mathbf{cH}^\top = \mathbf{0}$. The latter equation corresponds to a linear system of equations which must be satisfied. The basic idea of LDPC codes is to express these equations in terms of factor graphs. We define as $m$ the number of equations (checks), namely $m \triangleq n - k$.

**Definition 3.10** (**Regular-Irregular code**)**.** Consider a LDPC code with a $(n-k) \times n$ parity check matrix $\mathbf{H}$. The code is said to be regular if and only if a constant number of non zero coded bits are involved in each equation and a constant number of equations contain each symbol. Namely if the variables $w_r$, $w_c$, indicate the number of non-zero elements in each row and in each column of the parity check matrix, respectively, then if

$$mw_r = nw_c, \tag{3.10}$$

the code is regular. In any other case, the code is irregular. ▼

For example the code with the following parity check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix},$$

is a regular code since it contains constant number of column and row weights ($w_c = 2$ and $w_r = 4$).

### 3.2.1 Graph representation of LDPC codes

In this subsection, we will examine how we can construct a graph representation of a LDPC code. For every LDPC code there is an equivalent *Tanner* graph. Since Tanner graphs come under the family of factor graphs, from now on we will refer to factor graphs. Given a $m \times n$ parity check matrix $\mathbf{H}$, we create the equivalent factor graph as follows:

- For every variable corresponding to bit, we create a variable node, i.e. in terms of matrix $\mathbf{H}$, we create $n$ variable nodes (equal the number of columns of the parity check matrix).
- For every equation (check), we create a factor (check) node, namely we create $m$ check nodes.
- If the $j, i$th element of parity check matrix is equal to 1 ($\mathbf{H}_{j,i} = 1$), then we create an edge between variable node $i$ and check node $j$.

Following the standard notation, the symbol for check nodes will be the non-solid square with the symbol '+' inside it. More specifically, in figure 3.1 below, are presented the symbols representing check nodes and variable nodes.

Figure 3.1: Left: check node - Right: variable node.

The factor corresponding to every check node is equal to the indicator function of the constraints imposed from the corresponding row of parity check matrix $\mathbf{H}$. Namely check node $j$ is denoted with $f_j(\cdot)$ and is defined as

$$f_j(\cdot) = \mathbb{I}\left\{\left(\sum_{i|\mathbf{H}_{j,i}=1} c_i\right) = 0\right\}, \tag{3.11}$$

where $\mathbb{I}\{\cdot\}$ denotes the indicator function of the expression inside the hooks. The indicator function is defined as

$$\mathbb{I}\{x\} = \begin{cases} 1, & \text{if } x \text{ is true,} \\ 0, & \text{otherwise,} \end{cases} \tag{3.12}$$

The arguments of every factor $f_j(\cdot)$ are the variables whose nodes are adjacent with check node $f_j$. If we denote the set of arguments of check node $f_j$, $\mathbf{x}_j$, then

$$\mathbf{x}_j = \{c_i \mid \mathbf{H}_{j,i} = 1\}, \tag{3.13}$$

with $\mathbf{c} = [c_1 \quad \cdots \quad c_n]$.

Therefore a code word belong to a LDPC code $\mathcal{C}$ if all the $m$ constraints from parity check matrix $\mathbf{H}$ are satisfied, therefore we have

$$\mathbf{c} \in \mathcal{C} \Longleftrightarrow \prod_{i=1}^{m} f_j(\mathbf{x}_j) = 1. \tag{3.14}$$

The last equation shows that the FG of LDPC codes expresses a factorisable function (rigth-hand side of Eq. 3.14).

**Example 3.3.** Consider the $(8, 4)$ code with the parity check matrix corresponding to the

code of example 3.2, recall

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^\top & \mathbf{I}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that the above matrix is not very sparse, since it contains 32 elements, where the 16 of them are non-zero elements. The factor graph corresponding to that code is illustrated in figure 3.3 below, since

$$\mathbf{c} \in \mathcal{C} \Longleftrightarrow \prod_{i=1}^{4} f_j(\mathbf{x}_j) = 1.$$

As we can see, the variable nodes correspond to the code-bits, here $c_1$ to $c_8$, while the check nodes correspond to indicator functions according to equation 3.11:

$$\begin{aligned} f_1(\mathbf{x}_1) &= \mathbb{I}(c_2 + c_3 + c_4 + c_5 = 0), \\ f_2(\mathbf{x}_2) &= \mathbb{I}(c_1 + c_3 + c_4 + c_6 = 0), \\ f_3(\mathbf{x}_3) &= \mathbb{I}(c_1 + c_2 + c_4 + c_7 = 0), \\ f_4(\mathbf{x}_4) &= \mathbb{I}(c_1 + c_2 + c_3 + c_8 = 0), \end{aligned}$$

with $\mathbf{x}_1 = [c_2 \;\; c_3 \;\; c_4 \;\; c_5]$, all the other $\mathbf{x}_i,\ i = 2, 3, 4$, computed similarly. ■

### 3.2.2 Encoding of LDPC codes

As we stated in the above subsection, LDPC codes are usually defined across their parity check matrix, instead of their generator matrix. Given a parity check matrix $\mathbf{H}$ of a LDPC code, the simplest way to construct a generator matrix corresponding to the parity check matrix is to construct an equivalent systematic parity check matrix which has the following form

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^\top & \mathbf{I}_m \end{bmatrix}, \tag{3.15}$$

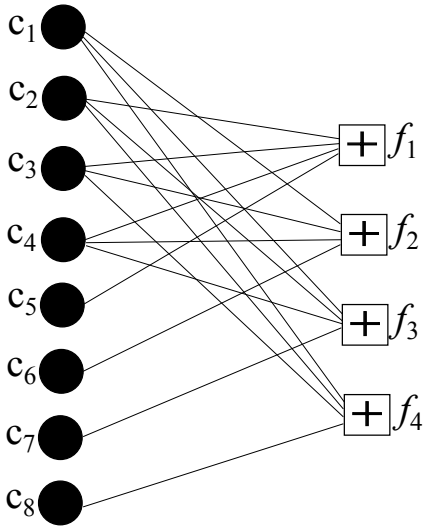



Figure 3.2: The factor graph corresponding to the $(8, 4)$ code with the parity check matrix of example 3.3.

where $\mathbf{P}$ is the parity matrix. Finally, we can get easily the equivalent systematic generator matrix, as

$$\mathbf{G} = \left[ \begin{array}{cc} \mathbf{I}_k & \mathbf{P} \end{array} \right]. \tag{3.16}$$

Since we have created the generator matrix, we are able to encode the transmitted bit sequence $\mathbf{b}$. The encoded sequence $\mathbf{c}$ is given by

$$\mathbf{c} = \mathbf{bG}, \quad \mathbf{b} \in \mathbb{B}^k. \tag{3.17}$$

### 3.2.3 Decoding of LDPC codes

The decoding algorithm of LDPC codes generated the idea of factor graphs (FGs) and the sum-product algorithm (SPA) [1]. Since the factor graph of LDPC codes contains cycles, the decoding algorithm must be performed under the presence of cycles, therefore it is applied on iterative fashion, while its convergence is not always sure. It has been noted that the performance of the decoding algorithms is improved when the FG girth is growing [31]. If we want to enlarge the girth of a FG, we must increase the size of block length $n$, and thus, the complexity of the decoding algorithm increases too. The good error performance of LDPC codes relies on the soft decision rules of SPA.

The following lemma simplifies the update rules for SPA decoding algorithm.

**Lemma 3.11** ([1]). *For a sequence of $M$ independent binary digits $x_i$ with probability $p_i$ for $x_i = 1$ (probability for $x_i = 0$ equal to $1 - p_i$), the probability that the whole sequence contains an even number of 1's (hence even parity) is*

$$P(\{x_1, ..., x_M\} \text{ has even parity}) = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{M}(1 - 2p_i). \tag{3.18}$$

*Proof.* Consider the function

$$f(t) = \prod_{i=1}^{M}(1 - p_i + p_i t).$$

This function is a polynomial in $t$, and note that the coefficients of $t^n$ is the probability of $n$ 1's. Similarly, the function

$$g(t) = \prod_{i=1}^{M}(1 - p_i - p_i t).$$

is identical except that all the odd powers of $t$ are negative. If we add these 2 functions all the even terms will be doubled, whereas the odd terms will cancel out. Finally, if we let $t = 1$ and if we divide by 2, we take the probability of even parity. Therefore

$$\begin{aligned} P(\{x_1, ..., x_M\} \text{ has even parity}) &= \frac{f(t) + g(t)}{2}\Big|_{t=1} \\ &= \frac{\prod_{i=1}^{M}(1 - p_i + p_i t) + \prod_{i=1}^{M}(1 - p_i - p_i t)}{2}\Big|_{t=1} \\ &= \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{M}(1 - 2p_i). \end{aligned}$$

This completes the proof of lemma □

The above lemma associated with the update rule for check nodes of SPA. To clarify the aforementioned claim consider a check node $f_j$ with $M$ neighboring variable nodes $c_1, ..., c_i, ..., c_M$, corresponding to independent coded bits with probabilities of each one taking the value 1, $p_1, ..., p_i, ..., p_M$, respectively. Then according to SPA, the update rule

for the outgoing message from factor node $f_j$ to a neighboring variable node $c_i$ is given by

$$\mu_{f_j \longrightarrow c_i}(c_i) = \sum_{\sim\{c_i\}} f_j(c_1, ..., c_i, ..., c_M) \prod_{c_l \in \mathcal{N}(f_j)\backslash\{c_i\}} \mu_{c_l \longrightarrow f_j}(c_l). \tag{3.19}$$

As we mentioned above the factor $f_j(c_1, ..., c_i, ..., c_M) = \mathbb{I}\{c_1 + ... + c_i + ... + c_M = 0\}$. The right-hand side of the latter equation can be written as

$$\mathbb{I}\{c_1 + ... + c_i + ... + c_M = 0\} = \mathbb{I}\{c_1 + ... + c_{i-1} + c_{i+1} + ... + c_M = -c_i\}, \tag{3.20}$$

but the coded bits are binary variables, therefore $-c_i = c_i$ and hence

$$\mathbb{I}\{c_1 + ... + c_i + ... + c_M = 0\} = \mathbb{I}\{c_1 + ... + c_{i-1} + c_{i+1} + ... + c_M = c_i\}. \tag{3.21}$$

Note that the indicator function of Eq. 3.21 becomes 1 if and only if $c_1 + ... + c_{i-1} + c_{i+1} + ... + c_M = c_i$. If we fix the variable $c_i$ to be equal to zero (0), then the indicator factor of Eq. 3.21 takes the value 1 if and only if the coded bits $c_1, ..., c_{i-1}, c_{i+1}, ..., c_M$, have even parity.

If we further consider that messages $\mu_{c_l \longrightarrow f_j}(1) = p_l,\ l = 1, ..., M\backslash\{i\}$, then the factor $\mu_{f_j \longrightarrow c_i}(0)$ is the sum over all possible configurations of factor $f_j$ and the joint pmf of $M-1$ independent coded bits (the joint pmf is the product of individual pmfs since the coded bits are independent). However,

$$\mathbb{I}\{c_1 + ... + c_{i-1} + c_{i+1} + ... + c_M = 0\} = \begin{cases} 1, & \text{if the coded bits } c_1, ..., c_{i-1}, c_{i+1}, ...c_M \\ & \text{have even parity,} \\ 0, & \text{otherwise,} \end{cases} \tag{3.22}$$

therefore the message $\mu_{f_j \longrightarrow c_i}(0)$ is the sum of the joint pmf of all possible even parity configurations of coded bits $c_1, ..., c_{i-1}, c_{i+1}, ..., c_M$, i.e. the probability of coded bits $c_1, ..., c_{i-1}, c_{i+1}, ..., c_M$, have even parity. Therefore we conclude that

$$\begin{aligned} \mu_{f_j \longrightarrow c_i}(0) &= \frac{1}{2} + \frac{1}{2} \prod_{c_l \in \mathcal{N}(f_j)\backslash\{c_i\}} (1 - 2p_l) \\ &= \frac{1}{2} + \frac{1}{2} \prod_{c_l \in \mathcal{N}(f_j)\backslash\{c_i\}} (1 - 2\mu_{c_l \longrightarrow f_j}(1)), \end{aligned} \tag{3.23}$$

and

$$\mu_{f_j \longrightarrow c_i}(1) = \frac{1}{2} - \frac{1}{2} \prod_{c_l \in \mathcal{N}(f_j) \setminus \{c_i\}} (1 - 2\mu_{c_l \longrightarrow f_j}(1)) \tag{3.24}$$

$$= 1 - \mu_{f_j \longrightarrow c_i}(0). \tag{3.25}$$

The update rule for variable nodes are the same with the classic SPA, i.e. any outgoing message from a variable node $c_i$ to a neighboring factor node $f_j$ is given by

$$\mu_{c_i \longrightarrow f_j}(c_i) = \frac{1}{N} \prod_{f_l \in \mathcal{N}(c_i) \setminus \{f_j\}} \mu_{f_l \longrightarrow c_i}(c_i), \tag{3.26}$$

where $N$ is a normalization constant.

Before we derive the SPA decoding algorithm we first create a more convenient FG accounting the effects of the communication channel. We assume that the coded bit sequence $\mathbf{c}$ is mapped according to a binary phase shift keying (BPSK) modulation mapping, $f_{\text{BPSK}}$:

$$f_{\text{BPSK}}(\mathbf{c}) = -2\mathbf{c} + \mathbf{1}_{1\times n} = \mathbf{s}, \quad \mathbf{s} \in \{+1, -1\}^n, \tag{3.27}$$

and $\mathbf{s} = [s_1 \quad \cdots \quad s_n]$. If we further assume memoryless channel, then the likelihood of the observation vector $\mathbf{y}$ given the transmitted signal $\mathbf{s}$ will take the following form:

$$f_{\mathbf{Y}|\mathbf{S}}(\mathbf{y} \mid \mathbf{s}) = \prod_{i=1}^{n} f_{Y_i|S_i}(y_i \mid s_i). \tag{3.28}$$

Assuming that all codewords are equiprobable, then the a posteriori probability of signal vector $\mathbf{s}$ given the observation vector $\mathbf{y}$ according to Bayes rule is given by

$$p(\mathbf{s} \mid \mathbf{y}) = \frac{1}{|\mathcal{C}|} \prod_{i=1}^{n} \frac{f_{Y_i|S_i}(y_i \mid s_i)}{f_{Y_i}(y_i)}, \tag{3.29}$$

where $|\mathcal{C}|$ denotes the cardinality of code $\mathcal{C}$.

Without lose of correctness, we utilize the symbols $s_i,\ i = 1, ..., n$, instead of the coded bits $c_i,\ i = 1, ..., n$, as variable nodes of FG. Without changing the structure of FG we insert also the factor $p(s_i \mid y_i)$ of the posterior of symbol $s_i$ transmitted given the observation $y_i$, in order to take into account the channel effects. Consequently, the arguments of check nodes $f_j,\ \forall j = 1, ..., m$, as well as of factors $p(s_i \mid y_i),\ \forall i = 1, ..., n$, are the symbols

$s_i$ and not the coded bits $c_i$. Since the mapping from every $c_i$ to the corresponding $s_i$, $\forall i = 1, ..., n$, is one-to-one, the update rule for factor nodes result according to Eqs. 3.23 and 3.25, except the fact that variables $s_i,\ \forall i$, take the values $+1$ or $-1$. Hence $(\forall j)$

$$\mu_{f_j \longrightarrow s_i}(+1) = \frac{1}{2} + \frac{1}{2} \prod_{s_l \in \mathcal{N}(f_j) \backslash \{s_i\}} (1 - 2\mu_{s_l \longrightarrow f_j}(-1)), \tag{3.30}$$

and

$$\mu_{f_j \longrightarrow s_i}(-1) = 1 - \mu_{f_j \longrightarrow s_i}(+1). \tag{3.31}$$

We utilize the abbreviation $p_i(s_i)$ standing for the function $p(s_i \mid y_i)$. The update rule for variable nodes are the following $(\forall i)$:

$$\mu_{c_i \longrightarrow f_j}(c_i) = \frac{1}{N}\ p_i(s_i) \prod_{f_m \in \mathcal{N}(s_i) \backslash \{f_j\}} \mu_{f_m \longrightarrow s_i}(s_i), \tag{3.32}$$

The augmented FG of the LDPC code of the example 3.2 is illustrated in figure 3.3.

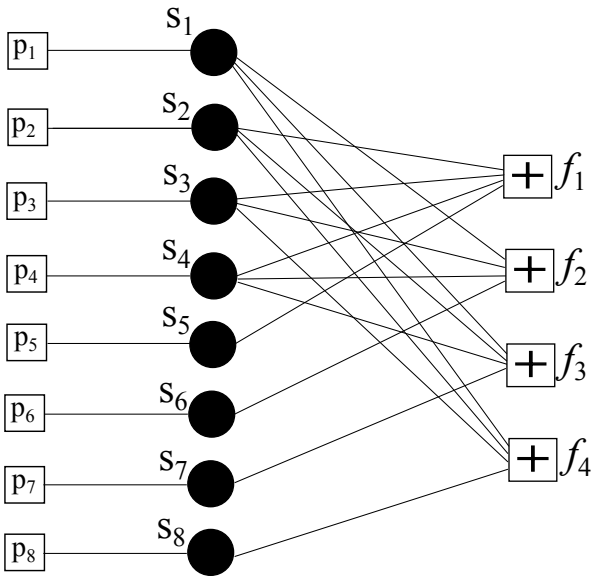



Figure 3.3: The augmented factor graph of the LDPC of the example 3.2. This graph takes into account the effect of the channel. Notice that the variable nodes $s_i,\ i = 1, ..., n$, take the values $\{+1, -1\}$.

The SPA is applied in order to perform decoding. Briefly, the algorithm follows the

below scheduling:

- Each variable node sends a message (vector) to every neighboring check node.
- Each check node sends a message to every neighboring variable node.
- Each variable node calculates the a posteriori probability that symbol $s_i$ takes the value $+1$ or $-1$, based on the incoming messages from check nodes and on the observation $y_i$.

The following are utilized:

- Given a code $\mathcal{C}$ and its corresponding parity check matrix $\mathbf{H}$, there exists an edge between check node $f_j$ and variable node $s_i$ if and only if $\mathbf{H}_{j,i} = 1$;
- for every variable node $s_i$, the message to a neighboring check node $f_j$ is denoted by $\mu_{s_i \to f_j}(s_i)$ and is a function of variable $s_i$;
- for every check node $f_j$ the message to a neighboring variable node $s_i$ is denoted by $\mu_{f_j \to s_i}(s_i)$ and is a function of the variable $s_i$;
- the marginal corresponding to a variable node $s_i$, $g_{s_i}(s_i)$, is equal to

$$g_{s_i}(s_i) = \mu_{s_i \to f_j}(s_i)\mu_{f_j \to s_i}(s_i), \tag{3.33}$$

  this marginal corresponds to the a posteriori probability distribution of symbol $s_i$ transmitted given the channel effects, given the code's constraints;

- let $R_j$ denote the set of indexes corresponding to the positions of 1's in row $j$ of parity check matrix $\mathbf{H}$. Similarly let $C_i$ denote the set of indexes corresponding to the positions of 1's in column $i$ of matrix $\mathbf{H}$. Finally the symbol $'\backslash'$ denotes the expression except, namely $R_{j\backslash i}$ denotes all the positions of 1's of the $j$'th column except the position $i$.

Finally the following 6 steps constitute the decoding algorithm for LDPC codes, based on iterative SPA.

1. 
   - Inputs: messages $p_i(s_i)$ corresponding to the observations $y_i$, $i = 1, ..., n$ (assuming all codewords are equiprobable).
   - Set a counter $count = 1$.

2. - Set $q_i := p_i(s_i = +1) = p(s_i = +1 \mid y_i)$, i.e. initialize the a posteriori probabilities of symbols based on the channel observations;
   - set $\mu_{s_i \to f_j}(+1) := q_i$;
   - set $\mu_{s_i \to f_j}(-1) := 1 - q_i$.

   $\forall i = 1, ..., n, \quad \forall j = 1, ..., m$. The messages of this step are depicted in figure 3.4.

3. Information from check nodes to variable nodes, i.e. $(\forall j, i \ : \ \mathbf{H}_{j,i} = 1)$:

   - $\mu_{f_j \to s_i}(+1) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j \setminus i}} (1 - 2\mu_{s_{i'} \to f_j}(-1))$;
   - $\mu_{f_j \to s_i}(-1) = 1 - \mu_{f_j \to s_i}(+1)$.

   The messages according to this step are illustrated in 3.5.

4. Information from variable nodes to check nodes, i.e. $(\forall j, i \ : \ \mathbf{H}_{j,i} = 1)$:

   - $\mu_{s_i \to f_j}(+1) = \frac{1}{N} \ q_i \ \prod_{j' \in C_{i \setminus j}} \mu_{f_{j'} \to s_i}(+1)$;
   - $\mu_{s_i \to f_j}(-1) = \frac{1}{N} \ (1 - q_i) \ \prod_{j' \in C_{i \setminus j}} \mu_{f_{j'} \to s_i}(-1)$,

   where $N$ denotes a normalization constant, namely $N = \mu_{s_i \to f_j}(-1) + \mu_{s_i \to f_j}(+1)$. The messages of this step are illustrated in figure 3.6.

5. Calculate the marginals with respect to variables $s_i$ corresponding to a posteriori probabilities, i.e $(\forall i = 1, ..., n)$:

   - $g_{s_i}(+1) = \mu_{s_i \to f_j}(+1) \times \mu_{f_j \to s_i}(+1)$,
   - $g_{s_i}(-1) = \mu_{s_i \to f_j}(-1) \times \mu_{f_j \to s_i}(-1)$,

   for any $j$, such that $\mathbf{H}_{j,i} = 1$. The messages corresponding to step 4 are shown in figure 3.7.

6. Apply the hard decisions, in order to restore the transmitted codeword, that is, for every $i = 1, ..., n$:

$$\hat{c}_i = \begin{cases} 0, & \text{if } \ g_{s_i}(+1) > g_{s_i}(-1), \\ 1, & \text{otherwise}, \end{cases} \tag{3.34}$$

   the whole estimated codeword is $\hat{\mathbf{c}} = [\hat{c}_1 \quad \cdots \quad \hat{c}_n]$.

7. If $\hat{\mathbf{c}}\mathbf{H}^{\top} = \mathbf{0}$ OR $count = N_{iter}$, then terminate, else $count = count + 1$ and goto step 2.

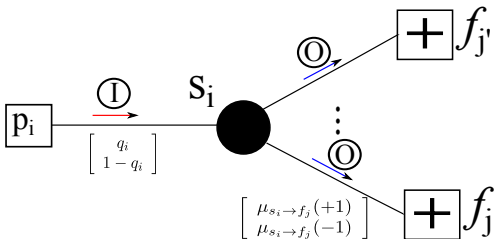



Figure 3.4: The messages of this step correspond to posterior probability of symbol $s_i$ transmitted given the channel observation $y_i$, $i = 1, ..., N$, and they are sent from factor node $p_i$ to variable node $s_i$. In sequel, they are propagated to every check node $f_j$ which neighboring to variable node $s_i$. As we can see the messages can be represented as vectors for two values, $+1$ or $-1$.

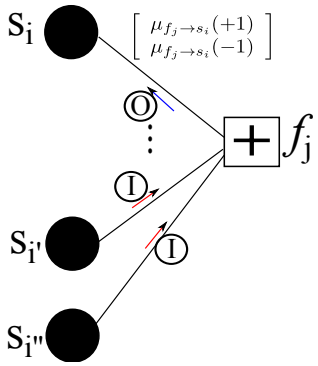



Figure 3.5: Messages from check nodes to variable nodes according to step 2.

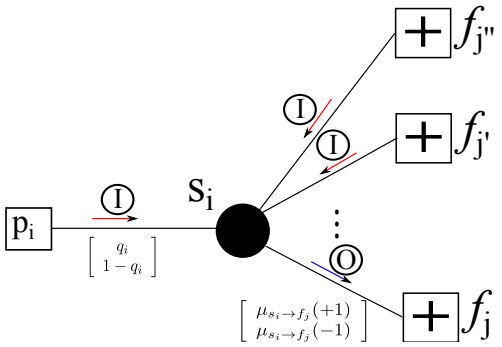



Figure 3.6: Messages from variable nodes to check nodes according to step 3.

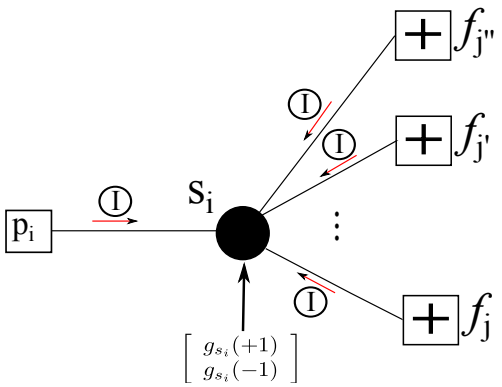



Figure 3.7: Marginal with respect to variable $s_i$.

In the figures 3.4-3.7, internal messages of the nodes are depicted with I-red arrows, while the O-blue arrows correspond to the outgoing messages.

The decoding algorithm can also be performed via likelihood ratios of the messages from variable nodes to check nodes and vice versa. We won't detail that method, since it follows the same idea with the SPA decoding algorithm, above. The interested reader can find details about this variant of SPA decoding algorithm in [31] and in [27].

The following example concludes the encoding-decoding procedure.

**Example 3.4.** Consider the $(8, 4)$ code $\mathcal{C}$ of the example 3.2. We assume that it is given the systematic parity check matrix, namely,

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^\top & \mathbf{I}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

As we stated above, the corresponding factor graph of the above parity check matrix $\mathbf{H}$ takes into account the channel observations and is given in figure 3.2.

Firstly, we need the generator matrix $\mathbf{G}$, in order to start the encoding phase. The generator matrix can be easily extracted from the systematic parity check matrix, namely,

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_4 & \mathbf{P} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Consider that we want to transmit the information bit sequence $\mathbf{b} = [1 \;\; 1 \;\; 0 \;\; 0]$, then according to encoding phase, the codeword $\mathbf{c}$, to be transmitted, is given by

$$\mathbf{c} = \mathbf{bG} = [1 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; 0 \;\; 0].$$

Moreover we assume that all codewords of $\mathcal{C}$ are equiprobable. We apply BPSK modulation on the coded sequence $\mathbf{c}$, and the transmitted signal becomes

$$\mathbf{s} = [-1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; -1 \;\; +1 \;\; +1].$$

The symbol sequence is transmitted over a channel with additive white Gaussian noise (AWGN) with variance $\sigma^2 = 0.31$. The corrupted received signal at the receiver is

$$\mathbf{y} = [-0.1316 \;\; -0.9517 \;\; 0.1612 \;\; 0.5826 \;\; -1.5970 \;\; 0.3218 \;\; 0.6538 \;\; 1.4207],$$

if we perform directly hard decisions in the received sequence $\mathbf{y}$ the estimated symbol sequence $\hat{\mathbf{s}}$ will be

$$\hat{\mathbf{s}} = [-1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; +1 \;\; +1].$$

Notice that the estimated sequence has an error at its 6th position. At this step, we initialize the SPA decoding algorithm.

Assuming that the a-priori probability of $s_i = +1$ is $p(s_i = +1) = p(s_i = -1) = 1/2$, we take

$$q_i = p_i(s_i = +1) = p(s_i = +1 \mid y_i) = \frac{f_{Y_i|S_i}(y_i \mid s_i = 1)p(s_i = +1)}{f_{Y_i}(y_i)},$$

where $f_{Y_i|S_i}(y_i \mid s_i = 1) \sim \mathcal{N}(1, \sigma^2)$ and $f_{Y_i}(y_i) = \sum_{s_i} f_{Y_i|S_i}(y_i|s_i)p(s_i)$. If we substitute these expressions, we take

$$\begin{aligned} p_i(s_i = +1) &= \frac{\frac{1}{\sqrt{2\pi\sigma^2}}\big(\exp\{-(y_i - 1)^2/2\sigma^2\}\big) \times \frac{1}{2}}{\frac{1}{\sqrt{2\pi\sigma^2}}\big(\exp\{-(y_i - 1)^2/2\sigma^2\} + \exp\{-(y_i + 1)^2/2\sigma^2\}\big) \times \frac{1}{2}} \\ &= \frac{1}{1 + \exp\{-[(y_i + 1)^2 - (y_i - 1)^2]/2\sigma^2\}} \\ &= \frac{1}{1 + \exp\{-2y_i/\sigma^2\}} \; . \end{aligned}$$

With similar way is can be shown that

$$p(s_i = -1 \mid y_i) = \frac{1}{1 + \exp\{+2y_i/\sigma^2\}} = 1 - q_i \ .$$

Generally, it can be shown that for any symbol $s_i$ with probability of taking values $+1$ or $-1$ equal to $1/2$, and furthermore normal likelihood distribution $f_{Y_i|S_i}(y_i|s_i) \sim \mathcal{N}(s_i, \sigma^2)$, the resulting a posteriori probability of symbol $s_i$ having a specific value ($+1$ or $-1$) conditioned on the observation $y_i$, is given ($\forall\ i = 1, ..., n$) by

$$p(s_i \mid y_i) = \frac{1}{1 + \exp\{-2s_i y_i/\sigma^2\}} \ , \qquad s_i = \{+1, -1\} \ . \tag{3.35}$$

Having calculated the a posteriori probability $p(s_i \mid y_i)$, we can initialize the SPA decoding algorithm. Therefore according to step 2, we compute all the external messages from variable nodes to check nodes, i.e. $\forall j, i \ : \ \mathbf{H}_{j,i} = 1$, we set

$$\mu_{s_i \to f_j}(+1) = q_i \ ,$$

therefore for degree 3 variable nodes we take

$$\begin{aligned}
&\mu_{s_1 \to f_2}(+1) = 0.3031, \quad \mu_{s_1 \to f_3}(+1) = 0.3031, \quad \mu_{s_1 \to f_4}(+1) = 0.3031, \\
&\mu_{s_2 \to f_1}(+1) = 0.0024, \quad \mu_{s_2 \to f_3}(+1) = 0.0024, \quad \mu_{s_2 \to f_4}(+1) = 0.0024, \\
&\mu_{s_3 \to f_1}(+1) = 0.7349, \quad \mu_{s_3 \to f_2}(+1) = 0.7349, \quad \mu_{s_3 \to f_4}(+1) = 0.7349, \\
&\mu_{s_4 \to f_1}(+1) = 0.9755, \quad \mu_{s_4 \to f_2}(+1) = 0.9755, \quad \mu_{s_4 \to f_3}(+1) = 0.9755,
\end{aligned}$$

while for degree 1 variable nodes we have

$$\begin{aligned}
&\mu_{s_5 \to f_1}(+1) = 0.0000, \\
&\mu_{s_6 \to f_2}(+1) = 0.8844, \\
&\mu_{s_7 \to f_3}(+1) = 0.9843, \\
&\mu_{s_8 \to f_4}(+1) = 0.9999.
\end{aligned}$$

Following the rule of step 2, we compute the message $\mu_{s_i \to f_j}(-1) = 1 - q_i, \ \forall j, i \ : \ \mathbf{H}_{j,i} = 1$. Now we are able to calculate the external messages from check nodes back to variable

nodes, e.g.

$$\begin{aligned}\mu_{f_2\to s_1}(+1) &= \frac{1}{2}+\frac{1}{2}\prod_{i'\in R_{2\setminus 1}}\big(1-2\mu_{s_{i'}\to f_2}(-1)\big)\\ &= \frac{1}{2}+\frac{1}{2}\big(1-2\mu_{s_3\to f_2}(-1)\big)\big(1-2\mu_{s_4\to f_2}(-1)\big)\big(1-2\mu_{s_6\to f_2}(-1)\big)\\ &= 0.6718.\end{aligned}$$

The rest of the messages for $s_i=+1,\ \forall i$, are computed accordingly, i.e.

$$\begin{aligned}&\mu_{f_1\to s_2}(+1)=0.2766,\quad \mu_{f_1\to s_3}(+1)=0.9732,\quad \mu_{f_1\to s_4}(+1)=0.7337,\quad \mu_{f_1\to s_5}(+1)=0.2777,\\ &\mu_{f_2\to s_1}(+1)=0.6718,\quad \mu_{f_2\to s_3}(+1)=0.3560,\quad \mu_{f_2\to s_4}(+1)=0.4289,\quad \mu_{f_2\to s_6}(+1)=0.4120,\\ &\mu_{f_3\to s_1}(+1)=0.0417,\quad \mu_{f_3\to s_2}(+1)=0.3187,\quad \mu_{f_3\to s_4}(+1)=0.6898,\quad \mu_{f_3\to s_7}(+1)=0.6863,\\ &\mu_{f_4\to s_1}(+1)=0.2663,\quad \mu_{f_4\to s_2}(+1)=0.4075,\quad \mu_{f_4\to s_3}(+1)=0.6959,\quad \mu_{f_4\to s_8}(+1)=0.5920,\end{aligned}$$

This step finishes with the calculation of $\mu_{f_j\to s_i}(-1),\ \forall j,i\ :\ \mathbf{H}_{j,i}=1$, according to the 3rd step, i.e. $\mu_{f_j\to s_i}(-1)=1-\mu_{f_j\to s_i}(+1),\ \forall j,i\ :\ \mathbf{H}_{j,i}=1$. The external messages from variable nodes to check nodes are computed thereafter, e.g.

$$\begin{aligned}\mu_{s_1\to f_2}(+1) &= \frac{1}{N}\times q_1\prod_{j'\in C_{1\setminus 2}}\mu_{f_{j'}\to s_1}(+1)\\ &= \frac{1}{N}\times q_1\big(\mu_{f_3\to s_1}(+1)\big)\big(\mu_{f_4\to s_1}(+1)\big)=0.0068,\end{aligned}$$

and

$$\begin{aligned}\mu_{s_1\to f_2}(-1) &= \frac{1}{N}\times (1-q_1)\prod_{j'\in C_{1\setminus 2}}\mu_{f_{j'}\to s_1}(-1)\\ &= \frac{1}{N}\times (1-q_1)\big(\mu_{f_3\to s_1}(-1)\big)\big(\mu_{f_4\to s_1}(-1)\big)=0.9932,\end{aligned}$$

the rest of the external variable node messages for $s_i=\{1,-1\},\ \forall i$, after normalization are calculated similarly (according to step 4). The posterior probabilities (marginals) for every variable $s_i,\ \forall i$, are given by

$$\begin{aligned}g_{s_1}(+1) &= \mu_{f_2\to s_1}(+1)\times\mu_{f_3\to s_1}(+1)\times\mu_{f_4\to s_1}(+1)\times q_1\\ &= \mu_{s_1\to f_2}(+1)\times\mu_{f_2\to s_1}(+1)=0.0068\times 0.6718=0.0046,\end{aligned}$$

$$g_{s_1}(-1) = \mu_{s_1 \to f_2}(-1) \times \mu_{f_2 \to s_1}(-1) = 0.9932 \times 0.4382 = 0.4352.$$

The rest of the marginals are calculated similarly. Notice that the calculation of marginals has a lot of different (equivalent) ways. If we apply hard decisions to the marginals according to step 5 (Eq. 3.34), we take that the estimated codeword after one iteration is given by

$$\hat{\mathbf{c}} = [1 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; 0 \;\; 0],$$

which is equal to the transmitted codeword. The algorithm terminates, since

$$\hat{\mathbf{c}}\mathbf{H}^\top = \mathbf{0}.$$

After 1 iteration the SPA decoding algorithm restores the erroneous received codeword. Notice that SPA decoding algorithm exploits the soft information from the received posterior pmfs of the channel, $p_i(s_i),\ \forall i$, in order to provide reliability. ■

### 3.2.4 Remarks on LDPC codes

SPA decoding algorithm sometimes cannot restore the transmitted erroneous codeword. When SPA converges, it is executed for a finite number of iterations.

The FG corresponding to a LDPC code has cycles. In practice, we ignore the fact that cycles exist and perform SPA as analyzed in the previous susections. In this case, SPA is strictly suboptimal since it no longer performs maximum aposteriori (MAP) decoding. However, excellent performance in terms of bit error rate can be achieved even with the presence of cycles.

The very good performance of LDPC codes in terms of bit error rate relies firstly on that they have very sparse parity check matrix $\mathbf{H}$, and thus, the cycle lengths are very large, consequently the SPA decoding algorithm operates in almost cycle free FG. Secondly, LDPC decoding update rules account the soft information passing through the edges of FG, increasing the reliability of information.

Regular LDPC codes are "asymptotically" good [1]. However it has been shown that the irregular LDPC codes can reach very closely to Shannon limit [31] for large block lengths. If we increase the block length and simultaneously sustain the sparse structure of parity check matrix $\mathbf{H}$ the decoding algorithm can perform better. Equivalent representations of

the same code can differ significantly in terms of girth, so a particularly interesting and important problem is finding the best possible representation ([31],[27]).

### 3.2.5 Performance example of LDPC codes

We consider a regular $(504, 252)$ LDPC code over AWGN channel, using BPSK modulation. The performance is considered in terms of bit error rate (BER) as a function of $\text{SNR}_{\text{db}} = 10\log_{10}(\frac{1}{\sigma^2})$. If we increase the length of the transmitted sequence (and thus, the block length), then the bit error rate (BER) decreases, since the girth of the corresponding factor graph increases, with the cost of larger temporal and storage complexity. Therefore we note a performance-complexity trade-off regarding to SPA decoding algorithm. Notice in figure 3.8 the huge gap of the BER curve between LDPC coded system and the uncoded system.

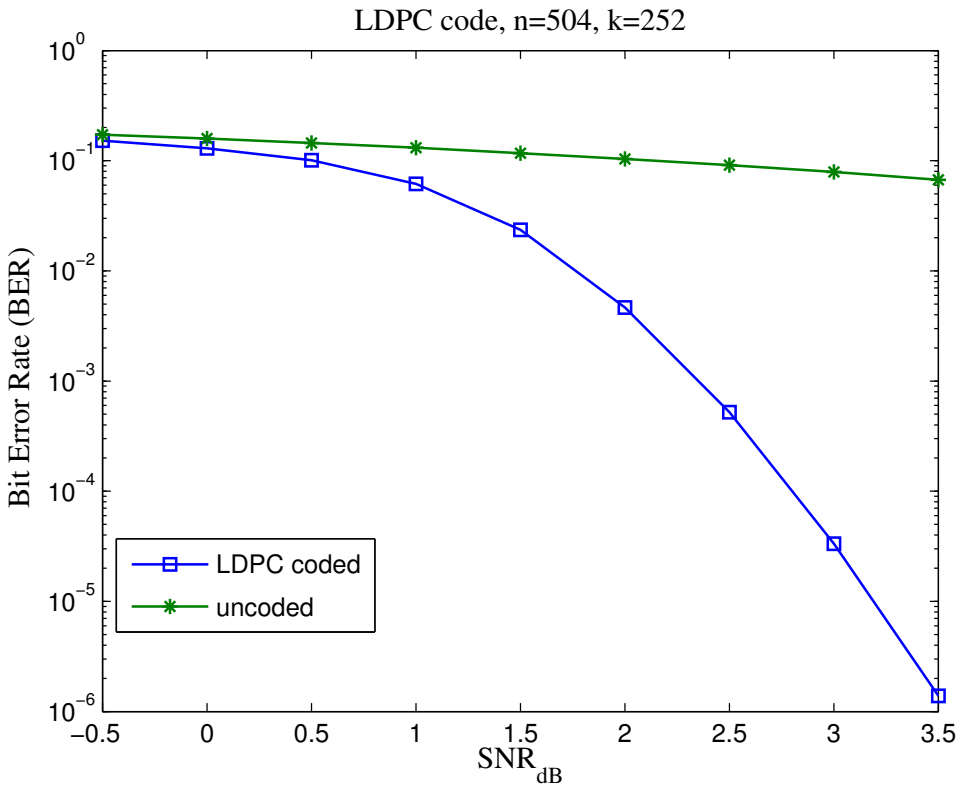



Figure 3.8: Performance of $(504, 252)$ LDPC regular code and uncoded system in terms of bit error rate (BER) as a function of $\text{SNR}_{\text{db}}$. Notice the huge gap of BER curve between them, showing the significance of error correcting codes.

## 3.3 Convolutional codes

Convolutional codes are linear block codes which consist of a finite-length shift register, which determines the *current state* and the *output* of the *encoder*.[1] They are linear, because any linear combination of the codewords generated from the encoder, belongs to the code. We assume that the shift register of the encoder has $L$ memory blocks. An example of a convolutional encoder with feedback is given in figure 3.9.

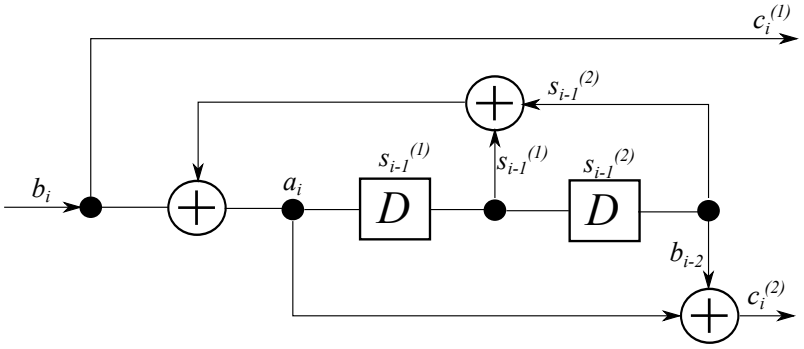



Figure 3.9: An example of a rate 1/2 linear convolutional encoder. At each time $i$, the input of the encoder is an information bit $b_i$, whereas the output is two coded-bits, denoted by $c_i^{(1)}$ and $c_i^{(2)}$. The memory length of the shift register is $L = 2$.

The code is linear since the output bits are linear combination of the input as well as the stored bits of the shift register (since we only use adders). We denote as $\mathbf{s}_{i-1}$ the state of the shift register at time $i-1$. For the example encoder of Fig. 3.9 the state at time $i-1$ consist of $L = 2$ memory state bits, namely $\mathbf{s}_{i-1} \triangleq (s_{i-1}^{(1)}, s_{i-1}^{(2)})$. The symbol $D$ inside the box stands for delay operator, i.e. $\hat{b}_i = D \times b_i \Longleftrightarrow \hat{b}_i = b_{i-1}$. The systematic form results from the fact that the first coded bit $c_i^{(1)}$ is equal to the information bit $b_i$ at any time $i$.

The theory of convolutional codes requires knowledge of fields/sequence theory, which is beyond the scope of this thesis. We will introduce convolutional codes in a simpler way, in order to construct the corresponding factor graph of the convolutional decoder. The SPA will be applied onto this decoder.

In this chapter will be considered systematic convolutional codes where every information bit corresponds to 2 coded bits. Convolutional codes usually are described via their feedback and feedforward polynomials. The feedback polynomial determines the current

[1] We will examine only recursive convolutional encoders, i.e. only those which have feedback.

state of the encoder, while the feedforward polynomial determines the non-systematic output of the encoder. The feedback and feedforward polynomials are denoted by $g_{\text{fb}}(D)$ and $g_{\text{ff}}(D)$, respectively. Note that they are polynomials in $D$ (delay operators). At any time $i$ the coefficient of $D^l$ is $l$-th state bit at time $i$ ($s_i^{(l)}$). The powers of the operators $D$ which appear at feedback and feedforward polynomials at time $i$ indicate which state bits of state $\mathbf{s}_{i-1}$ contribute in the computation of feedback output and feedforward output, respectively. Finally we must say that the coefficient $D^0 = 1$ of the feedback polynomial corresponding to the input output bit (bit $b_i$ in Fig. 3.9) and not to the current feedback output bit $a_i$.

When no feedback is used, the code is called non-recursive. To understand all these definitions above, we provide the following example.

**Example 3.5.** Consider the convolutional encoder of figure 3.9. Its length is $L = 2$. The rate of the code is equal to $1/2$. The feedback output at time $i$ (most left adder) depends on the input bit ($b_i$), the output of the first register ($s_{i-1}^{(1)}$) and the output of the second register ($s_{i-1}^{(2)}$), hence the feedback polynomial is described by

$$g_{\text{fb}}(D) = 1 + D + D^2 \ . \tag{3.36}$$

On the other hand, the non-systematic output $c_i^{(2)}$, which is the feedforward output (most right adder) depends on the feedback output $a_i$ as well as the output of the second register ($s_{i-1}^{(2)}$), therefore the feedforward polynomial is described by

$$g_{\text{ff}}(D) = 1 + D^2 \ . \tag{3.37}$$

Since there exists feedback, the encoder is recursive. ■

### 3.3.1 Encoding of convolutional codes

The following subsection is based on the corresponding chapter of [24]. Encoding of the convolutional codes is determined from the length and the form of the shift register. Essentially it is a state space system. We denote as $\mathcal{S}$, the set of all possible state values of the shift register. Consider that the length of shift register is $L$, then the set $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{\mathbf{s}_i = [s_i^{(1)} \quad \cdots \quad s_i^{(L)}]\} \ , \quad \forall i, \tag{3.38}$$

the cardinality of $\mathcal{S}$, $|\mathcal{S}|$, is equal to $2^L$. We make the convention that the state at time 0 is equal to all zero state, i.e. $\mathbf{s}_0 = \mathbf{0}_{1\times L}$. Notice the set $\mathcal{S}$ does not have a subscript, because it is the same for every state $\mathbf{s}_i$ at any time $i$.

The current state $\mathbf{s}_i$ and the current output $\mathbf{c}_i$ of the encoder at time $i$ are uniquely determined from information bit $b_i$ as well as from the previous state $\mathbf{s}_{i-1}$ of the encoder, according to feedback and the feedforward polynomials, respectively. Specifically, for the encoder of example 3.5, the update rules for output and next state at time $i$, are described as follows:

$$\mathbf{s}_i = \begin{bmatrix} a_i & s_{i-1}^{(1)} \end{bmatrix}, \tag{3.39}$$

$$c_i^{(1)} = b_i, \tag{3.40}$$

$$c_i^{(2)} = a_i + s_{i-1}^{(2)}, \tag{3.41}$$

where $a_i = b_i + s_{i-1}^{(1)} + s_{i-1}^{(2)}$ denotes the feedback output which is computed according to feedback polynomial of Eq. 3.36. Eq. 3.39 describes the calculation of the next state of encoder's shift register, which is the shifting of all bits of the shift register to the direction of the most significant bit (MSB) (most right in shift register), then puncturing the MSB, and finally insertion of the feedback output ($a_i$) at the least significant position of the shift register (most left in shift register). Eq. 3.40 describes the systematic output of the encoder, while Eq. 3.41 describes the computation of the non-systematic output of the encoder taking into account the feedback output ($a_i$) as well as the output of the second register $s_{i-1}^{(2)}$, i.e. exactly as indicates the feedforward polynomial of Eq. 3.37.

From here and now on, we will consider that the input sequence has finite number of bits and its length is equal to the number of total transmitted bits ($k$), therefore the coded output sequence will have $n = 2k$ coded bits. The shift register of every convolutional encoder can be considered as a finite state machine (FSM) with $2^L$ distinct states, providing a direct way of computing the output and the current state according to feedforward and feedback polynomials, respectively. The FSM of the encoder of the example 3.5 is depicted in figure 3.10 below.

Finally, another crucial issue in convolutional codes is the *termination*. Termination is the process, during which, we add $L$ specific bits in the information sequence, in order to predetermine the final state and to equalize it with the initial state. E.g. consider an information sequence of length $k$, $\mathbf{b}_{1:k}$. Before the transmission we add $L$ specific bits $\mathbf{b}_{(k+1):(k+L)}$ at the end of the sequence $\mathbf{b}_{1:k}$ equalizing the final and the initial states

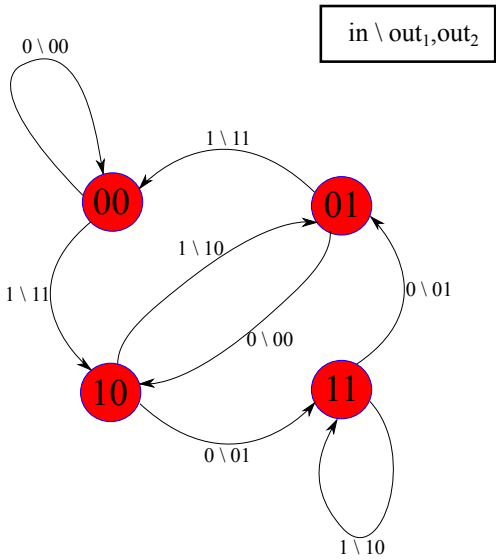



Figure 3.10: The FSM corresponding to the shift register of the example 3.5. The red cycles stand for the $2^L$ states (left bit - least significant bit, right bit - most significant bit). The arrows denote the transition from one state to another, based on the input, as well as the previous state. The numbers beside the arrows correspond to the current input (left single binary digit) and the corresponding outputs (right couple of binary digits).

($\mathbf{s}_{k+L} = \mathbf{s}_0$) in order to make the code terminated. The final information sequence for transmission is equal to $\mathbf{b}_{1:(k+L)}$.

We define as $\varrho$ the ratio of information bit per coded bits. Notice that this ratio is constant, since the number of coded bits corresponding to an information bit is fixed.

A code which is not terminated is called *unterminated*. The codewords of an unterminated code can be constructed by puncturing the last $\frac{L}{\varrho}$ bits of a codeword of a terminated code. In unterminated codes the rate $r$ is equal to $\varrho$, due to the absence of extra bits. For terminated codes, $\varrho$ is unequal to the rate of the code $r$ (i.e. $r \neq \varrho$), since the rate is the number of information bits divided with the total transmitted bits, i.e.

$$r = \frac{k}{n + \frac{L}{\varrho}}, \tag{3.42}$$

which is unequal to $\varrho$. Notice that when we use terminated codes we have a rate loss.

The following example illustrates the encoding phase of convolutional codes.

**Example 3.6.** Consider the linear convolutional code of example 3.5. The number $\varrho$ is equal to 1/2, since for 1 information bit we utilize 2 coded bits. The set $\mathcal{S}$ consist of all

possible binary 2-tuples, since $L = 2$. Specifically,

$$\mathcal{S} = \{\mathbf{s}_i = [s_i^{(1)} \quad s_i^{(2)}], \quad s_i^{(1)}, s_i^{(2)} \in \mathbb{B}\}, \quad \forall i,$$

i.e

$$\mathcal{S} = \{[0 \quad 0], [0 \quad 1], [1 \quad 0], [1 \quad 1]\}, \quad \forall i,$$

Suppose that we want to send the information sequence $\mathbf{b} = [1 \quad 0 \quad 1 \quad 0]$. As we stated above, we initialize the shift register to the all zero state, namely

$$\mathbf{s}_0 = [s_0^{(1)} \quad s_0^{(2)}] = [0 \quad 0].$$

At the first time instance corresponding to the first bit of the information sequence $\mathbf{b}$, $b_1 = 1$, we calculate the coded output as well as the current state. Namely,

$$\begin{aligned} a_1 &= b_1 + s_0^{(1)} + s_0^{(2)} = 1 + 0 + 0 = 1, \\ \mathbf{s}_1 &= [s_1^{(1)} \quad s_1^{(2)}] = [a_1 \quad s_0^{(1)}] = [1 \quad 0], \\ c_1^{(1)} &= b_1 = 1, \\ c_1^{(2)} &= a_1 + s_0^{(2)} = 1 + 0 = 1. \end{aligned}$$

At time $i = 2$, the 2nd bit is $b_2 = 0$ and the previous state is $\mathbf{s}_1 = [1 \quad 0]$, and thus,

$$\begin{aligned} a_2 &= b_2 + s_1^{(1)} + s_1^{(2)} = 0 + 1 + 0 = 1, \\ \mathbf{s}_2 &= [a_2 \quad s_1^{(1)}] = [1 \quad 1], \\ c_2^{(1)} &= b_2 = 0, \\ c_2^{(2)} &= a_2 + s_1^{(2)} = 1 + 0 = 1. \end{aligned}$$

At time $i = 3$, the 3rd bit is $b_3 = 1$ and the previous state is $\mathbf{s}_2 = [1 \quad 1]$, we take

$$\begin{aligned} a_3 &= b_3 + s_2^{(1)} + s_2^{(2)} = 1 + 1 + 1 = 1, \\ \mathbf{s}_3 &= [a_3 \quad s_2^{(1)}] = [1 \quad 1], \\ c_3^{(1)} &= b_3 = 1, \\ c_3^{(2)} &= a_3 + s_2^{(2)} = 1 + 1 = 0. \end{aligned}$$

At time $i = 4$, the 4th bit is $b_4 = 0$ and the previous state is $\mathbf{s}_3 = [1 \;\; 1]$, therefore

$$\begin{aligned} a_4 &= b_4 + s_3^{(1)} + s_3^{(2)} = 0 + 1 + 1 = 0, \\ \mathbf{s}_4 &= [a_4 \;\; s_3^{(1)}] = [0 \;\; 1], \\ c_4^{(1)} &= b_4 = 0, \\ c_4^{(2)} &= a_4 + s_3^{(2)} = 0 + 1 = 1. \end{aligned}$$

If we arrange the coded-bits as

$$\mathbf{c} = [c_1^{(1)} \; c_1^{(2)} \;\; c_2^{(1)} \;\; c_2^{(2)} \;\; c_3^{(1)} \;\; c_3^{(2)} \;\; c_4^{(1)} \;\; c_4^{(2)}] = [1 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 0 \;\; 0 \;\; 1],$$

we take the codeword corresponding to an unterminated convolutional code. If we want to obtain a terminated code, we add $L = 2$ termination bits. The rate of the unterminated code is $r = \varrho = \frac{1}{2}$. The last state of the unterminated code is the $\mathbf{s}_4 = [0 \;\; 1]$ state. According to FSM of figure 3.10, if we add the suitable terminated bits, that is $\mathbf{b}_{(k+1):(k+L)} = [1 \;\; 0]$, we equalize the final state $\mathbf{s}_6$ with the initial state $\mathbf{s}_0$ (i.e. $\mathbf{s}_6 = \mathbf{s}_0 = [0 \;\; 0]$). Therefore at time $i = 5$ the 5th bit is $b_5 = 1$ and the previous state is $\mathbf{s}_4 = [0 \;\; 1]$ and hence

$$\begin{aligned} a_5 &= b_5 + s_4^{(1)} + s_4^{(2)} = 1 + 0 + 1 = 0, \\ \mathbf{s}_5 &= [a_5 \;\; s_4^{(1)}] = [0 \;\; 0], \\ c_5^{(1)} &= b_5 = 1, \\ c_5^{(2)} &= a_5 + s_4^{(2)} = 0 + 1 = 1. \end{aligned}$$

Finally at time $i = 6$ the 6th bit is $b_6 = 0$ and the previous state is $\mathbf{s}_5 = [0 \;\; 0]$, consequently

$$\begin{aligned} a_6 &= b_6 + s_5^{(1)} + s_5^{(2)} = 0 + 0 + 0 = 0, \\ \mathbf{s}_6 &= [a_6 \;\; s_5^{(1)}] = [0 \;\; 0] = \mathbf{s}_0, \\ c_6^{(1)} &= b_6 = 0, \\ c_6^{(2)} &= a_6 + s_5^{(2)} = 0 + 0 = 0. \end{aligned}$$

The transmitted codeword corresponding to the terminated code is the following

$$\begin{aligned} \mathbf{c} &= [c_1^{(1)} \; c_1^{(2)} \;\; c_2^{(1)} \;\; c_2^{(2)} \;\; c_3^{(1)} \;\; c_3^{(2)} \;\; c_4^{(1)} \;\; c_4^{(2)} \;\; c_5^{(1)} \;\; c_5^{(2)} \;\; c_6^{(1)} \;\; c_6^{(2)}], \\ &= [1 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \;\; 0 \;\; 0]. \end{aligned}$$

The rate of the terminated code is

$$r = \frac{4}{8+4} = \frac{1}{3} \, .$$

■

### 3.3.2 Decoding of convolutional codes

Consider a rate $r$ convolutional code $\mathcal{C}$, with parameter $\varrho = \frac{1}{2}$ and with a shift register of length $L$. For design purposes, we define the following functions:

$$\mathbf{s}_i \stackrel{\triangle}{=} f_s(\mathbf{s}_{i-1}, b_i) = [a_i \;\; \mathbf{s}_{i-1}^{(1:L-1)}], \tag{3.43}$$

$$\mathbf{c}_i \stackrel{\triangle}{=} f_c(\mathbf{s}_{i-1}, b_i) = [c_i^{(1)} \;\; c_i^{(2)}] = [b_i \;\; h(b_i, \mathbf{s}_{i-1})], \tag{3.44}$$

where $a_i$, $\mathbf{s}_i$, $c_i^{(1)}$ and $c_i^{(2)}$ are defined according to previous subsection (see Eqs. 3.39, 3.40, 3.41). The abbreviation $\mathbf{s}_{i-1}^{(1:L-1)}$ stands for the truncated state $\mathbf{s}_{i-1}$ at the most significant bit $s_i^{(L)}$, namely $\mathbf{s}_{i-1}^{(1:L-1)} = [s_{i-1}^{(1)} \;\; \cdots \;\; s_{i-1}^{(L-1)}]$. We define

$$a_i = [b_i \quad \mathbf{s}_{i-1}]\mathbf{u}_{\text{fb}}^{\top} \stackrel{\triangle}{=} v(b_i, \mathbf{s}_{i-1}), \tag{3.45}$$

and

$$c_i^{(2)} = [a_i \quad \mathbf{s}_{i-1}]\mathbf{u}_{\text{ff}}^{\top} = [v(b_i, \mathbf{s}_{i-1}) \quad \mathbf{s}_{i-1}]\mathbf{u}_{\text{ff}}^{\top} \stackrel{\triangle}{=} h(b_i, \mathbf{s}_{i-1}), \tag{3.46}$$

where the $\mathbf{u}_{\text{fb}}$ and $\mathbf{u}_{\text{ff}}$ are binary row vectors, whose elements depend on the coefficients of delay operators $D$ of feedback and feedforward polynomials, respectively. E.g. for the code of example 3.5 we have

$$\mathbf{u}_{\text{fb}} = [1 \;\; 1 \;\; 1],$$

since the feedback polynomial of example 3.5 has the numbers 1, 1 and 1, as coefficients of delay operators $D^0(=1)$, $D^1$ and $D^2$, respectively. Similarly

$$\mathbf{u}_{\text{ff}} = [1 \;\; 0 \;\; 1].$$

The fact that $a_i$ and $c_i^{(2)}$ depend linearly from input $b_i$ and previous state $\mathbf{s}_{i-1}$ indicates that the functions $v(\cdot)$ and $h(\cdot)$ are linear.

The decoding algorithm relies on state space models. Specifically, we construct a *trellis* block diagram corresponding to a code $\mathcal{C}$ and then we apply the decoding algorithm for convolutional codes which will be discussed subsequently. The following example illustrates how a trellis block diagram is created.

**Example 3.7.** Consider the convolutional code of example 3.6. As we stated in previous example, the initial state is the zero state, i.e. $\mathbf{s}_0 = [0 \;\; 0]$, hence is the unique state of time 0. At time 1 the possible states are the ones which can result by state $\mathbf{s}_0 = [0 \;\; 0]$ as next states, namely the state $\mathbf{s}_1 = [0 \;\; 0]$ if the input bit at time 1 is $b_1 = 0$, while $\mathbf{s}_1 = [1 \;\; 0]$ if the input bit at time 1 is $b_1 = 1$. The calculation of all possible states at every time $i$ is computed similarly. If we consider that the code is terminated the corresponding trellis diagram for 4 information bits ($k = 4$) plus 2 terminated bits is depicted in figure 3.11. Notice that the transitions among states resulting according to FSM figure (3.10). For spatial limitations, we omit the input and the output for every transition. ■

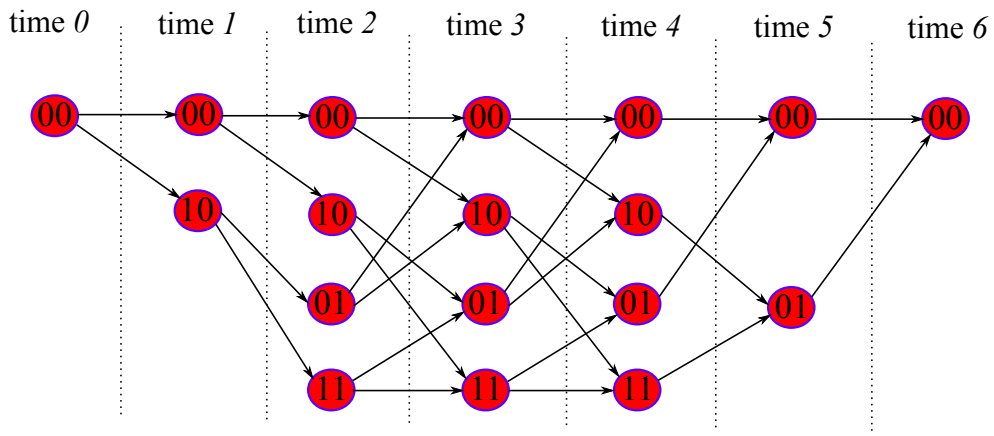



Figure 3.11: The trellis diagram corresponding to the terminated code of the example 3.6. The transitions between previous and current states following according to FSM of figure 3.10. Every time $i$ corresponds to an information bit $b_i$. The first 4 time instances associated with the information word bits, while the two additional time instances associated with terminated information bits (e.g. example 3.6). Notice that the utilization of the terminated code requires the first and the final states be equal.

Having the trellis diagram we are able to construct the factor graph corresponding to $\mathcal{C}$. We further assume a memoryless channel as defined in the subsection 3.2.3. The process of constructing the FG for convolutional decoding consist of the following steps:

- For every information bit $b_i$ of the sequence, we create a variable node with the same label;
- for every output $\mathbf{c}_i = [c_i^{(1)} \;\; c_i^{(2)}]$ we create 2 variable nodes $c_{2i-1}$ and $c_{2i}$, respectively;
- for every state $\mathbf{s}_i$ we create a variable node with the same label;
- in case of terminated code we create two extra factors $g_0(\mathbf{s}_0)$ and $g_{k+L}(\mathbf{s}_{k+L})$ to impose the first and the final states be equal. If we set as initial state the all zero state, then the factors $g_0$ and $g_{k+L}$ are

$$g_0(\mathbf{s}_0) = \begin{cases} 1, & \text{if } \mathbf{s}_0 = [0 \;\; 0], \\ 0, & \text{otherwise}, \end{cases} \tag{3.47}$$

and

$$g_{k+L}(\mathbf{s}_{k+L}) = \begin{cases} 1, & \text{if } \mathbf{s}_{k+L} = [0 \;\; 0], \\ 0, & \text{otherwise}, \end{cases} \tag{3.48}$$

In case of unterminated code there are not $L$ extra bits, hence the function $g_k$ is given by

$$g_k(\mathbf{s}_k) = \frac{1}{|\mathcal{S}|} \;, \quad \forall \mathbf{s}_k \in \mathcal{S}, \tag{3.49}$$

i.e. the final state has equiprobable its values.

- create external factor nodes ($p_i$) representing the effects of the communication channel. These factors denote the a posteriori distribution of coded bit $c_i$ transmitted given the observation $y_i$. For example, for a terminated code, if we use BPSK modulation ($0 \mapsto +1$ and $1 \mapsto -1$), assuming AWGN channel, then we have:

$$p_i(c_i) = p(c_i|y_i) = \frac{1}{1 + \exp\{-2 f_{\text{bpsk}}(c_i) y_i / \sigma^2\}} \;, \; i = 1, ..., 2(k+L), \tag{3.50}$$

where $f_{\text{bpsk}}(c_i) = -2c_i + 1$, $c_i \in \{0, 1\}$ $i = 1, ..., 2(k+L)$ (for unterminated code $i = 1, ..., 2k$);

- finally we create a factor node $f_i$, which represents the transition from the previous state to the current one as well as the calculation of the current output according to

the feedback and feedforward polynomials respectively, i.e.

$$f_i(\mathbf{s}_{i-1}, \mathbf{s}_i, b_i, c_{2i-1}, c_{2i}) = \mathbb{I}\{\ f_s(\mathbf{s}_{i-1}, b_i) = \mathbf{s}_i\ \} \times \mathbb{I}\{\ f_c(\mathbf{s}_{i-1}, b_i) = [c_{2i-1}\ \ c_{2i}]\ \}, \tag{3.51}$$

where $i = 1, ..., k+L$, for terminated code and $i = 1, ..., k$, for unterminated code. $\mathbb{I}\{\cdot\}$ denotes the indicator function of the expression inside the hooks.

In figure 3.12 is depicted the factor graph (FG) of the terminated convolutional code of the example 3.6, while in figure 3.13 is illustrated the FG for the unterminated code of the example 3.6. The one-to-one mapping from trellis diagram to FG results from the fact that the mathematical expression of trellis diagram for terminated convolutional codes (taking account the code's constraints via feedback and feedforward polynomials) is given by

$$g_0(\mathbf{s}_0)\ g_{k+L}(\mathbf{s}_{k+L}) \prod_{i=1}^{k+L} f_i(\mathbf{s}_{i-1}, \mathbf{s}_i, b_i, c_{2i-1}, c_{2i}), \tag{3.52}$$

while in the case of unterminated convolutional codes is given by

$$g_0(\mathbf{s}_0)\ g_k(\mathbf{s}_k) \prod_{i=1}^{k} f_i(\mathbf{s}_{i-1}, \mathbf{s}_i, b_i, c_{2i-1}, c_{2i}), \tag{3.53}$$

Notice that Eqs. 3.52 and 3.53 have one-to-one correspondence to FGs of figures 3.12 and 3.13, respectively (assuming that channel effects are not accounted).

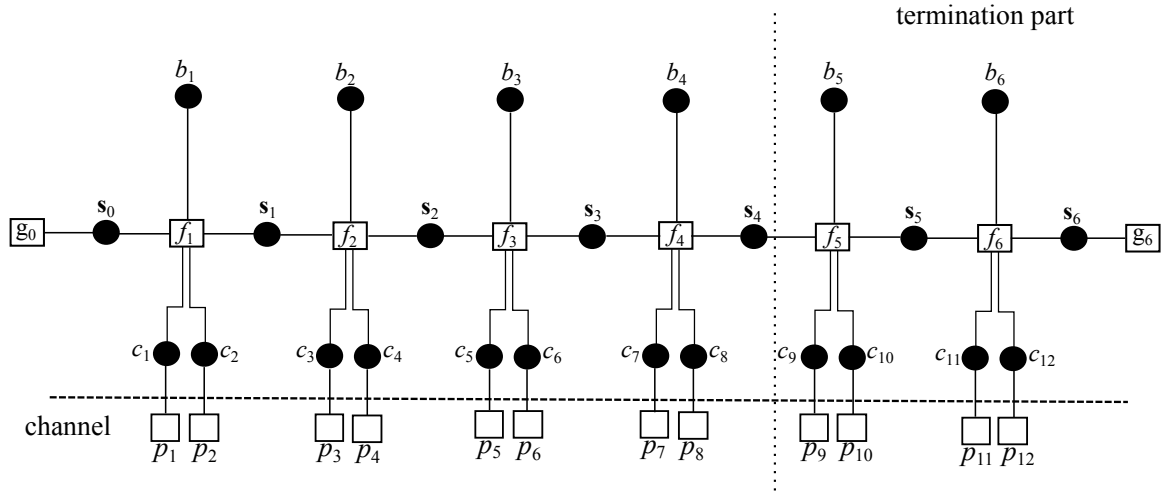



Figure 3.12: The factor graph of the terminated convolutional code of the example 3.6.

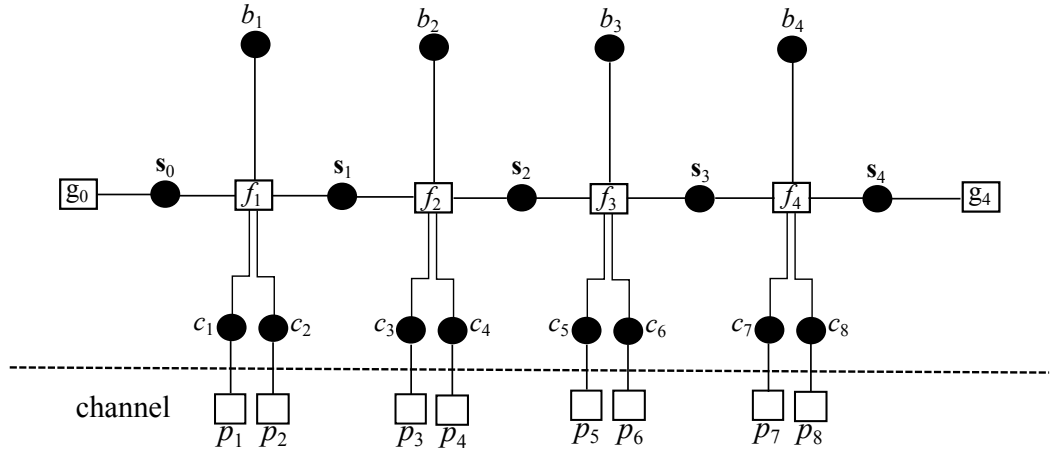



Figure 3.13: The factor graph of the unterminated convolutional code of the example 3.6.

Having the factor graph of the convolutional code $\mathcal{C}$, we are able to perform decoding via SPA. The notation of the SPA messages (vectors) is given below

- $k$ denotes the length of information sequence, $L$ denotes the length of the shift register, $\mathcal{S}$ denotes the set of all values of every state $\mathbf{s}_i$;
- the message from information bit variable node $b_i$ to factor node $f_i$ is denoted by $\mu_{b_i \longrightarrow f_i}(b_i)$, while the message $\mu_{f_i \longrightarrow b_i}(b_i)$ represents the opposite direction message;
- similarly, variable nodes of coded bits $c_{2i-1}$ and $c_{2i}$ correspond to information bit $b_i$ and are neighboring to factor node $f_i$. The messages from $c_{2i-1}$ and $c_{2i}$ to $f_i$ are denoted by $\mu_{c_{2i-1} \longrightarrow f_i}(c_{2i-1})$ and $\mu_{c_{2i} \longrightarrow f_i}(c_{2i})$, respectively. The opposite direction messages are denoted by $\mu_{f_i \longrightarrow c_{2i-1}}(c_{2i-1})$ and $\mu_{f_i \longrightarrow c_{2i}}(c_{2i})$, respectively;
- the forward message from the variable node $\mathbf{s}_{i-1}$ to factor node $f_i$ is denoted by $\mu_{\mathbf{s}_{i-1} \longrightarrow f_i}(\mathbf{s}_{i-1})$. With similar way, the forward message from the factor node $f_i$ to variable node $\mathbf{s}_i$ is denoted by $\mu_{f_i \longrightarrow \mathbf{s}_i}(\mathbf{s}_i)$. The backward messages, which have the opposite direction of the aforementioned messages are denoted $\mu_{f_i \longrightarrow \mathbf{s}_{i-1}}(\mathbf{s}_{i-1})$ and $\mu_{\mathbf{s}_i \longrightarrow f_i}(\mathbf{s}_i)$, respectively;
- the messages from factors $g_0$, $g_{k+L}$ for the terminated code and $g_0$, $g_k$ for the unterminated code to their corresponding neighboring variable nodes are denoted by $\mu_{g_0 \longrightarrow \mathbf{s}_0}(\mathbf{s}_0)$, $\mu_{g_{k+L} \longrightarrow \mathbf{s}_{k+L}}(\mathbf{s}_{k+L})$, $\mu_{g_0 \longrightarrow \mathbf{s}_0}(\mathbf{s}_0)$ and $\mu_{g_k \longrightarrow \mathbf{s}_k}(\mathbf{s}_k)$, respectively;

- finally, the message from factor node $p_i$ to variable node $c_i$ is denoted $\mu_{p_i \longrightarrow c_i}(c_i)$ and stands for the aposteriori distribution of code bit $c_i$ transmitted given the corresponding observation $y_i$.

The sum-product decoding algorithm for convolutional codes has the following steps:

1. For a terminated convolutional code the first and the last state must be equal, therefore we initialize the messages from factor node $g_0$ and $g_{k+L}$ according to equations 3.47 and 3.48, respectively. For an unterminated code the messages from factors $g_0$ and $g_k$ are initialized according to equations 3.47 and 3.49, respectively.

2. Initialization of external messages, i.e.

$$\mu_{b_i \longrightarrow f_i}(b_i) := [\frac{1}{2} \quad \frac{1}{2}]^\top, \tag{3.54}$$

$$\mu_{c_{2i-1} \longrightarrow f_i}(c_{2i-1}) := \mu_{p_{2i-1} \longrightarrow c_{2i-1}}(c_{2i-1}), \tag{3.55}$$

$$\mu_{c_{2i} \longrightarrow f_i}(c_{2i}) := \mu_{p_{2i} \longrightarrow c_{2i}}(c_{2i}), \tag{3.56}$$

   for $i = 1, ..., (k + L)$, for terminated code, and $i = 1, ..., k$, for unterminated code;

3. for $i = 1, ..., k + L$ ($i = 1, ..., k$, for unterminated code), we calculate the forward messages, i.e.

$$\begin{aligned}
\mu_{f_i \longrightarrow \mathbf{s}_i}(\mathbf{s}_i) =& \frac{1}{N_1} \sum_{\mathbf{s}_{i-1} \in \mathcal{S}} \sum_{b_i \in \mathbb{B}} \sum_{\mathbf{c}_i} f_i\big(\mathbf{s}_{i-1}, \mathbf{s}_i, b_i, c_{2i-1}, c_{2i}\big) \times \\
& \times \big( \mu_{\mathbf{s}_{i-1} \longrightarrow f_i}(\mathbf{s}_{i-1})\ \mu_{c_{2i-1} \longrightarrow f_i}(c_{2i-1})\ \mu_{c_{2i} \longrightarrow f_i}(c_{2i})\ \mu_{b_i \longrightarrow f_i}(b_i) \big) \\
=& \frac{1}{N_1} \sum_{\mathbf{s}_{i-1} \in \mathcal{S}} \sum_{b_i \in \mathbb{B}} \sum_{\mathbf{c}_i} \big( \mathbb{I}\{\ f_s(\mathbf{s}_{i-1}, b_i) = \mathbf{s}_i\ \} \times \mathbb{I}\{\ f_c(\mathbf{s}_{i-1}, b_i) = [c_{2i-1} \quad c_{2i}]\ \} \big) \times \\
& \times \big( \mu_{\mathbf{s}_{i-1} \longrightarrow f_i}(\mathbf{s}_{i-1})\ \mu_{c_{2i-1} \longrightarrow f_i}(c_{2i-1})\ \mu_{c_{2i} \longrightarrow f_i}(c_{2i})\ \mu_{b_i \longrightarrow f_i}(b_i) \big) \\
=& \frac{1}{N_1} \sum_{\mathbf{s}_{i-1} \in \mathcal{S}} \sum_{b_i \in \mathbb{B}} \big( \mathbb{I}\{\ f_s(\mathbf{s}_{i-1}, b_i) = \mathbf{s}_i\ \} \big) \times \big( \mu_{\mathbf{s}_{i-1} \longrightarrow f_i}(\mathbf{s}_{i-1}) \times \\
& \times \mu_{c_{2i-1} \longrightarrow f_i}(b_i) \times \mu_{c_{2i} \longrightarrow f_i}\big(h(b_i, \mathbf{s}_{i-1})\big) \times \mu_{b_i \longrightarrow f_i}(b_i) \big),
\end{aligned} \tag{3.57}$$

   where in the first equality we substitute the function $f_i(\mathbf{s}_{i-1}, \mathbf{s}_i, b_i, c_{2i-1}, c_{2i})$ with its equal expression (see equation 3.51), and the last equality results from the fact that

indicator function $\mathbb{I}\{\ f_c(\mathbf{s}_{i-1}, b_i) = [c_{2i-1}\ \ c_{2i}]\ \}$ is 1, only when $c_{2i-1} = b_i$ and $c_{2i} = h(b_i, \mathbf{s}_{i-1})$ according to equations 3.46 and 3.44. The number $N_1$ is a normalization constant, in order to have

$$\sum_{\mathbf{s}_i \in \mathcal{S}} \mu_{f_i \longrightarrow \mathbf{s}_i}(\mathbf{s}_i) = 1.$$

The expression $\sum_{\mathbf{c}_i}(\cdot)$ stands for abbreviation of $\sum_{c_{2i-1}} \sum_{c_{2i}}(\cdot)$.

In parallel we calculate the backward messages, setting $l = k + L - i + 1$, we take (for unterminated code is $l = k - i + 1$)

$$\begin{aligned}
\mu_{f_l \longrightarrow \mathbf{s}_{l-1}}(\mathbf{s}_{l-1}) &= \frac{1}{N_2} \sum_{\mathbf{s}_l \in \mathcal{S}} \sum_{b_l \in \mathbb{B}} \sum_{\mathbf{c}_l} f_i\big(\mathbf{s}_{l-1}, \mathbf{s}_l, b_l, c_{2l-1}, c_{2l}\big) \times \\
&\quad \times \big(\ \mu_{\mathbf{s}_l \longrightarrow f_l}(\mathbf{s}_l)\ \mu_{c_{2l-1} \longrightarrow f_l}(c_{2l-1})\ \mu_{c_{2l} \longrightarrow f_l}(c_{2l})\ \mu_{b_l \longrightarrow f_l}(b_l)\ \big) \\
&= \frac{1}{N_2} \sum_{\mathbf{s}_l \in \mathcal{S}} \sum_{b_l \in \mathbb{B}} \sum_{\mathbf{c}_l} \big(\mathbb{I}\{\ f_s(\mathbf{s}_{l-1}, b_l) = \mathbf{s}_l\ \} \times \mathbb{I}\{\ f_c(\mathbf{s}_{l-1}, b_l) = [c_{2l-1}\ \ c_{2l}]\ \}\big) \times \\
&\quad \times \big(\ \mu_{\mathbf{s}_l \longrightarrow f_l}(\mathbf{s}_l)\ \mu_{c_{2l-1} \longrightarrow f_l}(c_{2l-1})\ \mu_{c_{2l} \longrightarrow f_l}(c_{2l})\ \mu_{b_l \longrightarrow f_l}(b_l)\ \big) \\
&= \frac{1}{N_2} \sum_{\mathbf{s}_l \in \mathcal{S}} \sum_{b_l \in \mathbb{B}} \big(\mathbb{I}\{\ f_s(\mathbf{s}_{l-1}, b_l) = \mathbf{s}_l\ \}\big) \times \big(\ \mu_{\mathbf{s}_l \longrightarrow f_l}(\mathbf{s}_l) \times \\
&\quad \times \mu_{c_{2l-1} \longrightarrow f_l}(b_l) \times \mu_{c_{2l} \longrightarrow f_l}\big(h(b_i, \mathbf{s}_{i-1})\big) \times \mu_{b_l \longrightarrow f_l}(b_l)\ \big), \qquad (3.58)
\end{aligned}$$

for $i = 1, ..., k + L$, for terminated code and $i = 1, ..., k$, for unterminated code. $N_2$ is a normalization constant. The Eqs. 3.57 and 3.58 constitute the final forward-backward update rules for convolutional SPA decoding.

4. For $i = 1, ..., k + L$ ($i = 1, ..., k$, for unterminated code), we calculate the outward messages:

   - for the information bits $b_i$ we take

$$\begin{aligned}
\mu_{f_i \longrightarrow b_i}(b_i) = &\frac{1}{N_3} \sum_{\mathbf{s}_{i-1} \in \mathcal{S}} \sum_{\mathbf{s}_i \in \mathcal{S}} \sum_{\mathbf{c}_i} f_i\big(\mathbf{s}_{i-1}, \mathbf{s}_i, b_i, c_{2i-1}, c_{2i}\big) \times \\
&\times \big(\ \mu_{\mathbf{s}_{i-1} \longrightarrow f_i}(\mathbf{s}_{i-1})\ \mu_{c_{2i-1} \longrightarrow f_i}(c_{2i-1})\ \mu_{c_{2i} \longrightarrow f_i}(c_{2i})\ \mu_{\mathbf{s}_i \longrightarrow f_i}(\mathbf{s}_i)\ \big)
\end{aligned}$$

$$=\frac{1}{N_3}\sum_{\mathbf{s}_{i-1}\in\mathcal{S}}\sum_{\mathbf{s}_i\in\mathcal{S}}\sum_{\mathbf{c}_i}\big(\mathbb{I}\{\ f_s(\mathbf{s}_{i-1},b_i)=\mathbf{s}_i\ \}\times\mathbb{I}\{\ f_c(\mathbf{s}_{i-1},b_i)=[c_{2i-1}\ \ c_{2i}]\ \}\big)\times$$
$$\times\big(\ \mu_{\mathbf{s}_{i-1}\longrightarrow f_i}(\mathbf{s}_{i-1})\ \mu_{c_{2i-1}\longrightarrow f_i}(c_{2i-1})\ \mu_{c_{2i}\longrightarrow f_i}(c_{2i})\ \mu_{\mathbf{s}_i\longrightarrow f_i}(\mathbf{s}_i)\ \big),$$

if we substitute the indicator functions, we take

$$=\frac{1}{N_3}\sum_{\mathbf{s}_{i-1}\in\mathcal{S}}\big(\ \mu_{\mathbf{s}_{i-1}\longrightarrow f_i}(\mathbf{s}_{i-1})\times$$
$$\times\,\mu_{c_{2i-1}\longrightarrow f_i}(b_i)\times\mu_{c_{2i}\longrightarrow f_i}\big(h(b_i,\mathbf{s}_{i-1})\big)\times\mu_{\mathbf{s}_i\longrightarrow f_i}\big(f_s(\mathbf{s}_{i-1},b_i)\big)\ \big),\qquad(3.59)$$

where $N_3$ is a normalization constant, such that

$$\sum_{b_i\in\mathbb{B}}\mu_{f_i\longrightarrow b_i}(b_i)=1;$$

- for the systematic coded bit $c_{2i-1}$, following similar procedure as above, we take

$$\mu_{f_i\longrightarrow c_{2i-1}}(c_{2i-1})=\frac{1}{N_4}\sum_{\mathbf{s}_{i-1}\in\mathcal{S}}\sum_{b_i\in\mathbb{B}}\big(\mathbb{I}\{\ b_i=c_{2i-1}\ \}\big)\times\big(\ \mu_{\mathbf{s}_{i-1}\longrightarrow f_i}(\mathbf{s}_{i-1})\times$$
$$\times\,\mu_{b_i\longrightarrow f_i}(b_i)\times\mu_{c_{2i}\longrightarrow f_i}\big(h(b_i,\mathbf{s}_{i-1})\big)\times\mu_{\mathbf{s}_i\longrightarrow f_i}\big(f_s(\mathbf{s}_{i-1},b_i)\big)\ \big),\qquad(3.60)$$

  the number $N_4$ normalize the messages in order to sum in one;

- for parity coded bit $c_{2i}$, following similar procedure as above, we take

$$\mu_{f_i\longrightarrow c_{2i}}(c_{2i})=\frac{1}{N_5}\sum_{\mathbf{s}_{i-1}\in\mathcal{S}}\sum_{b_i\in\mathbb{B}}\big(\mathbb{I}\{\ h(b_i,\mathbf{s}_{i-1})=c_{2i}\ \}\big)\times\big(\ \mu_{\mathbf{s}_{i-1}\longrightarrow f_i}(\mathbf{s}_{i-1})\times$$
$$\times\,\mu_{b_i\longrightarrow f_i}(b_i)\times\mu_{c_{2i-1}\longrightarrow f_i}(b_i)\times\mu_{\mathbf{s}_i\longrightarrow f_i}\big(f_s(\mathbf{s}_{i-1},b_i)\big)\ \big),\qquad(3.61)$$

  and $N_5$ is a normalization constant. The Eqs. 3.59, 3.60 and 3.61 constitute the final outward update rules for convolutional SPA decoding.

5. Apply hard decisions in messages $\mu_{f_i\longrightarrow b_i}(b_i)$. Namely for every $i=1,...,k$:

$$\hat{b}_i=\begin{cases}0, & \text{if }\ \mu_{f_i\longrightarrow b_i}(0)>\mu_{f_i\longrightarrow b_i}(1),\\ 1, & \text{otherwise,}\end{cases}\qquad(3.62)$$

If we note the update rules of SPA decoding algorithm for convolutional codes we state the following comments:

- The first remark on convolutional codes is that their factor graph is cycle-free. This fact allows the application of MAP detection, in order to find the sequence with the largest likelihood [24].
- at every step of SPA decoding we have only factor node update rules, since the degree of every variable node of FG is at most 2, therefore the degree one variable nodes send their messages, whereas the degree 2 variable nodes propagate their incoming messages;
- the factor node update rules of SPA decoding are very similar with Chapter's 2 SPA update rules;
- as in case of LDPC codes, update rules for convolutional codes account the soft information passing through the edges of FG increasing the reliability of information;
- besides the application of SPA in the FG of convolutional decoder, it can also be applied the max-sum algorithm which is equivalent with Viterbi decoding algorithm.
- convolutional codes do not perform near to Shannon limit [28]. However, they consist a building block of Turbo codes which perform near to capacity. Turbo codes will be studied, subsequently.

### 3.3.3 Performance example of convolutional codes

In this subsection we examine the performance of the unterminated convolutional code of example 3.5 ($r = 1/2,\ L = 2,\ g_{\text{fb}}(D) = 1 + D + D^2,\ g_{\text{ff}}(D) = 1 + D^2$). Let an information bit sequence $\mathbf{b}$ of length equal to 1200 ($k = 1200$) transmitted over AWGN channel using BPSK modulation. The performance is considered in terms of bit error rate (BER) as a function of $\text{SNR}_{\text{db}} = 10\log_{10}(\frac{1}{\sigma^2})$. The increment of information bit sequence length $k$, leads to decrease in bit error rate (BER), and simultaneously to larger complexity. Notice in figure 3.14 the gap of the BER curve between convolutional coded system and the uncoded system. Observing figures 3.14 and 3.8 we can claim that the LDPC codes outperform the convolutional codes, since the BER curve in the first case has steep decrease in values of $\text{SNR}_{\text{db}}$, about 1 to 2 dB, while in the second case the steep decrease occurs at values of 2 to 5 dB.

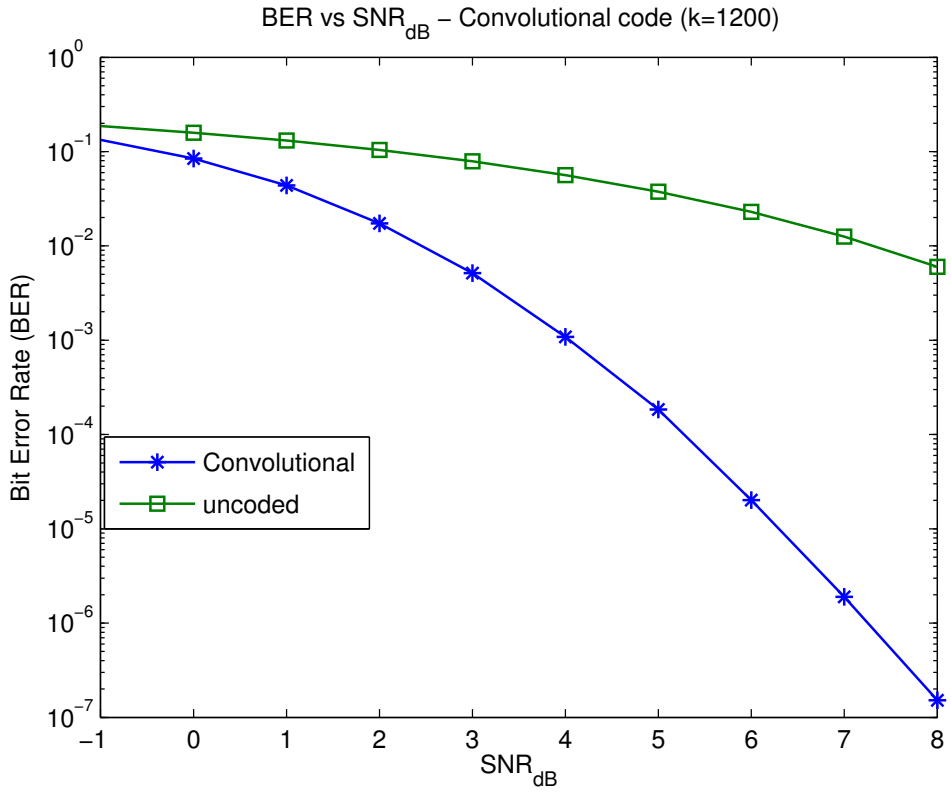



Figure 3.14: The performance of the unterminated convolutional code of example 3.5 in terms of BER vs $\text{SNR}_{\text{db}}$.

## 3.4 Turbo codes

Turbo codes are the second class of capacity-approaching codes and introduced from Berou *et al.* in [17]. The most important classes of Turbo codes are parallel concatenation of convolutional codes (PCCC) and serial concatenation of convolutional codes (SCCC). We study the first one in this thesis. In PCCC, as the name denotes, we deploy 2 convolutional encoders/decoders in parallel in order to encode/decode the information/received sequence. At this phase we will consider the convolutional encoder and decoder as building blocks in order to describe Turbo encoding and decoding. Before we start the analysis of Turbo codes, we provide the following definitions.

**Definition 3.12** (**Interleaver**)**.** Interleaver is an one-to-one function, denoted by $\pi$, which, if applied in a finite set $\mathcal{D}$, then its elements will appear in a different order, i.e. interleaver is a permutation of its indices. Consider a vector $\mathbf{d}$ of $n$ elements, i.e. $\mathbf{d} = [d_1 \quad \cdots \quad d_n]$,

then if we apply an interleaving function $\pi$ to an element $d_i$, we take

$$\tilde{d}_i = \pi(d_i) = d_{\pi(i)}, \tag{3.63}$$

where $\tilde{d}_i$ is an arbitrary element of vector $\mathbf{d}$. The element $\tilde{d}_i$ results uniquely from element $d_i$ via interleaving function $\pi$, i.e.

$$\tilde{d}_i = \pi(d_j) \Longleftrightarrow i = j. \tag{3.64}$$

The inverse function of interleaver is the *de-interleaver* and is denoted $\pi^{-1}$. Applying de-interleaving function to $\tilde{d}_i$, we take

$$d_i = \pi^{-1}(\tilde{d}_i) = \tilde{d}_{\pi^{-1}(i)}, \tag{3.65}$$

with $d_i$ being an arbitrary element of $\mathbf{d}$. The element $d_i$ results uniquely from element $\tilde{d}_i$ via de-interleaving function $\pi^{-1}$, i.e.

$$d_i = \pi^{-1}(\tilde{d}_j) \Longleftrightarrow i = j. \tag{3.66}$$

▼

The following example clarifies the notion of interleaver and de-interleaver.

**Example 3.8.** Consider a row vector $\mathbf{x} = [x_1 \;\; x_2 \;\; x_3 \;\; x_4]$. If we permute randomly the columns of the identity matrix $\mathbf{I}_4$, a possible random permutation matrix $\mathbf{P}$ is

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

If we define $\tilde{\mathbf{x}} = \pi(\mathbf{x}) = \mathbf{xP}$, with $\tilde{\mathbf{x}} = [\tilde{x}_1 \;\; \tilde{x}_2 \;\; \tilde{x}_3 \;\; \tilde{x}_4]$ and $\mathbf{xP} = [x_4 \;\; x_2 \;\; x_1 \;\; x_3]$, we take

$$\begin{aligned} \tilde{x}_1 &= \pi(x_1) = x_{\pi(1)} = x_4, \\ \tilde{x}_2 &= \pi(x_2) = x_{\pi(2)} = x_2, \\ \tilde{x}_3 &= \pi(x_3) = x_{\pi(3)} = x_1, \\ \tilde{x}_4 &= \pi(x_4) = x_{\pi(4)} = x_3, \end{aligned}$$

where the permutations of indices of the elements of vector $\mathbf{x}$ resulting according to the permutations of matrix $\mathbf{P}$. Notice that every element $\tilde{x}_i,\ i = 1, ..., 4$, results uniquely from the corresponding $x_i$ via the interleaving function $\pi$. Therefore the multiplication of a vector with a permutation matrix $\mathbf{P}$ expresses an interleaver applied on that vector.

Similarly, a de-interleaver associated with the transpose of permutation matrix $\mathbf{P}$, since $\mathbf{P}^\top = \mathbf{P}^{-1}$. If we multiply the matrix $\mathbf{P}^\top$ with the interleaved vector $\tilde{\mathbf{x}}$ we take back the vector $\mathbf{x}$, namely

$$\begin{aligned} x_1 &= \pi^{-1}(\tilde{x}_3) = \tilde{x}_{\pi^{-1}(3)}, \\ x_2 &= \pi^{-1}(\tilde{x}_2) = \tilde{x}_{\pi^{-1}(2)}, \\ x_3 &= \pi^{-1}(\tilde{x}_4) = \tilde{x}_{\pi^{-1}(4)}, \\ x_4 &= \pi^{-1}(\tilde{x}_1) = \tilde{x}_{\pi^{-1}(1)}, \end{aligned}$$

since $\pi^{-1}\big(\pi(\mathbf{x})\big) = \mathbf{x}$. Observe that the index permutations of the elements of vector $\tilde{\mathbf{x}}$ resulting according to the permutations of matrix $\mathbf{P}^\top$. ■

### 3.4.1 Encoding of PCCC Turbo codes

We define an unterminated convolutional recursive encoder $i$ based on its parameters, i.e.

$$\mathrm{CE}(r, g_{\mathrm{fb}}^{(i)}(D), g_{\mathrm{ff}}^{(i)}(D)) \equiv \mathrm{CE}^{(i)}, \tag{3.67}$$

where $r$ is the rate of the encoder, $g_{\mathrm{fb}}^{(i)}(D)$ denotes its feedback polynomial and $g_{\mathrm{ff}}^{(i)}(D)$ denotes its feedforward polynomial. For example the encoder of figure 3.9 is a

$$\mathrm{CE}(\frac{1}{2}, 1 + D + D^2, 1 + D^2)$$

encoder.

A classical PCCC Turbo encoder consists of the parallel concatenation of two or more convolutional encoders. We will focus on the case of 2 encoders, which are identical and defined as above (expression 3.67). They are called parallel because they operate simultaneously. A typical turbo encoder is depicted in figure 3.15. Its rate $r$ is equal to $1/3$, since for $k$ information bits we send $n = 3k$ coded bits. The non-solid square with the symbol $\pi$ inside in figure 3.15 denotes an interleaver used by Turbo codes, in order to interleave the information sequence $\mathbf{b}$. The utilization of interleaver is a key parameter for the high

performance of Turbo codes [17].

Observing figure 3.15, we can see that at any time $i$, the information bit $b_i$ is divided in 3 parts. The first part consists of the systematic interleaved part of the second encoder $\mathrm{CE}^{(2)}$ and corresponds to the coded bit $c_{3i-2} = b_{\pi(i)}$. The second part consist of the parity interleaved part of the second encoder $\mathrm{CE}^{(2)}$ and corresponds to the coded bit $c_{3i-1}$ which is a linear function of the current interleaved bit $b_{\pi(i)}$ as well as the previous state $\mathbf{s}_{i-1}^{(2)}$ of the second encoder. The third part consist of the parity bit of the first encoder $\mathrm{CE}^{(1)}$ and corresponds to the coded bit $c_{3i-2}$ which is a linear function of the current bit $b_i$ as well as the previous state $\mathbf{s}_{i-1}^{(1)}$ of the first encoder. Finally the systematic part of the first encoder is not utilized, i.e. we can puncture the systematic output of the first encoder $\mathrm{CE}^{(1)}$.

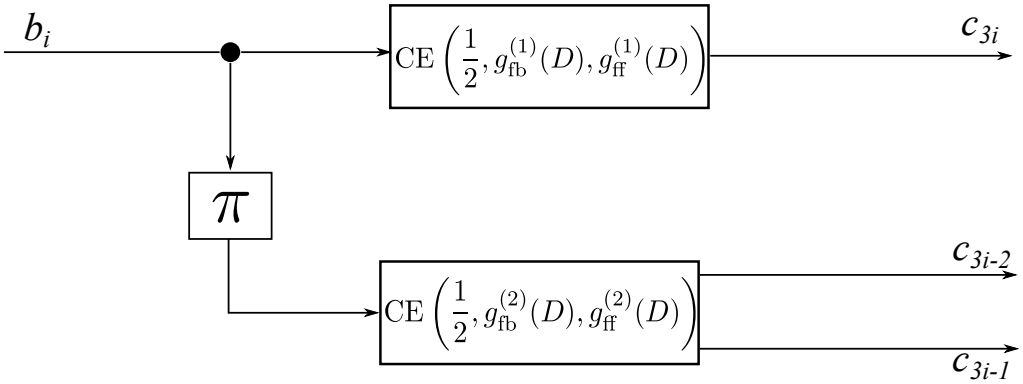



Figure 3.15: A rate $r = 1/3$ PCCC Turbo encoder, consisting of 2 identical parallel concatenated rate-1/2 convolutional encoders.

Another variant of the Turbo encoder is to use the systematic output of $\mathrm{CE}^{(1)}$, $c_i^{(1)}$, as input to the interleaver instead of information bit $b_i$, which is equivalent to the initial schema (Fig. 3.15), since $c_i^{(1)} = b_i, \ \forall i$.

### 3.4.2 Decoding of PCCC Turbo codes

A PCCC Turbo decoder, following the same idea with the Turbo encoder, consisting of 2 parallel concatenated unterminated convolutional decoders and a single interleaver. Having the convolutional decoders we are able to express them as factor graphs, in order to perform Turbo decoding by applying sum-product algorithm (SPA). We will use a convolutional decoder (factor graph) as building block, in order to construct the turbo decoder. The (FG) decoder for PCCC Turbo codes is depicted in figure 3.16

Before introducing the Turbo decoding algorithm via SPA, we will clarify the structure of factor graph of Turbo decoder. Firstly, observe that there are two 1/2 unterminated convolutional decoders, separated by an interleaver, labeled $\pi$. Let us denote them $\text{CD}^{(1)}$ and $\text{CD}^{(2)}$, respectively. We suppose that the interleaver is pseudo-random and it can perform interleaving as well as de-interleaving. Moreover, for every systematic bit output of the first decoder (after renaming $c_{2i-1}^{(1)} \triangleq d_i$) and information bit of the second one, we have the following mapping:

$$c_{2i-1}^{(1)} = d_i = \pi^{-1}(b_i^{(2)}) = b_{\pi^{-1}(i)}^{(2)}, \quad i = 1, ...k, \tag{3.68}$$

and simultaneously

$$b_i^{(2)} = \pi(d_i) = d_{\pi(i)}, \quad i = 1, ...k. \tag{3.69}$$

The renaming is performed, because we want the indexes of the interleaver take the values from 1 to $k$ (while $c_{2i-1}^{(1)}$ has index values bigger than $k$), hence variable $d_i$ is uniquely mapped to variable $b_{\pi^{-1}(i)}^{(2)}$ via de-interleaving and simultaneously variable $b_i^{(2)}$ is uniquely mapped to variable $d_{\pi(i)}$ via interleaving. The binary variables $b_1^{(1)}, ..., b_k^{(1)}$, correspond to the information bits and ideally, it is desirable, the output of the designed decoding algorithm (the output is the estimated bit sequence corresponding to information bit sequence) be equal with them. Finally the messages from factor nodes $p_i$ $(i = 1, ..., 3k)$ to their neighboring variable nodes denote the a posteriori distributions of the variables having a specific value (binary values) given the channel observations.

Observe that the factor graph contains cycles, therefore the decoding must be performed iteratively via SPA until a predetermined number of iterations is reached. Below is presented the SPA for Turbo decoding, and in parallel we show the messages on figure 3.17 of every distinct step, with different colored arrows and the index of the corresponding step next to them. The steps of the SPA PCCC-Turbo decoding algorithm are the following:

1. Set a counter $count = 0$. Initialize the messages, for $i = 1, ..., k$:

$$\mu_{b_i^{(1)} \longrightarrow f_i^{(1)}}(b_i^{(1)}) = [\frac{1}{2} \quad \frac{1}{2}]^\top,$$

and

$$\mu_{b_i^{(2)} \longrightarrow f_i^{(2)}}(b_i^{(2)}) = [\frac{1}{2} \quad \frac{1}{2}]^\top,$$

and for $i = 1, ..., k$:

$$\mu_{c_{2i}^{(1)} \longrightarrow f_i^{(1)}}(c_{2i}^{(1)}) := \mu_{p_{3i} \longrightarrow c_{2i}^{(1)}}(c_{2i}^{(1)}) = p(c_{2i}^{(1)} \mid y_{3i}),$$
$$\mu_{c_{2i-1}^{(2)} \longrightarrow f_i^{(2)}}(c_{2i-1}^{(2)}) := \mu_{p_{3i-2} \longrightarrow c_{2i-1}^{(2)}}(c_{2i-1}^{(2)}) = p(c_{2i-1}^{(2)} \mid y_{3i-2}),$$
$$\mu_{c_{2i}^{(2)} \longrightarrow f_i^{(2)}}(c_{2i}^{(2)}) := \mu_{p_{3i-1} \longrightarrow c_{2i}^{(2)}}(c_{2i}^{(2)}) = p(c_{2i}^{(2)} \mid y_{3i-1});$$

2. perform decoding on second convolutional decoder $\mathrm{CD}^{(2)}$ according to algorithm in subsection 3.3.2. The output of the SPA convolutional decoding algorithm corresponding to information bits of the $\mathrm{CD}^{(2)}$ are the following messages:

$$\mu_{f_i^{(2)} \longrightarrow b_i^{(2)}}(b_i^{(2)}), \quad i = 1, ..., k;$$

3. propagate the messages $\mu_{f_i^{(2)} \longrightarrow b_i^{(2)}}(b_i^{(2)})$ for $i = 1, ..., k$, to interleaver, namely,

$$\mu_{b_i^{(2)} \longrightarrow \pi}(b_i^{(2)}) = \mu_{f_i^{(2)} \longrightarrow b_i^{(2)}}(b_i^{(2)}), \quad i = 1, ..., k.$$

Then we de-interleave the messages above, taking

$$\mu_{\pi \longrightarrow d_i}(d_i) = \pi^{-1}\left(\mu_{b_i^{(2)} \longrightarrow \pi}(b_i^{(2)})\right), \quad i = 1, ..., k.$$

The de-interleaving can be performed since the function $\mu_{b_i^{(2)} \longrightarrow \pi}(b_i^{(2)})$ can be seen as a vector with size 2, because of $b_i^{(2)} \in \mathbb{B}$ and $|\mathbb{B}| = 2$. Set $c_{2i-1}^{(1)} = d_i,\ i = 1, ..., k$.

4. Propagation of messages $\mu_{\pi \longrightarrow c_{2i-1}^{(1)}}(c_{2i-1}^{(1)})$, i.e.

$$\mu_{\pi \longrightarrow c_{2i-1}^{(1)}}(c_{2i-1}^{(1)}) = \mu_{c_{2i-1}^{(1)} \longrightarrow f_i^{(1)}}(c_{2i-1}^{(1)}), \quad i = 1, ..., k.$$

5. Having the messages $\mu_{c_{2i-1}^{(1)} \longrightarrow f_i^{(1)}}(c_{2i-1}^{(1)})$, $\mu_{c_{2i}^{(1)} \longrightarrow f_i^{(1)}}(c_{2i}^{(1)})$ and the message $\mu_{b_i^{(1)} \longrightarrow f_i^{(1)}}(c_i^{(1)})$, we apply the SPA decoding algorithm on convolutional decoder $\mathrm{CD}^{(1)}$. The outputs of the algorithm (according to Eqs. 3.59, 3.60, 3.61 of chapter 3.3.2) are the messages

$$\mu_{f_i^{(1)} \longrightarrow b_i^{(1)}}(b_i^{(1)}), \quad i = 1, ..., k,$$
$$\mu_{f_i^{(1)} \longrightarrow c_{2i-1}^{(1)}}(c_{2i-1}^{(1)}), \quad i = 1, ..., k,$$

$$\mu_{f_i^{(1)} \longrightarrow c_{2i}^{(1)}}(c_{2i}^{(1)}), \quad i = 1, ..., k.$$

Increase counter variable *count*, i.e. $count = count + 1$

- if $count = N_{iter}$ perform hard decisions on messages $\mu_{f_i^{(1)} \longrightarrow b_i^{(1)}}(b_i^{(1)})$ for $i = 1, ..., k$, and **terminate**;
- else continue.

6. Propagation of messages $\mu_{f_i^{(1)} \longrightarrow c_{2i-1}^{(1)}}(c_{2i-1}^{(1)})$, i.e.

$$\mu_{c_{2i-1}^{(1)} \longrightarrow \pi}(c_{2i-1}^{(1)}) = \mu_{f_i^{(1)} \longrightarrow c_{2i-1}^{(1)}}(c_{2i-1}^{(1)}), \quad i = 1, ..., k.$$

Set $d_i = c_{2i-1}^{(1)}$ and interleave the messages above, i.e.

$$\mu_{\pi \longrightarrow b_i^{(2)}}(b_i^{(2)}) = \pi\big(\mu_{d_i \longrightarrow \pi}(d_i)\big), \quad i = 1, ..., k.$$

7. Propagation of messages $\mu_{\pi \longrightarrow b_i^{(2)}}(b_i^{(2)})$, i.e.

$$\mu_{b_i^{(2)} \longrightarrow f_i^{(2)}}(b_i^{(2)}) = \mu_{\pi \longrightarrow b_i^{(2)}}(b_i^{(2)}), \quad i = 1, ..., k.$$

Go to step 2.

The following remarks are noted:

- All variable nodes of the FG of Turbo decoder are either degree one or degree two variable nodes.
- The soft information passing across the edges of Turbo FG increases the reliability of information.
- The use of interleaver increase dramatically the BER performance.
- The presence of cycles in Turbo decoder FG leads to an iterative algorithm, which starts the decoding on the second decoder and finishes when a predetermined number of iterations is reached. The information part of first decoder corresponds to the information bit sequence.

### 3.4.3 Performance example of PCCC Turbo codes

In this subsection we set in parallel two unterminated convolutional encoders of example 3.5 ( $\mathrm{CE}(\frac{1}{2}, 1 + D + D^2, 1 + D^2) \equiv \mathrm{CE}^{(1)} \equiv \mathrm{CE}^{(2)}$), in order to create a rate 1/3 PCCC Turbo encoder. We send an information sequence $\mathbf{b}$ of $k = 1000$ bits over AWGN channel using BPSK modulation. The performance is considered in terms of bit error rate (BER) as a function of $\mathrm{SNR}_{\mathrm{db}} = 10\log_{10}(\frac{1}{\sigma^2})$. The length of information sequence plays crucial role, since its increase implies reduction in BER, but also an increase in computational complexity. Therefore we note a performance-complexity trade-off. Figure 3.18 depicts why Turbo codes belong to capacity-approaching codes, since as we can see at 4th iteration SPA decoding algorithm at 0 dB reaches probability on the order of $2 \times 10^{-5}$. Significant role in BER performance of Turbo codes plays the choice of interleaver, since some interleaver schemes achieve performance close to Shannon limit [28], [29] (the analysis of the interleaver is beyond the scope of this thesis). Notice the huge gap of BER curve between Turbo coded system and uncoded system, even in low SNR regime.

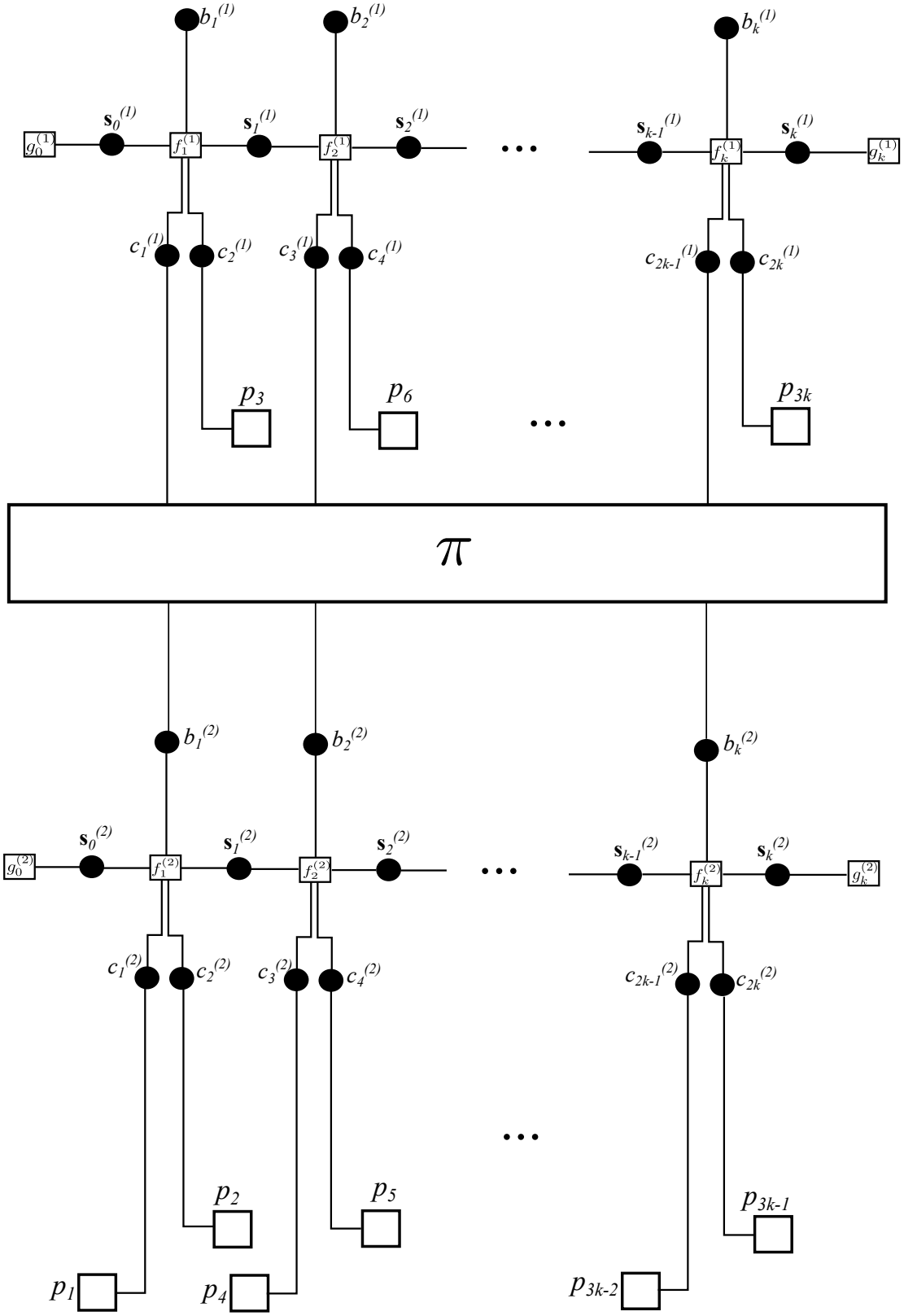



Figure 3.16: A PCCC Turbo decoder consisting of 2 unterminated convolutional decoders and an interleaver among them.

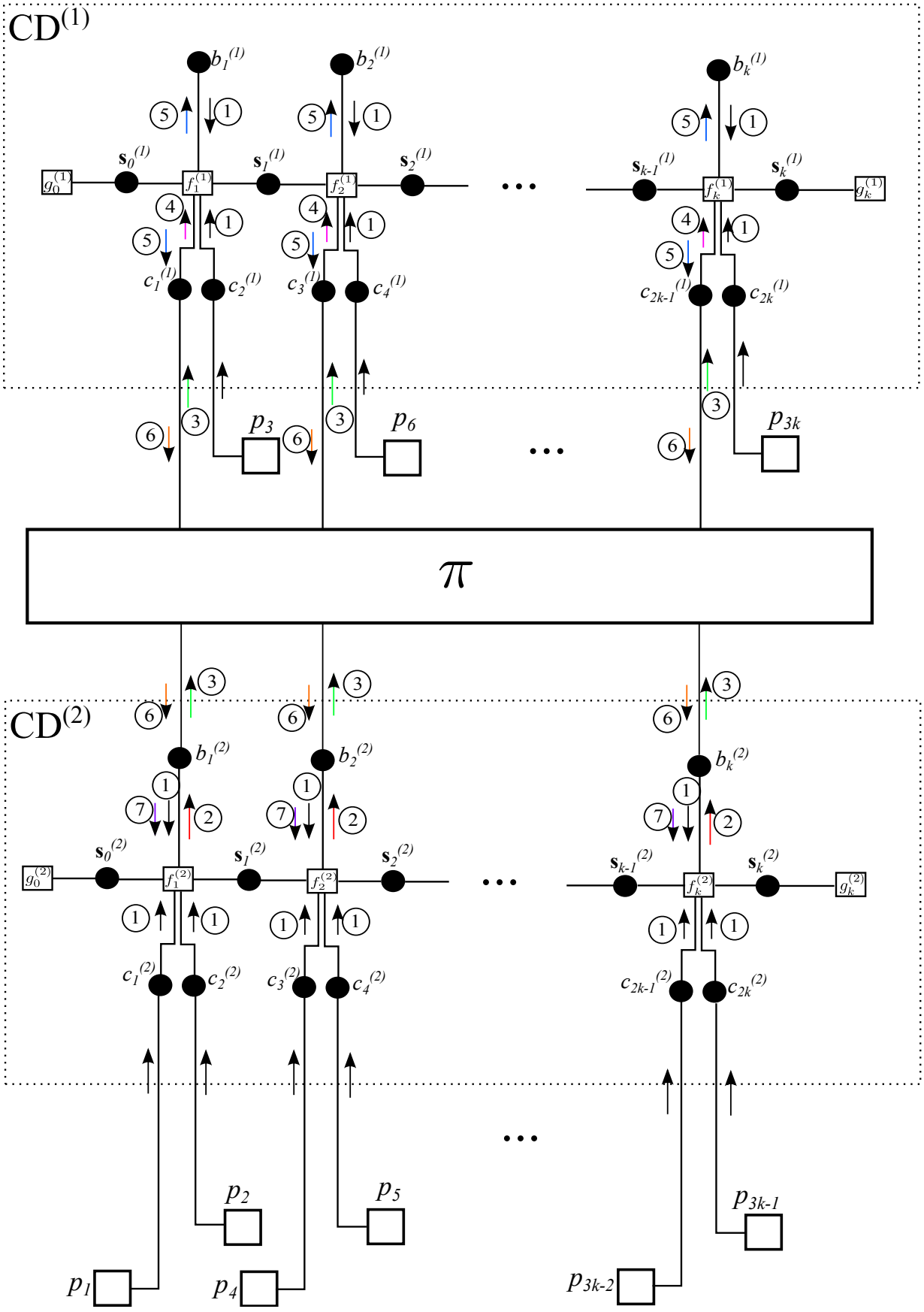



Figure 3.17: The message passing schedule during SPA algorithm, for Turbo decoding.

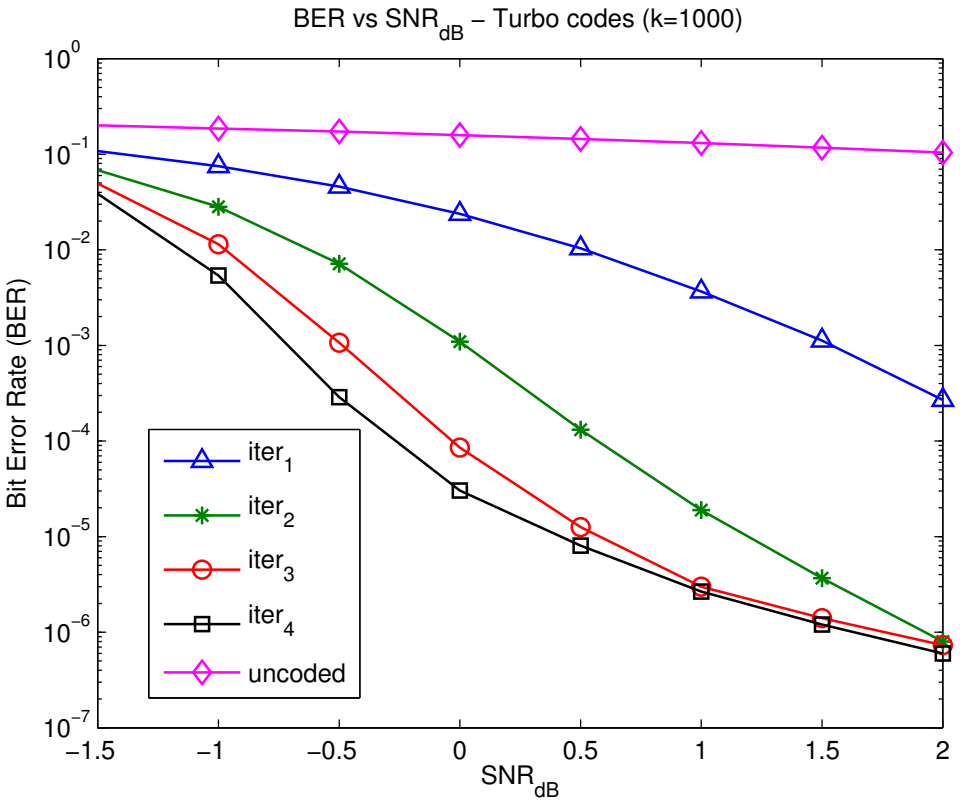



Figure 3.18: The performance of a rate 1/3 PCCC Turbo code in terms of BER vs $\mathrm{SNR_{db}}$.

# Chapter 4

# Factor Graph Applications: Cooperative Localization

This chapter provides some basic concepts about cooperative localization in wireless networks based on a factor graph approach. The material of this chapter relies on the work of Wymeersch *et al.* in [12]. At the first section we make a brief introduction about what localization is. At the next section we state some applications of the sum-product algorithm (SPA) in Bayesian estimation. In sequel, we derive the sum-product algorithm over wireless networks (aka SPAWN) providing a concluding example for clarification at the end of the section. At the final subsection we provide some experimental results as well as Cramer-Rao bound calculation according to [13], corroborating the efficiency of SPAWN algorithm. The interested reader in cooperative localization can see also [14], [15], [21], [22] and [23].

## 4.1 Introduction in localization

A major matter in wireless sensor networks is the awareness of the location of the sensor nodes composing it. Additionally, sometimes we are not only interested in the information passing along the network nodes, but also in the awareness of the location of the network nodes. Therefore new algorithms are generated trying to make the localization as possible practical, accurate, fast and costless. The localization algorithms are either *centralized* or *distributed*.

In centralized localization algorithms, all the information is collected in a base station. Base stations can determine the location of each node in the network based on the collected data, and then a estimation of node's locations is send back to them. The gathering of information is performed via wireless transmission. When the size of network increases the computational overhead as well as the delays and the energy consumption are increasing too.

On the other hand, in distributed localization algorithms the nodes can be localized

independently with a limited number of messages from other neighboring nodes, while the existence of a central station is no more needed. Therefore, in distributed localization algorithms the congestion in network nodes decreases. Another important issue of distributed algorithms is the decrease in computational overhead and in energy consumption. However the localization accuracy is sensitive when the number of network nodes increases, due to the absence of global information.

Additionally, localization can be segregated in *absolute* and in *relative*. In the first, a small subset of nodes of the network can be equipped with a small global positioning system (GPS), in order to be the reference nodes of the coordinate system. In such systems (GPS-based) the overhead is increased due to the operation of the GPS. In relative localization the coordinate system differs from neighborhood environment to neighborhood environment and every node is locally interested in distance or angle measurements between neighboring nodes.

In this thesis we will consider wireless networks which consist of nodes which know their exact location (*anchors*) and nodes which have not knowledge about their location (known as *agents*). A lot of localization techniques have proposed in literature, such as trilateration[1], triangulation[2], multilateration[3] and many others.

Localization algorithms suffer when the transmission among network nodes is affected by uncertainness. Some sources of uncertainness are the noise, the multipath effects, the fading effects and the internal errors in self measurements due to deficient mechanical parts of nodes and others.

The main purpose in wireless networks is the prediction of the exact location of agents, based on the knowledge of anchor's locations. Sometimes, in scenarios where the network nodes cannot be localized by a central system, the localization process is desirable to be executed in a distributed fashion.

Usually the network nodes exchange information via ranging measurements. When an anchor has neighboring agents send ranging measurements to them. In case where agents can *cooperate* they are able to exchange ranging measurements with other agent neighboring nodes. Cooperative localization is the process we described above and is necessary when

[1]Is the process of determining locations of points by measurement of distances, using the geometry of circles and spheres.

[2]Is the process of determining the location of a point by measuring angles to it from known points at either end of a fixed baseline, rather than measuring distances to the point directly (trilateration). The point can then be fixed as the third point of a triangle with one known side and two known angles.

[3]Multilateration is a navigation technique based on the measurement of the difference in distance to two or more stations at known locations that broadcasting signals at known times.

the anchors on the network are not many. More informations in cooperative localization can be found in [12].

The simple example below illustrates the significance of cooperation among agents.

**Example 4.1** ([12])**.** Consider a simple network with 5 sensor nodes. Three of them are anchors (e.g. base stations) and the two of them are agents (e.g. mobile units). We enumerate the three anchor nodes as nodes 1 to 3 and the rest of them (agents) are enumerated as nodes 4 and 5. We consider that agent 4 can exchange information with anchors 1 and 2, while agent 5 can exchange information with anchors 2 and 3. If we further assume that agents 4 and 5 are within communication range then they can cooperate, in order to acquire more information about their positions. Figure 4.1 depicts the network and the connectivity of the network. ■

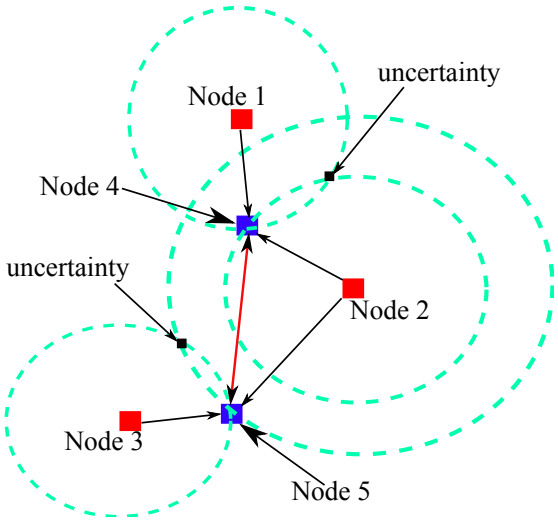



Figure 4.1: Agent node 4 can communicate with anchors 1 and 2, while agent node 5 can communicate with anchors 2 and 3. If we apply trilateration technique, the agents 4 and 5 have uncertainty regarding their location. Therefore agents 4 and 5 need to cooperate, in order to determine their position.

The proposed localization algorithm will be applied in ultra wide bandwidth (UWB) regime, since it provides resistance in multipath effects, robustness during communication and has small communication delays for distance resolution on the order of centimeters.

## 4.2 Bayesian cooperative localization

### 4.2.1 Sequential Estimation

All the content of this subsection is based on [12]. In many circumstances variables may change over the time and usually we are interested on the estimation of these variables. Such situations define the *sequential estimation*, in which we are interested on estimating variables at a certain time $t$, let, variable $x^{(t)}$, based on independent observations until time $t$, let them be denoted as $\mathbf{z}^{(1:t)} = [z^{(1)}, ..., z^{(t)}]^\top$. If we assume the following Markovian assumptions, i.e.:

$$p(x^{(t)} \mid \mathbf{x}^{(1:(t-1))}) = p(x^{(t)} \mid x^{(t-1)})$$

and

$$p(z^{(t)} \mid \mathbf{x}^{(0:t)}) = p(z^{(t)} \mid x^{(t)})$$

Then using these expressions, it can be shown that

$$p(x^{(t)} \mid \mathbf{z}^{(1:t)}) = \int_{x^{(t-1)}} p\left(x^{(t)}, x^{(t-1)} \mid \mathbf{z}^{(1:t)}\right) \mathrm{d}x^{(t-1)} \tag{4.1}$$

or equivalently

$$\begin{aligned} p\left(x^{(t)} \mid \mathbf{z}^{(1:t)}\right) &\propto p\left(x^{(t)}, z^{(t)} \mid \mathbf{z}^{(1:t-1)}\right) \\ &= p\left(z^{(t)} \mid x^{(t)}, \mathbf{z}^{(1:t-1)}\right) p\left(x^{(t)} \mid \mathbf{z}^{(1:t-1)}\right) \\ &= p\left(z^{(t)} \mid x^{(t)}\right) p\left(x^{(t)} \mid \mathbf{z}^{(1:t-1)}\right). \end{aligned} \tag{4.2}$$

Last equality results from the fact that given $x^{(t)}$, the observations $\mathbf{z}^{(1:t-1)}$ do not provide useful information. Note that the last term in 4.2 can be further factorized, namely,

$$\begin{aligned} p\left(x^{(t)} \mid \mathbf{z}^{(1:t-1)}\right) &= \int_{x^{(t-1)}} p\left(x^{(t)}, x^{(t-1)} \mid \mathbf{z}^{(1:t-1)}\right) \mathrm{d}x^{(t-1)} \\ &= \int_{x^{(t-1)}} p\left(x^{(t)} \mid x^{(t-1)}, \mathbf{z}^{(1:t-1)}\right) p\left(x^{(t-1)} \mid \mathbf{z}^{(1:t-1)}\right) \mathrm{d}x^{(t-1)} \\ &= \int_{x^{(t-1)}} p\left(x^{(t)} \mid x^{(t-1)}\right) p\left(x^{(t-1)} \mid \mathbf{z}^{(1:t-1)}\right) \mathrm{d}x^{(t-1)}, \end{aligned} \tag{4.3}$$

where in the last equality we used the fact that given the variable $x^{(t-1)}$ at time $t-1$, the information from observations until time $t-1$ ($\mathbf{z}^{(1:t-1)}$) does not provide useful information for the calculation of the conditional probability, hence it can be omitted. Therefore $p\left(x^{(t)} \mid x^{(t-1)}, \mathbf{z}^{(1:t-1)}\right) = p\left(x^{(t)} \mid x^{(t-1)}\right)$.

The expressions above indicate that the function $p(x^{(t)} \mid \mathbf{z}^{(1:t)})$ depends solely on the distribution $p(x^{(t-1)} \mid \mathbf{z}^{(1:t-1)})$ and on the distribution $p\left(z^{(t)} \mid x^{(t)}\right)$.

Consider a two step approach from which we can derive the *posterior* distribution $p(x^{(t)} \mid \mathbf{z}^{(1:t)})$ at any time $t$. Firstly, the *prediction* operation, in which we compute the distribution $p\left(x^{(t)} \mid \mathbf{z}^{(1:t-1)}\right)$ of variable $x^{(t)}$ at time $t$ given the observations until time $t-1$ using Eq. 4.3. Secondly the *correction* operation, during which we utilize the observation at time $t$, $z^{(t)}$, in order to evaluate the *posterior* distribution of the variable $x^{(t)}$ at time $t$ given all observations $\mathbf{z}^{(1:t)}$ until time $t$ using Eq. 4.2.

The *posterior* distribution $p(\mathbf{x}^{(0:T)} \mid \mathbf{z}^{(1:T)})$ using the Markovian assumptions above and accounting the independence in measurements for any time $t$ can be factorized as

$$p(\mathbf{x}^{(0:T)} \mid \mathbf{z}^{(1:T)}) \propto p\left(x^{(0)}\right) \prod_{t=1}^{T} p\left(x^{(t)} \mid x^{(t-1)}\right) p\left(z^{(t)} \mid x^{(t)}\right). \tag{4.4}$$

The Eq. 4.4 can be expressed in terms of factor graphs and by applying SPA we can compute the *posterior* distribution. Consequently, we note that FGs can be also used in sequential estimation. The following example from [12] clarifies how it is applied the SPA on the FGs corresponding to sequential estimation problems.

**Example 4.2** ([12])**.** Consider the factorization of the *posterior* distribution of Eq. 4.4 for $T = 2$, namely

$$p(\mathbf{x}^{(0:2)} \mid \mathbf{z}^{(1:2)}) \propto p\left(x^{(0)}\right) p\left(x^{(1)} \mid x^{(0)}\right) p\left(z^{(1)} \mid x^{(1)}\right) p\left(x^{(2)} \mid x^{(1)}\right) p\left(z^{(2)} \mid x^{(2)}\right),$$

the corresponding normal FG is illustrated in figure 4.2. The message flow is depicted with arrows with the direction therein. The red arrows with the symbol 'P.O.' next to them denote the messages corresponding to the prediction operation, while the purple arrows with the symbol 'C.O.' next to them represent the messages corresponding to the correction operation. Messages from leaf nodes are illustrated with black-colored arrows and the character 'L' next to them. We make the following abbreviations: $p\left(x^{(t)} \mid x^{(t-1)}\right) = f_A^{(t)}\left(x^{(t)}, x^{(t-1)}\right)$ and $p\left(z^{(t)} \mid x^{(t)}\right) = f_B^{(t)}\left(x^{(t)}\right)$, for $t = 1, ..., T$, with $p\left(x^{(0)}\right) = f_A^{(0)}\left(x^{(0)}\right)$. We will consider for convenience that the incident edges at equality nodes have the same

label since they correspond to the same variable. The steps of SPA are the following:

- step 0:

$$\mu_{f_A^{(0)} \longrightarrow X^{(0)}}\left(x^{(0)}\right) = f_A^{(0)}\left(x^{(0)}\right) = p\left(x^{(0)}\right),$$

- step 1:

$$\begin{aligned}\mu_{f_A^{(1)} \longrightarrow X^{(1)}}\left(x^{(1)}\right) &\propto \int \mu_{X^{(0)} \longrightarrow f_A^{(1)}}\left(x^{(0)}\right) \times f_A^{(1)}\left(x^{(1)}, x^{(0)}\right) \{\sim \mathrm{d}x^{(1)}\} \\ &= \int \mu_{X^{(0)} \longrightarrow f_A^{(1)}}\left(x^{(0)}\right) \times f_A^{(1)}\left(x^{(1)}, x^{(0)}\right) \mathrm{d}x^{(0)} \\ &= \int p\left(x^{(1)} \mid x^{(0)}\right) p\left(x^{(0)}\right) \mathrm{d}x^{(0)} = p\left(x^{(1)}\right), \end{aligned} \tag{4.5}$$

  and

$$\mu_{f_B^{(1)} \longrightarrow X^{(1)}}\left(x^{(1)}\right) = f_B^{(1)}\left(x^{(1)}\right) = p\left(z^{(1)} \mid x^{(1)}\right).$$

  Finally, the message from the 1st equality node to factor $f_A^{(2)}(\cdot)$ across the edge $X^{(1)}$ is given by

$$\mu_{X^{(1)} \longrightarrow f_A^{(2)}}\left(x^{(1)}\right) \propto \mu_{f_A^{(1)} \longrightarrow X^{(1)}} \times \mu_{f_B^{(1)} \longrightarrow X^{(1)}}\left(X^{(1)}\right) = p\left(z^{(1)} \mid x^{(1)}\right) p\left(x^{(1)}\right), \tag{4.6}$$

  where the expressions 4.5 and 4.6 correspond to the prediction and the correction operations at time $t = 1$, respectively.

- step 2:
  Similarly,

$$\mu_{f_A^{(2)} \longrightarrow X^{(2)}}\left(x^{(2)}\right) \propto \int \mu_{X^{(1)} \longrightarrow f_A^{(2)}}\left(x^{(1)}\right) \times f_A^{(2)}\left(x^{(2)}, x^{(1)}\right) \mathrm{d}x^{(1)}, \tag{4.7}$$

$$\mu_{f_B^{(2)} \longrightarrow X^{(2)}}\left(x^{(2)}\right) = f_B^{(2)}\left(x^{(1)}\right) = p\left(z^{(2)} \mid x^{(2)}\right),$$

$$\mu_{X^{(2)} \longrightarrow f_A^{(3)}} \propto \mu_{f_A^{(2)} \longrightarrow X^{(2)}}\left(x^{(2)}\right) \times p\left(z^{(2)} \mid x^{(2)}\right). \tag{4.8}$$

  Eq. 4.7 corresponds to the prediction operation, while Eq. 4.8 corresponds to cor-

rection operation at time $t = 2$.

Note that the flow of the messages is from past to future, since the temporal constraints must be satisfied. Furthermore, it is noted that correction operation is always followed by prediction operation. ■

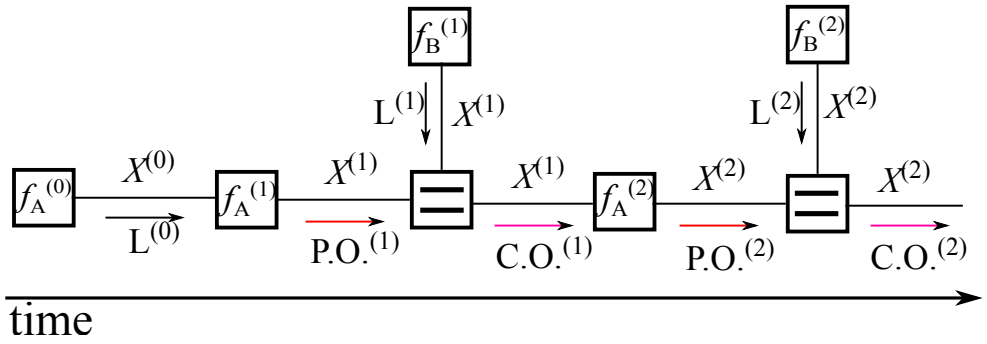



Figure 4.2: In this figure is illustrated the factorization of example 4.2. P.O.$^{(t)}$ stands for the prediction operation at time $t$, similarly C.O.$^{(t)}$ denotes the correction operation at time $t$. L$^{(t)}$ denotes the leaf node messages at time $t$.

### 4.2.2 System model

All of the material below is exclusively based on [12]. In this subsection we describe the problem formulation as well as some assumptions considered, in order to derive the cooperative distributed localization algorithm.

Assume a wireless network with $N$ nodes, where the time is slotted and the movement is independent for all nodes from their position at a time slot $t$ to a new position at time slot $t+1$. The position of node $i$ at time slot $t$ is denoted by $\mathbf{x}_i^{(t)}$ and belongs either in $\mathbb{R}^2$ or $\mathbb{R}^3$. Consider a node $i$, then the set of nodes from which node $i$ can receive information at time slot $t$ is denoted $\mathcal{S}_{\rightarrow i}^{(t)}$, while the set of nodes which can receive information from node $i$ at time slot $t$ is denoted $\mathcal{S}_{i\rightarrow}^{(t)}$. Every node $i$ can have its own measurements at time slot $t$ denoted by $\mathbf{z}_{i,\text{self}}^{t}$. A node $i$ can also receive measurements at time slot $t$ from neighboring nodes $j$ ($j \in \mathcal{S}_{\rightarrow i}^{(t)}$). These measurements are called relative measurements of node $i$ and denoted $\mathbf{z}_{i,\text{rel}}^{(t)}$ or $z_{j\rightarrow i}^{(t)}$, $\forall j \in \mathcal{S}_{\rightarrow i}^{(t)}$. Every agent $i$ desires to estimate its position $\mathbf{x}_i^{(t)}$ at time $t$ based on relative and its self measurements. All this must be performed under the presence of noise in ranging measurements as well as of multipath effect. Low

computational complexity with as small as possible overhead in the transmission and small delays are desirable for the design of the localization algorithm.

The matrix of all network node positions at time $t$ is denoted by $\mathbf{x}^{(t)}$ and is matrix since the positions belong either in $\mathbb{R}^2$ or in $\mathbb{R}^3$. The following assumptions are considered [12]:

- The positions of nodes are a priori independent, that is:

$$p(\mathbf{x}^{(0)}) = \prod_{i=1}^{N} p(\mathbf{x}_i^{(0)}). \tag{4.9}$$

- The movement of nodes follows a memoryless walk:

$$p(\mathbf{x}^{(0:T)}) = p(\mathbf{x}^{(0)}) \prod_{t=1}^{T} p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}), \tag{4.10}$$

  where the expression $\mathbf{x}^{(0:T)}$ (tensor) denotes the sequence of positions from time index 0 to time index $T$.

- Nodes move independently, i.e.:

$$p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) = \prod_{i=1}^{N} p(\mathbf{x}_i^{(t)} \mid \mathbf{x}_i^{(t-1)}), \tag{4.11}$$

- Relative measurements are independent of internal measurements given the node positions:

$$p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}, \mathbf{z}_{\text{self}}^{(1:T)}) = p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}), \tag{4.12}$$

  where $\mathbf{z}_{\text{rel}}^{(1:T)}$ denotes the relative measurements of all network nodes from time slot 1 until time slot $T$. The self measurements $\mathbf{z}_{\text{self}}^{(1:T)}$ are defined, similarly.

- Self measurements are mutually independent and depend only on the current and previous node positions, namely:

$$p(\mathbf{z}_{\text{self}}^{(1:T)} \mid \mathbf{x}^{(0:T)}) = \prod_{t=1}^{T} p(\mathbf{z}_{\text{self}}^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}). \tag{4.13}$$

- Self measurements of node $i$ depend only on the position of node $i$:

$$p(\mathbf{z}_{\text{self}}^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}) = \prod_{i=1}^{N} p(\mathbf{z}_{i,\text{self}}^{(t)} \mid \mathbf{x}_i^{(t)}, \mathbf{x}_i^{(t-1)}). \tag{4.14}$$

- Relative measurements are independent from time slot to time slot given the node positions. They depend only on the current position:

$$p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}) = \prod_{t=1}^{T} p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}). \tag{4.15}$$

- The relative measurements at any time slot $t$ are conditionally independent, depending solely on the 2 nodes involved:

$$p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}) = \prod_{i=1}^{N} \prod_{j \in \mathcal{S}_{\rightarrow i}^{t}} p(z_{j\rightarrow i}^{(t)} \mid \mathbf{x}_j^{(t)}, \mathbf{x}_i^{(t)}), \tag{4.16}$$

  where $z_{j\rightarrow i}^{(t)}$ denotes the relative measurement from node $j$ to node $i$ at time slot $t$.

The final assumption in [12] is that every node $i$ knows the following:

1. the initial position distribution $p(\mathbf{x}_i^{(0)})$;
2. the probability $p(\mathbf{x}_i^{(t)} \mid \mathbf{x}_i^{(t-1)})$, which describes the mobility model at any time $t$;
3. the self measurements $\mathbf{z}_{i,\text{self}}^{(t)}$ at any time $t$ and the corresponding likelihood function $p(\mathbf{z}_{i,\text{self}}^{(t)} \mid \mathbf{x}_j^{(t)}, \mathbf{x}_i^{(t)})$;
4. the ranging measurements from any neighboring node $j$, $z_{j\rightarrow i}^{(t)}$, and the corresponding likelihood function $p(z_{j\rightarrow i}^{(t)} \mid \mathbf{x}_j^{(t)}, \mathbf{x}_i^{(t)})$ at any time $t$;

The above assumption constitutes the local information of any node $i$ in the network at any time $t$.

### 4.2.3 Sum product algorithm over wireless networks (SPAWN)

The content of this subsection relies on [12]. In this subsection, we use the term 'node' to state a node of the network, while we use the term 'vertex' to state a node of the factor graph (FG). Our goal is to find the positions of all nodes in the network for every time $t$,

$\mathbf{x}^{(0:T)}$, based on the measurements as well as on the prior distribution of the locations of nodes, i.e. based on the vector $\mathbf{z}^{(1:T)} = [\mathbf{z}_{\text{self}}^{(1:T)^\top} \mathbf{z}_{\text{rel}}^{(1:T)^\top}]^\top$ and the pdf $p(\mathbf{x}^{(0)})$, respectively. The idea of the algorithm is to factorize the posterior distribution $p(\mathbf{x}^{(0:T)} \mid \mathbf{z}^{(1:T)})$, in order to construct the corresponding FG of that factorization. The next step is to map that FG onto the time-varying network topology. Finally, sum-product algorithm (SPA) is applied on the FG in order to perform localization.

We will see that the aforementioned FG consists of some subgraphs which contain cycles. Therefore a scheduling must be performed, in order the spatial and temporal constraints in message passing are satisfied. Finally, the application of the sum-product algorithm on that FG provides a distributed cooperative localization algorithm also known as (a.k.a.) sum-product algorithm over wireless networks (SPAWN). Regarding SPAWN, it is considered a three-step approach [12].

In the first step, we simplify the factorization of global function $p(\mathbf{x}^{(0:T)} \mid \mathbf{z}^{(1:T)})$. More specifically, according to Bayes rule we take

$$\begin{aligned} p(\mathbf{x}^{(0:T)} \mid \mathbf{z}^{(1:T)}) &\propto p(\mathbf{x}^{(0:T)}, \mathbf{z}^{(1:T)}) \\ &= p(\mathbf{x}^{(0:T)}, \mathbf{z}_{\text{rel}}^{(1:T)}, \mathbf{z}_{\text{self}}^{(1:T)}) \\ &= p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}, \mathbf{z}_{\text{self}}^{(1:T)}) p(\mathbf{x}^{(0:T)}, \mathbf{z}_{\text{self}}^{(1:T)}) \\ &\overset{4.12}{=} p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}) p(\mathbf{x}^{(0:T)}, \mathbf{z}_{\text{self}}^{(1:T)}). \end{aligned} \tag{4.17}$$

If we substitute Eqs. 4.10, 4.13 and 4.15 in Eq. 4.17 we take

$$\begin{aligned} p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}) p(\mathbf{x}^{(0:T)}, \mathbf{z}_{\text{self}}^{(1:T)}) &= p(\mathbf{z}_{\text{rel}}^{(1:T)} \mid \mathbf{x}^{(0:T)}) p(\mathbf{z}_{\text{self}}^{(1:T)} \mid \mathbf{x}^{(0:T)}) p(\mathbf{x}^{(0:T)}) \\ &\overset{4.13,4.15}{=} p(\mathbf{x}^{(0:T)}) \prod_{t=1}^{T} p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}) p(\mathbf{z}_{\text{self}}^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}) \\ &\overset{4.10}{=} p(\mathbf{x}^{(0)}) \prod_{t=1}^{T} p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}) p(\mathbf{z}_{\text{self}}^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}). \end{aligned} \tag{4.18}$$

Furthermore, due to the assumption of independence in movement and in internal measurements, utilizing Eqs. 4.9, 4.11 and 4.14 in 4.18 we take

$$(4.18) = p(\mathbf{x}^{(0)}) \prod_{t=1}^{T} p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}) p(\mathbf{z}_{\text{self}}^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}) \overset{4.9,4.11,4.14,4.16}{=}$$

$$= \prod_{i=1}^{N} p(\mathbf{x}_i^{(0)}) \prod_{t=1}^{T} p(\mathbf{x}_i^{(t)} \mid \mathbf{x}_i^{(t-1)}) p(\mathbf{z}_{i,\text{self}}^{(t)} \mid \mathbf{x}_i^{(t)}, \mathbf{x}_i^{(t-1)}) p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}). \tag{4.19}$$

The expression above could be abbreviated, namely if we create 2 new factors defined as:

$$h_i^{(t-1)}(\mathbf{X}_i^{(t)}, \mathbf{X}_i^{(t-1)}) = p(\mathbf{X}_i^{(t)} \mid \mathbf{X}_i^{(t-1)}) p(\mathbf{z}_{i,\text{self}}^{(t)} \mid \mathbf{X}_i^{(t)}, \mathbf{X}_i^{(t-1)}), \tag{4.20}$$

$$f_i(\mathbf{X}_i^{(0)}) = p(\mathbf{X}_i^{(0)}), \tag{4.21}$$

then the resulting expression becomes

$$(4.19) \overset{4.20,4.21}{=} \prod_{i=1}^{N} f_i(\mathbf{x}_i^{(0)}) \prod_{t=1}^{T} h_i^{t-1}(\mathbf{x}_i^{(t)}, \mathbf{x}_i^{(t-1)}) p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)}). \tag{4.22}$$

The factorization above can be expressed in terms of FGs. The FG corresponding to equation 4.22 is depicted in figure 4.3. Observe that the temporal flow of messages is from past to present. The vertices of the FG of figure 4.3 corresponding to the factors of Eq. 4.22 have the label of the respective factors inside them. The last term in 4.22, namely the likelihood pdf $p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)})$ can be further simplified since it is factorisable (see expression 4.16). The latter indicates which nodes of the network can communicate and represents the likelihood probability of the ranging measurements among neighboring nodes at every time $t$. The FG of the pdf $p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)})$ of the network corresponding to example 4.1 is illustrated in figure 4.4.

In the second step, we utilize the FGs of figures 4.3 and 4.4 mapping them onto the time-varying network topology. The local information for every network node consists of the mobility model and the self measurements, consequently, the factor $h_i^{(t-1)}(\mathbf{X}_i^{(t)}, \mathbf{X}_i^{(t-1)})$ holds all the local information for every node $i$ of the network. Furthermore, every equality vertex of figure 4.4 corresponds to every variable $\mathbf{X}_i^{(t)}$ which states the location of node $i$ at time $t$. For every node $j$ which can send information to node $i$ ($j \in \mathcal{S}_{\to i}^{(t)}$) we create a vertex $g_{j\to i}(\mathbf{X}_j^{(t)}, \mathbf{X}_i^{(t)}) = p(z_{j\to i}^{(t)} \mid \mathbf{X}_j^{(t)}, \mathbf{X}_i^{(t)})$ which denotes the likelihood distribution of the ranging measurements $z_{j\to i}^{(t)}$ given the locations of nodes $i$ and $j$. If we join all the vertices corresponding to every node $i$ at time $t$ according to connectivity of the network, we create the FG of the factorization $p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{X}^{(t)})$.

In the final step, the algorithm must take into account the temporal and the spatial constraints of network topology in order to derive a correct message scheduling. There are 2 type of messages, firstly, the *intranode* messages, which involve messages within a

node, secondly, the *internode* messages corresponding to the ranging measurements between neighboring nodes. In figure 4.4 the internode messages are depicted with I-red arrows, while the intranode messages are depicted with E-green arrows.

More specifically, the temporal constraints determine the message flow, which is obviously from past to present. For example in the factorization of Eq. 4.19 the calculation of term $p(\mathbf{x}_i^{(1)} \mid \mathbf{x}_i^{(0)})$ must follows the calculation of term $p(\mathbf{x}_i^{(0)})$ for every node $i$, and so forth. The messages from past to present are not calculated, since the location of nodes changes through the passing of time. In figure 4.3 is illustrated the message flow which is illustrated by arrows from past to present (downward) direction. On the other hand, regarding spatial constraints, the message transmission among neighboring nodes does not depend on the receiver, due to the single directional message flow (see figure 4.4). Therefore, every node which has available information to neighboring nodes broadcasts its information.

The broadcasted messages for every node $i$ to its neighboring nodes $j$ ($j \in \mathcal{S}_{i\rightarrow}$) are denoted $b^{(l)}_{X_i^{(t)}}(\cdot)\ \forall i$, and express the probability density function (pdf) of node's $i$ location all over the network, where $l$ is iteration index and $t$ is time index.

---

**Algorithm 1** [12]

---

1: given the prior distributions of every node $i$, $p(\mathbf{x}_i^{(0)})$, $\forall i$

2: (loop$_1$)*for* $t = 1$ to $T$

3: (loop$_2$)$\forall$ node $i = 1$ to $N$ **in parallel**

4: prediction operation (local information)

$$\mu_{h_i^{(t-1)}\rightarrow X_i^{(t)}}\left(\mathbf{x}_i^{(t)}\right) \propto \int_{\mathbf{x}_i^{(t-1)}} h_i^{(t-1)}\left(\mathbf{x}_i^{(t-1)}, \mathbf{x}_i^{(t)}\right) \mu_{X_i^{(t-1)}\rightarrow h_i^{(t-1)}}\left(\mathbf{x}_i^{(t-1)}\right) \mathrm{d}\mathbf{x}_i^{(t-1)}$$
$$= \int_{\mathbf{x}_i^{(t-1)}} p\left(\mathbf{x}_i^{(t)} | \mathbf{x}_i^{(t)}\right) p\left(\mathbf{z}_{i,\mathrm{self}}^{(t)} | \mathbf{x}_i^{(t-1)}, \mathbf{x}_i^{(t)}\right) \mu_{X_i^{(t-1)}\rightarrow h_i^{(t-1)}}\left(\mathbf{x}_i^{(t-1)}\right) \mathrm{d}\mathbf{x}_i^{(t-1)}$$

5: end(loop$_2$)

6: Correction Operation Algorithm (Algorithm 2)

7: end(loop$_1$)

---

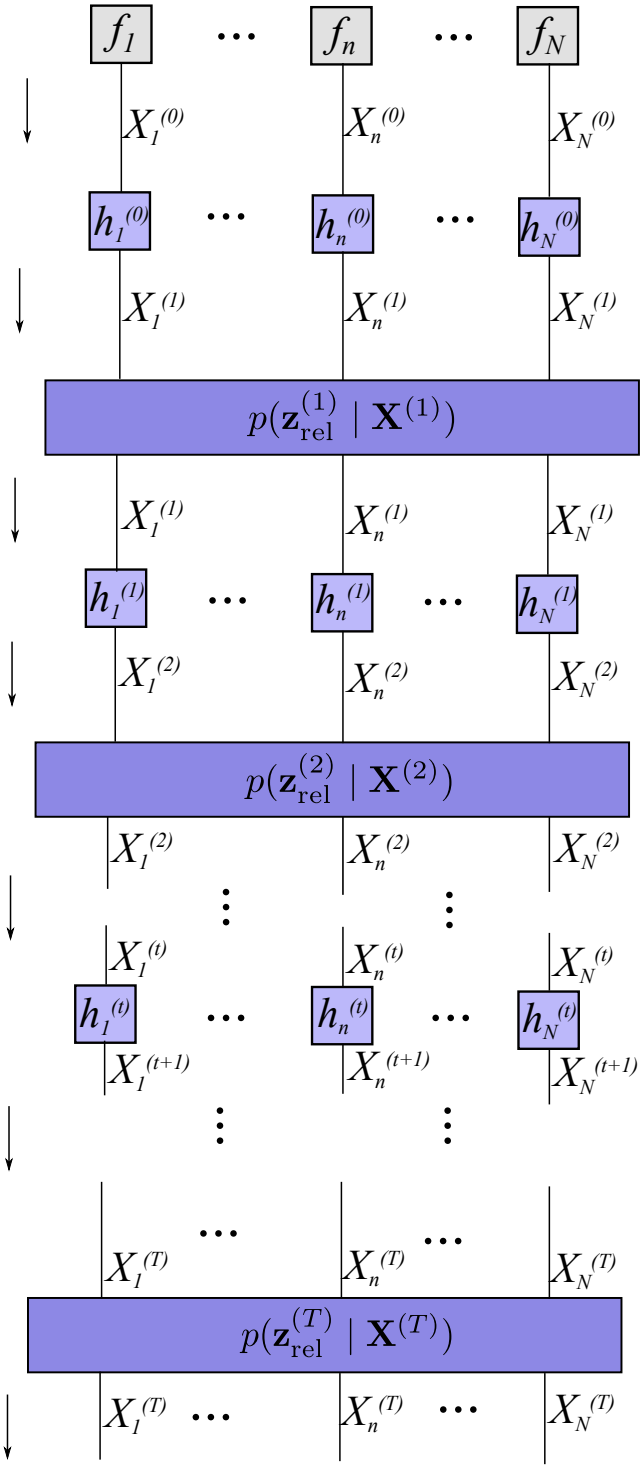



Figure 4.3: This figure illustrates the FG corresponding to the expression 4.22, that is a network with $N$ nodes from time 0 until time $T$. The message flow is from past to present, hence the direction of arrows is downward.

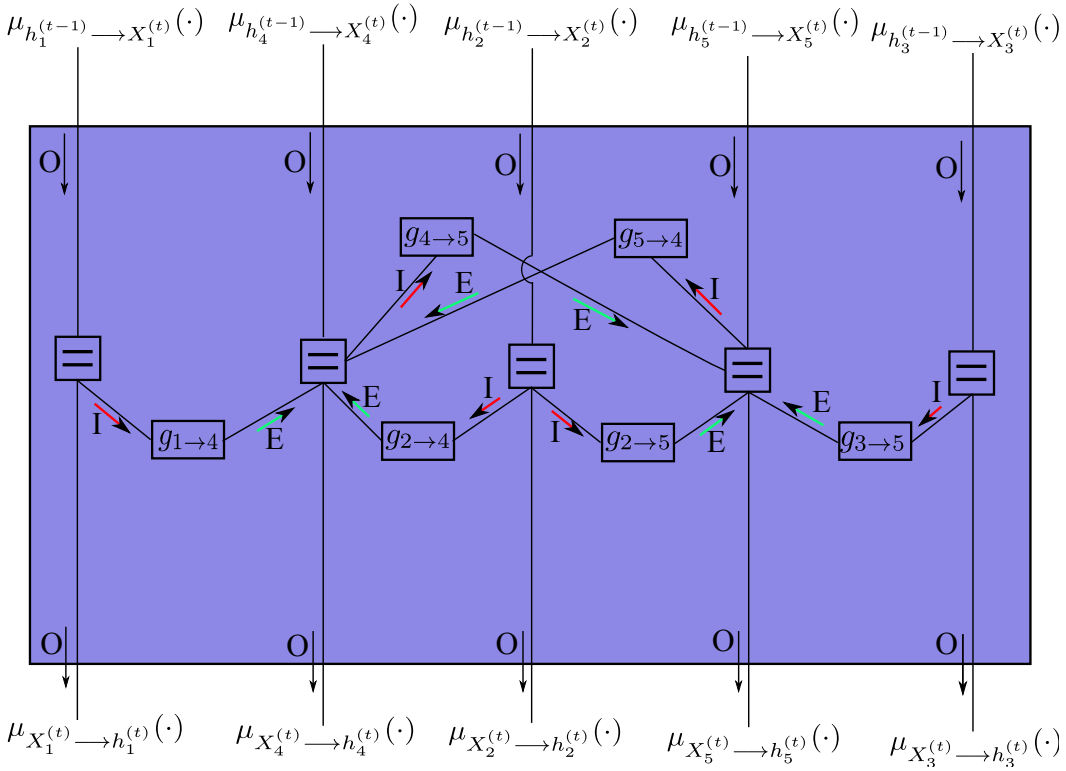



Figure 4.4: The factor graph of the factorization of factor $p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)})$ for the network of example 4.1. The I-red arrows correspond to internode messages, while the E-green arrows correspond to intranode messages at equality vertices. Finally the O-black arrows correspond to incoming and outgoing messages of factor $p(\mathbf{z}_{\text{rel}}^{(t)} \mid \mathbf{x}^{(t)})$.

---

**Algorithm 2** [12]

---

1: *for* every node $i = 1$ to $N$ **in parallel**
2: initialize *belief* $b_{X_i^{(t)}}^{(0)}(\cdot) = \mu_{h_i^{(t-1)} \to X_i^{(t)}}(\cdot)$
3: end
4: (loop$_1$)*for* $l = 1$ to $N_{iter}$ **in parallel**
5: (loop$_2$)$\forall$ node $i = 1$ to $N$ **in parallel**
6: broadcast $b_{X_i^{(t)}}^{(l-1)}(\cdot)$
7: receive $b_{X_i^{(t)}}^{(l-1)}(\cdot)$ from neighbors $\in \mathcal{S}_{j\to i}^{(t)}$
8: convert $b_{X_i^{(t)}}^{(l-1)}(\cdot)$ to a distribution with respect to variable $\mathbf{X}_i^{(t)}$, using integration over $\mathbf{X}_j^{(t)}$ (sum -

product algorithm update schedule), i.e.

$$\mu^{(l)}_{g_{j\to i}\to X_i^{(t)}}\left(\mathbf{x}_i^{(t)}\right) \propto \int_{\mathbf{x}_j^{(t)}} p\left(z_{j\to i}^{(t)}|\mathbf{x}_i^{(t)},\mathbf{x}_j^{(t)}\right) b^{(l-1)}_{X_j^{(t)}}\left(\mathbf{x}_j^{(t)}\right) \mathrm{d}\mathbf{x}_j^{(t)}$$

where $b^{(l-1)}_{X_j^{(t)}}\left(\mathbf{x}_j^{(t)}\right)$ is the belief of node $j$ at the just previous iteration and $p\left(z_{j\to i}^{(t)}|\mathbf{x}_i^{(t)},\mathbf{x}_j^{(t)}\right)$ is the likelihood of the relative measurements of node $j$ to node $i$.
9: Update the beliefs based on the measurements and the incoming messages

$$b^{(l)}_{X_i^{(t)}}\left(\mathbf{x}_i^{(t)}\right) \propto \mu_{h_i^{(t-1)}\to X_i^{(t)}}\left(\mathbf{x}_i^{(t)}\right) \prod_{j\in\mathcal{S}^{(t)}_{j\to i}} \mu^{(l)}_{g_{j\to i}\to X_i^{(t)}}\left(\mathbf{x}_i^{(t)}\right)$$

10: end (loop$_2$)

11: end (loop$_1$)

12: (loop$_3$)$\forall$ node $i = 1$ to $N$ **in parallel**

13: update the outgoing messages based on new beliefs, namely $\mu_{X_i^{(t)}\to h_i^{(t)}}(\cdot) = b^{(N_{iter})}_{X_i^{(t)}}(\cdot)$

14: end (loop$_3$)

---

The idea of SPAWN algorithm is illustrated in Algorithm 1, while the correction operation during the last step of SPAWN algorithm is given in Algorithm 2. The broadcast message of every node $i$, $b^{(l)}_{X_i^{(t)}}(\cdot)$, is also known as *belief* of node $i$. The estimated location of every node $i$ at time $t$ results by computing the mean squared error estimate [30] of its belief $b^{(N_{iter})}_{X_i^{(t)}}(\cdot)$ after the termination of correction operation (Algorithm 2). Namely, the estimated location of node $i$ at time $t$ is given by

$$\hat{\mathbf{x}}_i^{(t)} = \int_{\mathbf{x}_i^{(t)}} \mathbf{x}_i^{(t)}\ b^{(N_{iter})}_{X_i^{(t)}}\left(\mathbf{x}_i^{(t)}\right) \mathrm{d}\mathbf{x}_i^{(t)} \tag{4.23}$$

**Example 4.3.** Consider a $50 \times 50$ network with the 5 nodes of figure 4.5(a). Nodes 1 to 3 are anchors, while nodes 4 and 5 are agents. Distance measurements have the following form:

$$z_{j\to i}^{(t)} = \left|\left|\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}\right|\right|_2 + w_{ji},\ \forall j \in \mathcal{S}^{(t)}_{\to i}\ ,$$

where $w_{ji}$ is zero mean, Gaussian random variable with variance $0.4^2$.
Agent node 4 can receive distance measurements from anchor nodes 1 and 2, while agent node 5 can receive distance measurements from anchor nodes 2 and 3. Agents 4 and 5 can

communicate among them, and hence, it can occur cooperation among them. Let $t$ denote any arbitrary time of SPAWN algorithm.

Initially, before the first iteration ($l = 0$) of correction phase the beliefs of anchors 1,2 and 3 are Dirac delta functions since they know exactly their locations. Specifically,

$$
\begin{aligned}
b^{(0)}_{X_1^{(t)}}(\mathbf{x}_1^{(t)}) &= \delta(\mathbf{x}_1^{(t)} - \mathbf{x}_1), \\
b^{(0)}_{X_2^{(t)}}(\mathbf{x}_2^{(t)}) &= \delta(\mathbf{x}_2^{(t)} - \mathbf{x}_2), \\
b^{(0)}_{X_3^{(t)}}(\mathbf{x}_3^{(t)}) &= \delta(\mathbf{x}_3^{(t)} - \mathbf{x}_3),
\end{aligned}
$$

where $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are the known constant locations of anchors 1,2 and 3, respectively, e.g. $\mathbf{x}_1 = [30 \;\; 40]^\top$. Suppose that agent nodes 4 and 5 have no prior information about their positions, consequently their beliefs are uniform distributions all over the network map, namely,

$$
\begin{aligned}
b^{(0)}_{X_4^{(t)}}(\mathbf{x}_4^{(t)}) &= \frac{1}{50^2}, \quad 0 \leq x_{4,1}^{(t)}, x_{4,2}^{(t)} \leq 50, \\
b^{(0)}_{X_5^{(t)}}(\mathbf{x}_5^{(t)}) &= \frac{1}{50^2}, \quad 0 \leq x_{5,1}^{(t)}, x_{5,2}^{(t)} \leq 50,
\end{aligned}
$$

with $\mathbf{x}_i^{(t)} = [x_{i,1}^{(t)} \;\; x_{i,2}^{(t)}]^\top, \; \forall i = 1, ..., 5$.

- 1st iteration(l=1):
  At the first iteration, the network nodes which have knowledge about their position broadcast information to their neighboring nodes. Anchors 1,2 and 3 broadcast their beliefs as well as the distance measurements to their corresponding neighboring agent nodes. Agents 4 and 5 remain idle during iteration 1 since the have no information about their location. Agent 4 receives the beliefs of anchors 1 and 2 as well as their distance measurements, since $\mathcal{S}^{(1)}_{\to 4} = \{1, 2\}$. Therefore, agent 4 is able to compute the messages $\mu^{(1)}_{g_{1\to 4}\to X_4^{(t)}}(\cdot)$ and $\mu^{(1)}_{g_{2\to 4}\to X_4^{(t)}}(\cdot)$ according to line 8 in Algorithm 2, i.e.

$$
\begin{aligned}
\mu^{(1)}_{g_{1\to 4}\to X_4^{(t)}}(\mathbf{x}_4^{(t)}) &= \frac{1}{D}\int_{\mathbf{x}_1^{(t)}} p\left(z^{(t)}_{1\to 4}|\mathbf{x}_1^{(t)}, \mathbf{x}_4^{(t)}\right) \; b^{(0)}_{X_1^{(t)}}\left(\mathbf{x}_1^{(t)}\right) \mathrm{d}\mathbf{x}_1^{(t)}, \\
\mu^{(1)}_{g_{2\to 4}\to X_4^{(t)}}(\mathbf{x}_4^{(t)}) &= \frac{1}{D}\int_{\mathbf{x}_2^{(t)}} p\left(z^{(t)}_{2\to 4}|\mathbf{x}_2^{(t)}, \mathbf{x}_4^{(t)}\right) \; b^{(0)}_{X_2^{(t)}}\left(\mathbf{x}_2^{(t)}\right) \mathrm{d}\mathbf{x}_2^{(t)}
\end{aligned}
$$

  where $D$ is a normalization constant. Similarly, agent node 5 calculates the mes-

sages $\mu^{(1)}_{g_{2\to5}\to X_5^{(t)}}(\cdot)$ and $\mu^{(1)}_{g_{3\to5}\to X_5^{(t)}}(\cdot)$ utilizing line 8 in Algorithm 2, having received information from its neighboring nodes 2 and 3 ($\mathcal{S}^{(1)}_{\to5} = \{2,3\}$). The messages $\mu^{(1)}_{g_{1\to4}\to X_4^{(t)}}(\cdot)$ and $\mu^{(1)}_{g_{2\to4}\to X_4^{(t)}}(\cdot)$ are depicted in figure 4.5(b), whereas the messages $\mu^{(1)}_{g_{2\to5}\to X_5^{(t)}}(\cdot)$ and $\mu^{(1)}_{g_{3\to5}\to X_5^{(t)}}(\cdot)$ are depicted in figure 4.5(c). Note that these messages have grommet-shaped distributions, with center the location of anchor and radius the distance measurements. For example in figure 4.5(b) the distribution of the message $\mu^{(1)}_{g_{1\to4}\to X_4^{(t)}}(\cdot)$ (upper cycle) has circular shape with center the location of anchor 1 and radius the distance measurement $z^{(t)}_{4\to1}$. Finally the belief of every agent is updated according to line 9 in Algorithm 2, namely,

$$b^{(1)}_{X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) = \frac{1}{D} \times \mu^{(1)}_{g_{1\to4}\to X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) \times \mu^{(1)}_{g_{2\to4}\to X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) \times b^{(0)}_{X_4^{(t)}}(\mathbf{x}_4^{(t)}),$$
$$b^{(1)}_{X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) = \frac{1}{D} \times \mu^{(1)}_{g_{2\to5}\to X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) \times \mu^{(1)}_{g_{3\to5}\to X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) \times b^{(0)}_{X_5^{(t)}}(\mathbf{x}_5^{(t)}),$$

where we used the fact that $\mu_{h_i^{(t-1)}\to X_i^{(t)}}(\cdot) = b^{(0)}_{X_i^{(t)}}(\cdot),\ \forall i = 1,...,5$, from line 2 in Algorithm 2. The updated beliefs of agents 4 and 5 are illustrated in figure 4.5(d). Note that the updated beliefs of agents 4 and 5 are bimodal distributions. Observe that the estimation of the agent locations has ambiguity, therefore 1 more iteration of the correction operation needed.

- 2nd iteration(l=2):
  At the second iteration of the correction operation agents 4 and 5 broadcast their beliefs to their neighboring nodes since they have some information about their positions. Agent node 4 can receive messages from anchors 1 and 2 as well as from agent 5 (since $\mathcal{S}^{(2)}_{\to4} = \{1,2,5\}$). Namely, agent 4 computes the messages $\mu^{(2)}_{g_{1\to4}\to X_4^{(t)}}(\cdot)$, $\mu^{(2)}_{g_{2\to4}\to X_4^{(t)}}(\cdot)$ and $\mu^{(2)}_{g_{5\to4}\to X_4^{(t)}}(\cdot)$. The messages from anchors do not change through the passing of iterations since the distance measurements do not change, due to their belief (Dirac delta function). Therefore $\mu^{(2)}_{g_{1\to4}\to X_4^{(t)}}(\cdot) = \mu^{(1)}_{g_{1\to4}\to X_4^{(t)}}(\cdot)$ and $\mu^{(2)}_{g_{2\to4}\to X_4^{(t)}}(\cdot) = \mu^{(1)}_{g_{2\to4}\to X_4^{(t)}}(\cdot)$. The message from agent 5 to agent 4, $\mu^{(2)}_{g_{5\to4}\to X_4^{(t)}}(\cdot)$, is depicted in figure 4.5(e). The calculation of the latter message follows according to line 8 in Algorithm 2. Observe that due to the uncertainty of location of agent 5 the message to agent 4 has double grommet shaped distribution. With similar way, agent 5 receives messages from its neighbors 2, 3 and 4 (since $\mathcal{S}^{(2)}_{\to5} = \{2,3,4\}$) and calculates the messages $\mu^{(2)}_{g_{2\to5}\to X_5^{(t)}}(\cdot)$, $\mu^{(2)}_{g_{3\to5}\to X_5^{(t)}}(\cdot)$ and $\mu^{(2)}_{g_{4\to5}\to X_5^{(t)}}(\cdot)$ with similar way as

described above. The updated beliefs of agents 4 and 5 are calculated according to line 9 in Algorithm 2 as follows:

$$b^{(2)}_{X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) = \frac{1}{D} \times \mu^{(2)}_{g_{1\to 4}\to X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) \times \mu^{(2)}_{g_{2\to 4}\to X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) \times \mu^{(2)}_{g_{5\to 4}\to X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) \times b^{(0)}_{X_4^{(t)}}(\mathbf{x}_4^{(t)}),$$
$$b^{(2)}_{X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) = \frac{1}{D} \times \mu^{(2)}_{g_{2\to 5}\to X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) \times \mu^{(2)}_{g_{3\to 5}\to X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) \times \mu^{(2)}_{g_{4\to 5}\to X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) \times b^{(0)}_{X_5^{(t)}}(\mathbf{x}_5^{(t)}).$$

The beliefs of network nodes after the 2nd iteration are illustrated in figure 4.5(f).

If we apply mean squared error estimation [30] we take:

$$\hat{\mathbf{x}}_4^{(t)} = \int_{\mathbf{x}_4^{(t)}} \mathbf{x}_4^{(t)}\; b^{(2)}_{X_4^{(t)}}\left(\mathbf{x}_4^{(t)}\right) \mathrm{d}\mathbf{x}_4^{(t)},$$
$$\hat{\mathbf{x}}_5^{(t)} = \int_{\mathbf{x}_5^{(t)}} \mathbf{x}_5^{(t)}\; b^{(2)}_{X_5^{(t)}}\left(\mathbf{x}_5^{(t)}\right) \mathrm{d}\mathbf{x}_5^{(t)}.$$

Notice that the structure of correction operation of SPAWN algorithm operates in a tree fashion. At the first iteration, the nodes connected with anchors have partial or full information about their location, while at the second iteration agents which neighboring with anchor's neighbors will have useful information about their location, and so forth. ■

Finally, we state some remarks about SPAWN algorithm.

- The distributed operation of SPAWN relies on the fact that during correction operation every node uses only the local available information from its neighboring nodes in order to estimate its position (lines 8 and 9 in Algorithm 2). Regarding prediction operation, factor $h_i^{(\cdot)}(\cdot)$ contains only local information of every node $i$ (line 4 in Algorithm 1).

- SPAWN is sequential estimation problem within the FG framework, since its idea is to factorize the posterior distribution $p(\mathbf{x}^{(0:T)} \mid \mathbf{z}^{(1:T)})$ in order to create the corresponding FG of that factorization and finally to apply SPA. Furthermore, SPAWN consists of prediction operation and correction operation, showing that it is a sequential estimation problem.

Finally we will not proceed to the analysis about mobility model and UWB ranging models since they are beyond the scope of this thesis. All issues regarding SPAWN algorithm, which omitted can be found in more detail in [12].

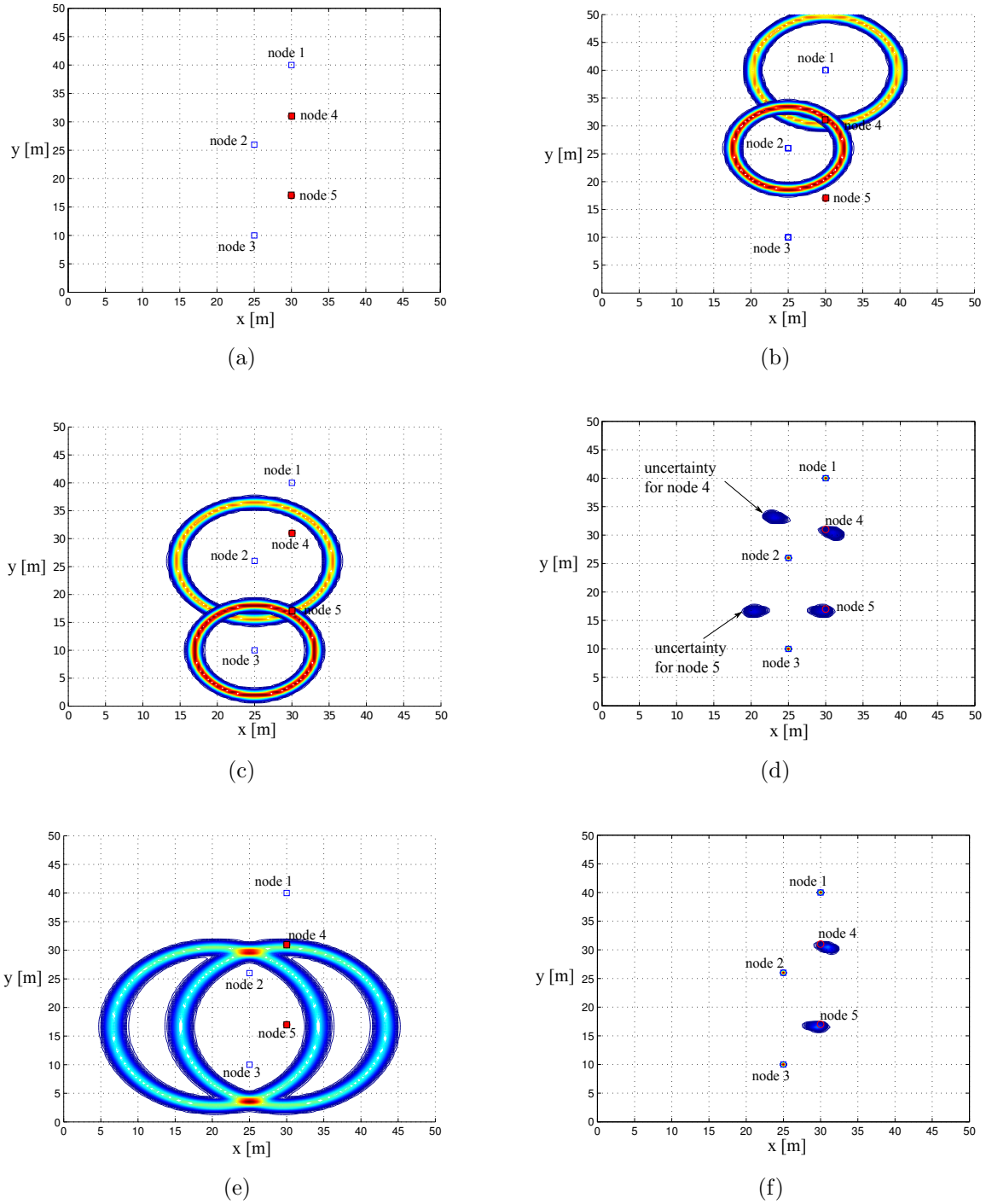



Figure 4.5: An example of a five-node network where the true locations of the 3 anchor nodes are $\mathbf{x}_1 = [30 \quad 40]^\top$, $\mathbf{x}_2 = [25 \quad 26]^\top$, $\mathbf{x}_3 = [25 \quad 10]^\top$, respectively, while the true locations of the 2 agent nodes are $\mathbf{x}_4 = [30 \quad 31]^\top$, $\mathbf{x}_5 = [30 \quad 17]^\top$, respectively. We apply two iterations of correction operation of SPAWN algorithm until the convergence of the agent's beliefs.

### 4.2.4 Experimental results

In this subsection we provide and discuss some numerical results. We consider a 2D $50 \times 50$ $m^2$ network where the coordinates of nodes $\mathbf{x} = [x \;\; y]^\top$ and the connectivity are given in figure 4.6. As we can see the network consist of 3 anchors and 2 agents. Although, there is uncertainty in localization since agents can exchange information only with 2 anchor nodes, therefore there exist ambiguity between two possible locations, thus, additional information is required. However, each agent is also connected to another agent, therefore cooperation among the two provides that necessary additional information. Cramer-Rao bound(CRB) was calculated for the 2D topology according to [13], showing the efficiency of SPAWN algorithm.[4] Moreover SPAWN algorithm utilized for a grid resolution of $\delta = 0.5m$. Figure 4.7 illustrates the performance in terms of MSE calculation across all agents as a function of noise variance for 2D localization.

The bandwidth metric as a function of MSE is depicted in figure 4.8. Specifically, figure 4.8 shows the size of the messages exchanged among all nodes (anchors and agents) and the MSE corresponding to the given size. The size of the messages is expressed as the total real numbers exchanged in the network, namely, if the broadcasted message at iteration $l$ from agent $i$ requires $m$ real numbers, then the total number will be increased cumulatively by $m$. Figure 4.8 depicts the total real numbers exchanged during SPAWN algorithm until its termination, for the topology of figure 4.6. As we can see SPAWN algorithm requires in the order of $80K$ real numbers. For larger ranging errors in noise variance, SPAWN has poor performance in terms of MSE. Finally, the Cramer-Rao bound (CRB) is depicted in figure 4.9, i.e. the MSE lower bound achieved by an unbiased estimator of the unknown coordinates across all agents. Notice that CRB curve follows the same tendency with SPAWN curve.

[4] In order to compare SPAWN algorithm with CRB calculation, we consider symmetric ranging measurements, i.e. $z_{i\to j} = z_{j\to i}$ (as in [13]).

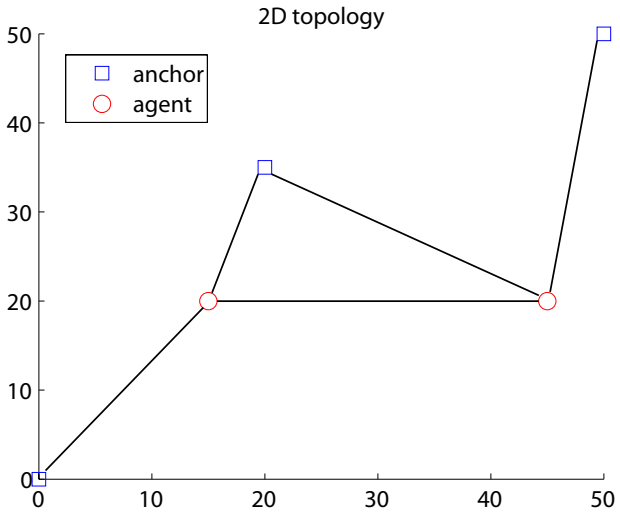



Figure 4.6: Test 2D topology and the corresponding node connectivity: three anchors are placed at $[0 \;\; 0]^\top$ ; $[20 \;\; 35]^\top$ ; $[50 \;\; 50]^\top$ and two agents at $[15 \;\; 20]^\top$ ; $[45 \;\; 20]^\top$ , respectively. Notice that each agent can communicate with only two anchors and another agent.

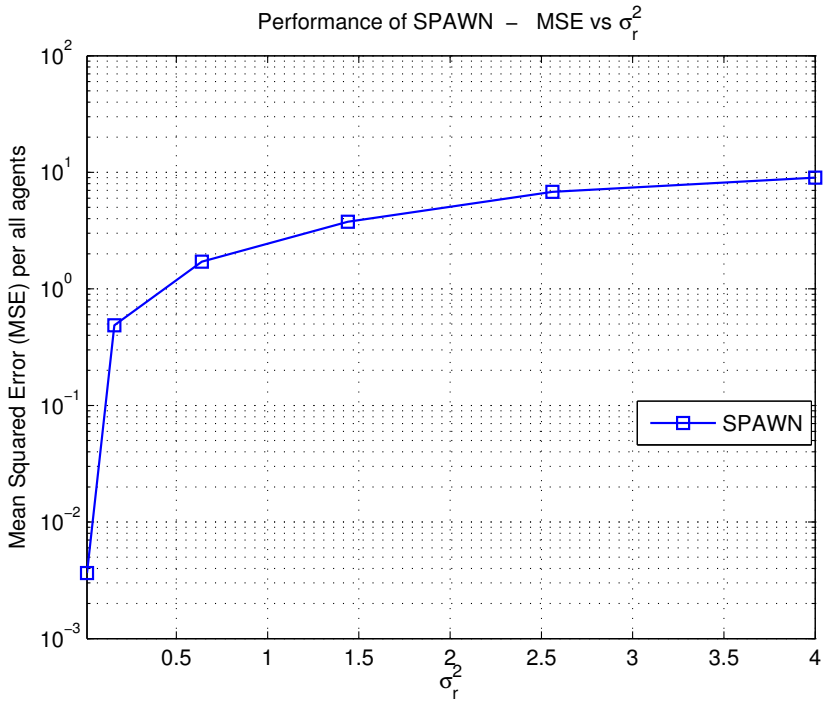



Figure 4.7: Mean squared error (MSE) as a function of ranging noise variance $\sigma_r^2$ for 2D localization.

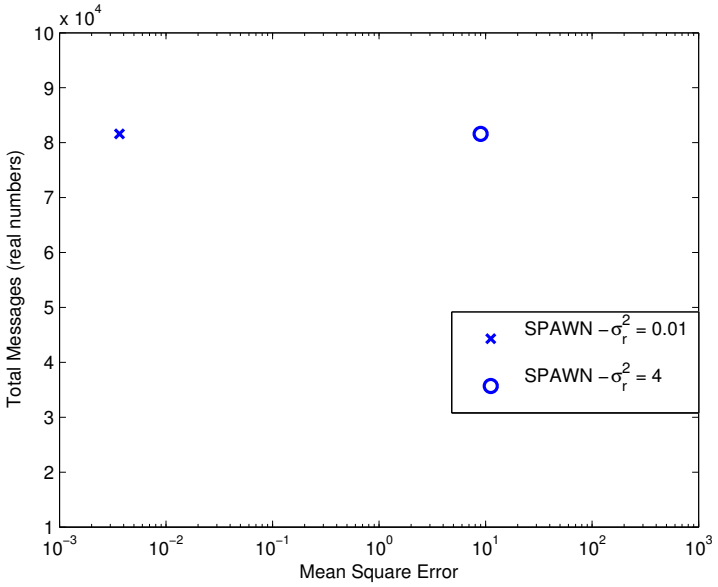



Figure 4.8: Total size of exchanged messages for small and large ranging error noise variance for SPAWN algorithm.

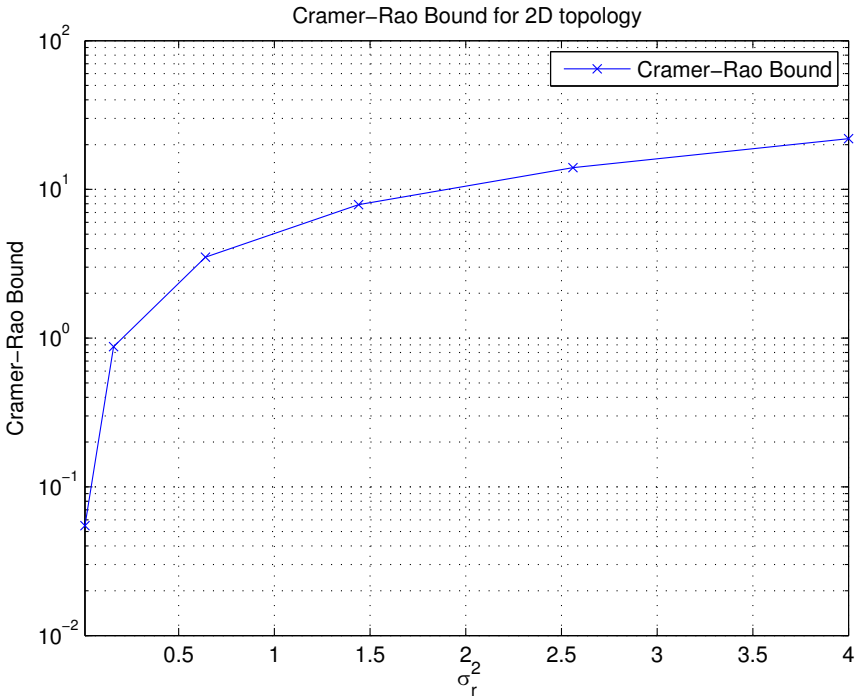



Figure 4.9: Cramer-Rao lower bound (CRB) for the given topology with the given connectivity. The computation of CRB follows according to [13].

# Chapter 5

# Factor Graph Applications: Time Scheduling in Multi-hop Networks

The content of this chapter is based on the work of Jung-Chieh Chen *et al.* [18]. This chapter examines time scheduling in a packet radio network (PRN), based on a factor graph approach. The goal is to design a distributed algorithm with low complexity and as small as possible scheduling delay, in order to reduce the end-to-end communication. The main idea is to express the PRN in terms of factor graphs (FGs) and to apply the sum-product algorithm (SPA), i.e. passing soft information through the nodes via messages at the FG. We will see that the FG corresponding to a PRN exploits local information passing among the neighboring network nodes of the PRN. The algorithm will be executed in iterative fashion, since the FG corresponding to a PRN is not cycle-free.

## 5.1 Introduction

A *packet radio network* (PRN) is a network in which the *stations* that constitute it, are scattered over the space and exchange information via packets wirelessly sharing common frequency channel. Every station of the network can receive and transmit packets, i.e. it is a *transceiver*. The problem in PRNs is to find a fair, reliable and effective *time scheduling*, that offers as small as possible end-to-end delays during transmission. It has been shown that finding such schedule is a NP-hard problem [20].

When two stations of the PRN are within communication range, we say that they are *one-hop neighbors*. Stations which are not within communication range, but they have one or more common one-hop stations, they are *two-hop neighbors*. Any station can receive/transmit packets only from/to one-hop neighbors. If two one-hop stations transmit simultaneously then they suffer from *collision*. *Time frame* is the period of time during which transmission occurs; it is repeated continuously. Moreover, the utilization of a single frequency channel requires mechanisms that avoid collisions. One popular time scheduling is the time-division multiple-access (TDMA) protocol, in which all stations of the PRN are

synchronized and the time is divided to a frame of time slots, during which one or more stations transmit their packet. TDMA scheduling is designed, so that one-hop neighboring stations won't transmit simultaneously. Additionally a station won't transmit and receive a packet at the same time slot. It is noted that the TDMA schedule usually is not optimal in terms of channel utilization. Finally, the synchronization process in TDMA causes overhead.

The algorithm proposed must encounter the following problems:

- In PRNs there exist two possible cases of collisions. The first one occurs when two or more one-hop neighboring stations transmit their packet simultaneously. The second one occurs when two or more two-hop neighboring stations transmit their packet to their common adjacent neighbor station simultaneously, and thus, the latter receives 2 packets from neighboring stations, hence collision.
- Every station of the PRN must transmit at least one packet during the time frame.
- Channel utilization must be maximized.
- Time frame length must be minimized, in order to reduce the end-to-end communication delays.

## 5.2 System model

We consider time frame of $M$ equal-length time slots. Given a PRN consisting of $N$ stations, we create an undirected graph $G(V, E)$. The set of nodes $V$ represents the set of $N$ stations, while the set of edges $E$ stands for the links among one-hop stations. Namely, if a node $i$ and a node $j$ have edge among them, then the corresponding stations ($i$ and $j$) can exchange packets. We define a $N \times N$ symmetric binary matrix, named connectivity matrix and denoted by $\mathbf{F}$, where its $(i, j)$-th element ($[\mathbf{F}]_{i,j} = f_{i,j}$) is given by

$$[\mathbf{F}]_{i,j} = \begin{cases} 1, & \text{if stations } i,\ j \text{ are either one-hop-apart or two-hop-apart,} \\ 0, & \text{otherwise,} \end{cases} \tag{5.1}$$

and $[\mathbf{F}]_{i,i} = 0,\ i = 1, ..., N$, by convention. Essentially equation 5.1 denotes that when $[\mathbf{F}]_{i,j} = 0$ the stations $i$ and $j$ can transmit simultaneously without collision.

When we design a schedule for a PRN, we create a pattern which indicates the time slot in the time frame every station transmits. This pattern is repeated for every time frame.

We define a $M \times N$ binary matrix $\mathbf{S}$ for the scheduling pattern; its $(m, j)$-th element ($[\mathbf{S}]_{m,j} = s_{m,j}$) is given by

$$[\mathbf{S}]_{m,j} = \begin{cases} 1, & \text{if station } j \text{ can transmit a packet in time slot } m, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Given matrix $\mathbf{S}$, we calculate the performance of scheduling pattern in terms of the throughput of the PRN,

$$\tau = \sum_{m=1}^{M} \sum_{j=1}^{N} s_{m,j} \ , \tag{5.3}$$

which indicates how many time slots are utilized from PRN stations in the whole time frame for transmission. Furthermore, the channel utilization for station $j$ is given by

$$\rho_j = \frac{\sum_{m=1}^{M} s_{m,j}}{M} \ , \tag{5.4}$$

while the channel utilization for the entire PRN is given by

$$\rho = \frac{\sum_{m=1}^{M} \sum_{j=1}^{N} s_{m,j}}{M \times N} = \frac{\tau}{M \times N} \ , \tag{5.5}$$

which expresses the average slot utility per frame, with $0 < \rho < 1$.

The next step is to express the constraints in terms of the notation above, namely the optimal schedule $\mathbf{S}_{\mathrm{OPT}}$ must satisfies:

- Avoid simultaneous packet transmissions of any station and all its one-hop or two-hop away neighbors:

$$\sum_{m=1}^{M} \sum_{j=1}^{N} \sum_{i=1}^{N} f_{i,j} s_{m,j} s_{m,i} = 0. \tag{5.6}$$

- Every station must transmit at least one packet per frame, i.e.

$$\sum_{m=1}^{M} s_{m,j} > 0, \ \text{for } j = 1, ..., N. \tag{5.7}$$

- Maximization of the channel utilization $\rho$.

- Minimization of the frame length $M$.

Usually the minimum frame length depends on the topology of the PRN. Throughout, we consider that $M$ takes the lower bound (the bound is derived in [20] and [33] for any PRN scenario), while we optimize the 3rd condition (maximize channel utilization).

## 5.3 Sum-product algorithm for packet radio networks

The main idea of the algorithm is to construct the FG which describes a PRN topology, taking into account the connectivity constraints. In sequel, is applied SPA on the FG trying to maximize the channel utilization. The first remark is that the algorithm must executed in iterative fashion, since the FG has cycles. Therefore, FGs constitute the solution to that problem, since every station of the PRN (node of the graph) can locally exchange information with its neighbors, in order to determine whether it could send packet in the current slot or not.

The FG of the proposed algorithm consists of 2 types of nodes, the variable nodes and the factor nodes. The variable nodes are labeled $\{s_{m,i}\}$ and the variables associated with them (have the same label) are binary variables, taking value 1, if node $i$ can transmit at time slot $m$, and 0, if it cannot ($1 \leq i \leq N,\ 1 \leq m \leq M$). There exist 2 kinds of factor nodes denoted $\{I_{m,i}\}$ and $\{T_i\}$. The first expresses the constraint of equation 5.6, i.e. prevents the simultaneous transmission of station $i$ and all its one-hop or two-hop neighbors at time slot $m$. The second one expresses the constraint of equation 5.7, i.e. imposes station $i$ to transmit at least once during the whole time frame. Finally, a variable node $\{s_{m,i}\}$ is connected across an edge with a factor node $\{I_{m,j}\}$, if and only if, the station $i$ is involved in the $j$th constraint, i.e. if and only if stations $i$ and $j$ are one-hop or two-hop neighbors. The following example clarifies how to construct a factor graph from an arbitrary PRN.

**Example 5.1.** Consider a packer radio network (PRN) of 5 nodes depicted in figure 5.1. The corresponding FG of the PRN is illustrated in figure 5.2. The minimum frame length of that PRN is equal to the maximum degree plus 1, i.e. $M = 3$. $I_{m,i}$ factor imposes station $i$ not to transmit simultaneously with one- or two-hop neighbors, at any time slot $m$. $T_i$ factor imposes station $i$ to transmit at least once during the $M = 3$ time slots. The variable node $s_{m,i}$ corresponds to the station $i$ at time slot $m$ and it is associated with the

binary variable $s_{m,i}$ which indicates whether station $i$ transmits its packet at time slot $m$ ($1 \rightarrow$ yes, $0 \rightarrow$ no). ■


Figure 5.1: A simple PRN with 5 stations. Station 1 is one-hop-apart from station 2, while station 3 is two-hop-apart from station 1. Hence station 1, 2 and 3 cannot transmit simultaneously. With similar way we can examine which stations of the PRN cannot transmit simultaneously.

$I$ factors prevent the simultaneous transmissions of nodes with their one- or two-hop neighbors. Therefore:

$$I_{m,i}(\mathbf{s}_{m,i}) = \mathbb{I}\left\{\mathbf{s}_{m,i} \text{ is valid bit pattern for local factor } I_{m,i}\right\}, \tag{5.8}$$

where $\mathbb{I}\{\cdot\}$ denotes the indicator function of the expression inside the hooks, while $\mathbf{s}_{m,i}$ is a binary vector consisting of the $\{s_{m,i}\}$'s associated with the factor $I_{m,i}$. Notice that the size of vector $\mathbf{s}_{m,i}$ changes from factor to factor. As *valid local bit patterns* for factor $I_{m,i}$, we define the configurations of vector $\mathbf{s}_{m,i}$ that satisfy the local factor $I_{m,i}$, i.e. those which maximize the channel utilization for the PRNs as well as satisfy the constraints of factor $I_{m,i}$. To be specific, for the PRN of the example 5.1, regarding factor $I_{1,2}$, the valid local bit patterns are the configurations of the binary vector $\mathbf{s}_{1,2} = (s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4})$ equal to $(0,1,0,0)$, $(0,0,1,0)$ and $(1,0,0,1)$, i.e. the configurations where factor $I_{1,2}(s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4}) = 1$. Otherwise, it is equal to 0. Namely the factor $I_{1,2}$ indicates that we locally examine at the first time slot the network involving stations which are one- and two-hop away from station 2, i.e. the stations 1, 2, 3 and 4. That factor allows only station 1 and station 4 to transmit simultaneously since they are three-hop-away stations. Moreover stations 2 or 3 can transmit individually without any other one- or two-hop away neighboring station transmitting simultaneously, since in the subnetwork of stations 1, 2, 3 and 4, stations 2 and 3 have only one- or two-hop away neighboring stations. Notice that the patterns $(1,0,0,0)$ and $(0,0,0,1)$ are valid and satisfy the connectivity constraints, but they don't satisfy the optimality constraints since if we use one of them, we are not optimal in terms of channel utilization. For the rest of the $I$-factors the valid local bit patterns can be calculated similarly.

Regarding $T$ factor, as we stated above they impose a station to transmit at least once

during the time frame. Thus,

$$T_i(\mathbf{s}_i) = \mathbb{I}\left\{\mathbf{s}_i \text{ is valid bit pattern for local factor } T_i\right\}, \tag{5.9}$$

where $\mathbf{s}_i$ is a $M \times 1$ binary vector consisting of the $\{s_{m,i}\}$'s associated with the factor $T_i$. It can be easily seen that the valid local bit patterns are all binary vectors of size $M \times 1$ except the zero vector. For the example 5.1, regarding factor $T_1$, the valid local bit patterns are the configurations of binary vector $\mathbf{s}_1 = (s_{1,1}, s_{2,1}, s_{3,1})$ equal to $(1,1,1)$, $(1,0,1)$, $(0,1,1)$, $(1,1,0)$, $(0,1,0)$, $(0,0,1)$, $(1,0,0)$, that is, the configurations that make the factor $T_1(s_{1,1}, s_{2,1}, s_{3,1}) = 1$. All other valid bit patterns for the rest of $T$-factors can be calculated accordingly.

As explained, every variable $s_{m,i},\ \forall i = 1, ..., N,\ m = 1, ..., M$, is binary variable which indicates if station $i$ transmits a packet at the time slot $m$. In terms of factor graphs the messages between variable nodes and factor nodes are $2 \times 1$ vectors which carry the probability the variable node $s_{m,i}$ takes the values 1 or 0, respectively. It easily can be shown that the factor graph of a PRN has $M \times N$ variables nodes, $M \times N$ $I$-factor nodes and $N$ $T$-factor nodes (totally $M \times N + N$ factor nodes). We define the set of factors as $\mathcal{F} = \{I_{1,1}, I_{1,2}, ..., I_{2,1}, ..., I_{2,N}, ..., I_{M,1}, ..., I_{M,N}, T_1, T_2, ..., T_N\} = \{f_1, f_2, ...., f_{|\mathcal{F}|}\}$, with $|\mathcal{F}| = M \times N + N$ and $I_{m,i} = f_{(m-1)N+i}$. Moreover, we define the set of variables as $\mathcal{X} = \{s_{1,1}, s_{1,2}, ..., s_{M,1}, ..., s_{M,N}\} = \{x_1, x_2, ...., x_{|\mathcal{X}|}\}$, with $|\mathcal{X}| = M \times N$ and $s_{m,i} = x_{(m-1)N+i}$. $\mathcal{N}(x_i)$ denotes the set of factor nodes adjacent to the variable node $x_i$. The set of variable nodes which are adjacent with factor node $f_j$ is denoted $\mathcal{N}(f_j)$. Finally, if we want to denote a set $\mathcal{K}$ except an element $k_i$ of the set $\mathcal{K}$, we write $\mathcal{K}\backslash\{k_i\}$.

The message from variable node $x_i$ to a neighboring factor node $f_j$ is denoted $\mu_{x_i \longrightarrow f_j}(x_i)$, which is a $2 \times 1$ vector indicating the probability of the variable $x_i$ taking the values 1 or 0. The message from factor node $f_j$ to a neighboring variable node $x_i$ is defined accordingly $(\mu_{f_j \longrightarrow x_i}(x_i))$.

Notice that the factor graph contains cycles, therefore the SPA must be executed iteratively until convergence. $N_{\text{iter}}$ denotes the number of total iterations and it is determined by the size of the PRN. While the number of stations that constitute PRN increases the total number of SPA required iterations increases too. The SPA for PRNs is described as follows:

1. Initially, we set a counter $count = 1$. For every variable node we associate an a priori

probability following uniform distribution over the range $(0, 1)$, namely

$$\pi_b(x_i) = p(x_i = b) \sim \mathcal{U}(0, 1), \quad i = 1, ..., M \times N, \quad b \in \{0, 1\}, \tag{5.10}$$

normalizing the probabilities in order to have $\pi_0(x_i)+\pi_1(x_i) = 1$, where $\pi_0(x_i)$ denotes the a priori probability of variable $x_i$ taking the value 0. Then for $i = 1, ..., M \times N$, and $j = 1, ..., (M \times N + N)$, we set

$$\mu_{x_i \longrightarrow f_j}(0) = \pi_0(x_i), \tag{5.11}$$

and

$$\mu_{x_i \longrightarrow f_j}(1) = \pi_1(x_i). \tag{5.12}$$

2. The second step is to calculate the messages (vectors) from factor nodes to variable nodes, following the idea of SPA, i.e. for $i = 1, ..., M \times N$, and $j = 1, ..., (M \times N + N)$, we take

$$\mu_{f_j \longrightarrow x_i}(x_i) = \frac{1}{D} \sum_{\sim\{x_i\}} f_j\left(\mathbf{x}_j\right) \prod_{x_l \in \mathcal{N}(f_j)\backslash\{x_i\}} \mu_{x_l \longrightarrow f_j}(x_l), \tag{5.13}$$

where $\mathbf{x}_j$ denotes the set of variables which are arguments in factor $f_j$, and recall that the symbol $\sim \{x_i\}$ under the sum denotes the sum over all variables which are arguments in factor $f_j$ except variable $x_i$. Finally, $D$ is a normalization constant, such that $\mu_{f_j \longrightarrow x_i}(0) + \mu_{f_j \longrightarrow x_i}(1) = 1$. Notice that the factors are not all the same, since the first $M \times N$ of them correspond to $I$-factors, while the rest of them correspond to $T$-factors. Since the aforementioned factors are indicator functions, we can rewrite the update rule of messages from factor nodes to variable nodes as (for $i = 1, ..., M \times N$ and $j = 1, ..., (M \times N + N)$):

$$\mu_{f_j \longrightarrow x_i}(x_i) = \frac{1}{D} \sum_{\mathbf{x}_j \in \mathcal{V}_j \backslash \{x_i\}} \prod_{x_l \in \mathcal{N}(f_j)\backslash\{x_i\}} \mu_{x_l \longrightarrow f_j}(x_l), \tag{5.14}$$

where $\mathcal{V}_j$ stands for the set of valid local bit patterns associated with local factor $f_j$.

3. In the third step we calculate the messages from variable nodes to factor nodes, which

for $i = 1, ..., M \times N$, and $j = 1, ..., (M \times N + N)$, are given by

$$\mu_{x_i \longrightarrow f_j}(x_i) = \frac{1}{D}\, \pi_b(x_i) \prod_{f_k \in \mathcal{N}(x_i) \setminus \{f_j\}} \mu_{f_k \longrightarrow x_i}(x_i), \tag{5.15}$$

where $D$ is a normalization constant.

4. At this step we calculate the marginals of all variable nodes, i.e. for $i = 1, ..., M \times N$, we take

$$g_{x_i}(x_i) = \pi_b(x_i) \prod_{f_k \in \mathcal{N}(x_i)} \mu_{f_k \longrightarrow x_i}(x_i), \tag{5.16}$$

5. Having all marginals $g_{x_i}(x_i),\ \forall x_i$, we apply hard decisions, in order to infer if the station corresponding to variable $x_i$ transmits or not. That is,

$$\hat{x}_i = \begin{cases} 1, & \text{if } \ g_{x_i}(1) > g_{x_i}(0) \\ 0, & \text{otherwise.} \end{cases} \tag{5.17}$$

We define the vector of estimated variables of size $M \times N$, denoted $\hat{\mathbf{x}} = [\hat{x}_1 \ \cdots \ \hat{x}_{MN}]$, as the vector includes $\hat{x}_i$'s of step 5, $i = 1, ..., MN$. Every local factor has different variable arguments of vector $\hat{\mathbf{x}}$, therefore the vector $\hat{\mathbf{x}}_j$ denotes the set of variables nodes $\hat{x}_i \in \hat{\mathbf{x}}$ which are adjacent to factor node $f_j$, i.e.

$$\hat{\mathbf{x}}_j = \{\hat{x}_i \in \hat{\mathbf{x}} : \hat{x}_i \in \mathcal{N}(f_j)\}\,. \tag{5.18}$$

If $\hat{\mathbf{x}} = [\hat{x}_1 \ \cdots \ \hat{x}_{MN}]$ satisfies all local factor constraints or the number of iteration is reached to $N_{\text{iter}}$ then the algorithm terminates, i.e.

$$\prod_{j=1}^{MN+N} f_j(\hat{\mathbf{x}}_j) = 1 \tag{5.19}$$

or $count = N_{iter}$ then *terminate*, otherwise the counter increases $count = count + 1$, goto step 2.

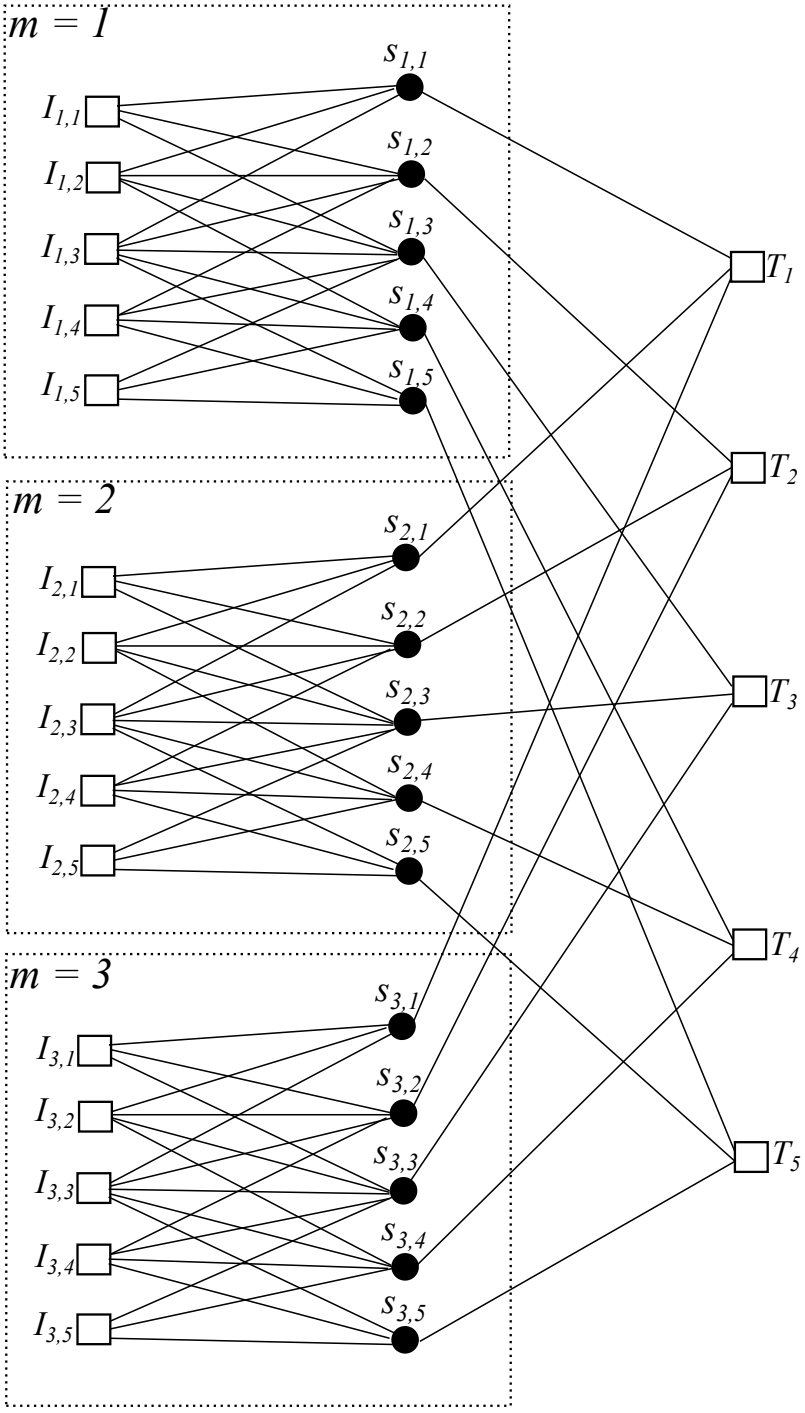



Figure 5.2: The FG corresponding to the PRN of the figure 5.1. The FG taking into account all the constraints of the PRN.

Noting the SPA for PRNs the following remarks are stated:

- The SPA for PRNs is distributed algorithm, since only local information passing

through the edges of FG.

- The presence of cycle leads to iterative algorithm. The schedule ignores the cycles and operates without accounting them (as in LDPC codes case).
- The soft information passing through the edges of the FG increases the reliability.

## 5.4 Performance example of time scheduling

We consider a network of the example 5.1, i.e. the network of figure 5.1. If we construct the corresponding FG of this PRN, applying the SPA for PRNs we take the time schedule of figure 5.3. Notice that this scheduling is consistent as well as near-optimal in the sense of channel utilization, since all nodes transmit at least once during time frame and as much as possible stations transmit simultaneously at any time slot without causing collision.

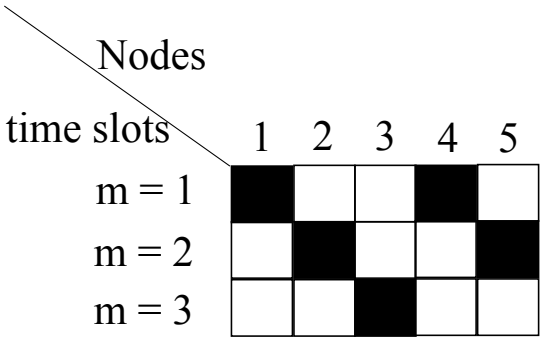



Figure 5.3: A near-optimal schedule after some iterations of SPA for PRNs.

# Chapter 6

# Conclusion

It was shown that factor graphs (FGs) are bipartite graphs which describe the factorization of a global function into a product of local functions. FGs can be applied in a wide range of fields such as signal processing, coding theory, detection and estimation, artificial intelligence and others ([3], [9], [24]).

Special emphasis was given to the sum-product algorithm (SPA), which can be applied in any FG stating some variations, such as max-product algorithm and the min-sum algorithm. SPA can also be applied when the FG has cycles, where in such cases an iterative algorithm is produced, with the corresponding message-passing schedule. The schedule usually depends on the nature of the problem.

The first presented application on FGs was the decoding of LDPC codes where the FG had cycles. The SPA-decoding algorithm was presented, ignoring the existence of cycles and executing it, iteratively. We note that there exists a performance-complexity trade-off, since the increased bit error rate performance implies larger computational complexity ([31], [27]).

For the case of convolutional codes the FG is cycle-free, hence the SPA-decoding algorithm is not iterative. The resulting FG is created according to a trellis diagram of the convolutional code. Convolutional decoding in terms of FGs follows the idea of forward-backward algorithm, but with a simpler approach. The update rules of the decoding algorithm follow the idea of update rules of SPA for cycle-free FGs ([24], [28]).

The last class of studied codes was Turbo codes. We examined the case of parallel concatenation of convolutional codes, in which we utilize 2 convolutional encoders/decoders in parallel and an interleaver among them. The SPA decoding algorithm is iterative since the FG of Turbo decoder has cycles. The performance of Turbo codes in terms of bit error rate is excellent and quite similar to the performance of LDPC codes. The performance-complexity trade-off also exists in the case of Turbo codes ([24], [28]).

In sequel, are studied Bayesian estimation problems and analyzed within the FG framework. More specifically, a cooperative localization algorithm was presented, based on Bayesian estimation following the idea of FGs. The idea of the algorithm is to construct a

network FG by mapping vertices of the FG onto the network topology. Since the FG has cycles a scheduling must be considered in order to satisfy the temporal/spatial constraints of the network. The resulting algorithm (SPAWN) is a distributed cooperative localization algorithm ([12]).

Finally we considered the application of FGs in the problem of time scheduling for multi-hop networks. The idea is to map a packet radio network (PRN) to a FG. The designed algorithm applies SPA onto the FG of the PRN in order to find a near-optimal time scheduling. The resulting time scheduling must take into account the consistency constraints (collisions) as well as the optimality constraints (minimization of frame length - maximization of channel utilization). The resulting FG has cycles, however the SPA is applied without regard to the cycles. The algorithm estimates a near-optimal time scheduling, given the constraints ([18]).

We conclude that FGs have great flexibility in modeling systems in a unified way. SPA is a very elegant and powerful tool to solve several inference problems, therefore it has application in many fields, including digital communications and networking.

# Bibliography


[1] R. G. Gallager, *Low-Density Parity-Check Codes.* Cambridge, MA: MIT Press, 1963.

[2] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 533–547, Sept. 1981.

[3] F. R. Kschischang, B. J. Frey, and H.-A. Loelinger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Information Theory*, vol. 47, pp. 498–519, Feb. 2001.

[4] N. Wiberg, H.-A. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecomm.*, vol. 6, pp. 513–525, Sept./Oct. 1995.

[5] V. Isham, "An introduction to spatial point processes and Markov random fields," *Int. Stat. Rev.,* vol. 49, pp. 2143, 1981

[6] F. V. Jensen, *An introduction to Bayesian networks.* New York: Springer-Verlag, 1996.

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, 2nd ed. San Francisco, CA: Kaufmann, 1988.

[8] G. D. Forney, "Codes on graphs: normal realizations," *IEEE Trans. Information Theory*, vol. 47, no. 1, pp. 520–545, February 2001.

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

[10] J. S. Yedidia,W. T. Freeman, and Y.Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory,* 51(7):22822312, July 2005.

[11] S. Aji and R. McEliece. "The generalized distributive law." *IEEE Transactions on Information Theory,* 46:325–353, March 2000.

[12] H. Wymeersch, J. Lien, and M. Z. Win, "Cooperative localization in wireless networks," *Proc. IEEE*, vol. 97, no. 2, pp. 427–450, Feb. 2009.

[13] N. Patwari, A. O. Hero III, M. Perkins, N. S. Correal, and R. J. ODea, "Relative location estimation in wireless sensor networks," *IEEE Trans. Signal Processing,* vol. 51, no. 8, pp. 2137–2148, Aug. 2003.

[14] P. Alevizos, N. Fasarakis-Hilliard and A. Bletsas, "Cooperative localization in wireless networks under bandwidth constraints", in 2012 Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, Nov. 2012.

[15] N. Fasarakis-Hilliard, P. Alevizos and A. Bletsas, "Cooperative localization in wireless networks under bandwidth constraints", *submitted*.

[16] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal,* vol. 27, pp. 379–423, 1948.

[17] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon limit error-correcting coding and decoding: turbo codes." In Proc. *IEEE International Conference on Communications (ICC),* pages 1064–1070, Geneva, Switzerland, May 1993

[18] Jung-Chieh Chen, Yeong-Cheng Wang, and Jiunn-Tsair Chen. "A novel broadcast scheduling strategy using factor graphs and the sum-product algorithm," *IEEE Transactions on Wireless Communications,* vol. 5, no. 6, pp. 1241–1249, June 2006.

[19] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, 1974.

[20] A. Ephremides and T. V. Truong, "Scheduling broadcast in multihop radio networks," *IEEE Trans. Commun.*, vol. 38, pp. 456–460, Apr. 1990

[21] M. Z. Win, Y. Shen, and H. Wymeersch, "On the position error bound in cooperative networks: A geometric approach," *in Proc. IEEE Int. Symp. Spread Spectr. Tech. Applicat.*, Bologna, Italy, Aug. 2008, pp. 637–643.

[22] Y. Shen, H. Wymeersch, and M. Z. Win, "Fundamental limits of wideband cooperative localization via Fisher information," *in Proc. IEEE Wireless Commun. Netw. Conf.*, Kowloon, Hong Kong, China, Mar. 2007, pp. 3951–3955.

[23] N. Patwari, J. N. Ash, S. Kyperountas, A. O. Hero, III, R. L. Moses, and N. S. Correal, "BLocating the nodes: Cooperative localization in wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 22, pp. 54–69, Jul. 2005.

[24] H. Wymeersch, *Iterative Receiver Design.* Cambridge, U.K.: Cambridge Univ. Press, 2007.

[25] G. D. Forney, *Introduction to finite fields.* Principles in digital communication II, course 6.451, MIT, fail 2005. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-451-principles-of-digital-communication-ii-spring-2005/

[26] Tero Harju, *Lecture notes in graph theory.* Finland, Department of Mathematics University of Turku, 1994–2011.

[27] A. S. Balatsoukas, *Analysis and design of LDPC codes for the relay channel.* Undergraduate thesis, 2010, Technical University of Crete.

[28] C. Schlegel, L. Perez, *Trellis and Turbo Coding.* Wiley-IEEE Press, March 2004.

[29] S. Lin and D. J. Costello Jr., *Error Control Coding.* Pearson Prentice Hall, 2nd ed., 2004.

[30] Bernard C. Levy, *Principles of Signal Detection and Parameter Estimation*, New York: Springer, 2008.

[31] T. Richardson and R. Urbanke, *Modern Coding Theory.* Cambridge University Press, 2008

[32] Gilbert Strang, *Linear Algebra and its Applications*, Wellesley-Cambridge Press and SIAM, 4th ed. 2009

[33] D. Jungnickel, Graphs, *Networks and Algorithms.* Berlin, Germany: Springer-Verlag, 1999.