SUB-GRAPH APPROACH IN ITERATIVE SUM-PRODUCT ALGORITHM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUHAMMET FATIH BAYRAMOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Ali Özgür YILMAZ	Prof. Dr. Buyurman BAYKAL
Co-advisor	Supervisor

Examining Committee Members

Assoc. Prof. Dr. Engin TUNCER	(METU, EEE)	
Prof. Dr. Buyurman BAYKAL	(METU, EEE)	
Asst. Prof. Dr. Ali Özgür YILMAZ	(METU, EEE)	
Dr. Afşar SARANLI	(METU, EEE)	
Dr. Emre AKTAŞ	(Hacettepe University, EEE)	

I hearby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required, I have fully cited and referenced all material and results that are not original to this work.

Name Lastname : Muhammet Fatih BAYRAMOĞLU

Signature :

ABSTRACT

SUB-GRAPH APPROACH IN ITERATIVE SUM-PRODUCT ALGORITHM

BAYRAMOĞLU, Muhammet Fatih

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Buyurman BAYKAL

Co-Advisor: Asst. Prof. Dr. Ali Özgür YILMAZ

September 2005, 75 pages

Sum-product algorithm can be employed for obtaining the marginal probability density functions from a given joint probability density function (p.d.f.). The sum-product algorithm operates on a factor graph which represents the dependencies of the random variables whose joint p.d.f. is given. The sum-product algorithm can not be operated on factor-graphs that contain loops. For these factor graphs iterative sum-product algorithm is used.

A factor graph which contains loops can be divided in to loop-free sub-graphs. Sum-product algorithm can be operated in these loop-free sub-graphs and results of these sub-graphs can be combined for obtaining the result of the whole factor graph in an iterative manner.

This method may increase the convergence rate of the algorithm significantly while keeping the complexity of an iteration and accuracy of the output constant.

A useful by-product of this research that is introduced in this thesis is a good approximation to message calculation in factor nodes of the inter-symbol interference (ISI) factor graphs. This approximation has a complexity that is linearly proportional with the number of neighbors instead of being exponentially proportional. Using this approximation and the sub-graph idea we have designed and simulated joint decoding-equalization (turbo equalization) algorithm and obtained good results besides the low complexity.

Keywords: Sum-Product Algorithm, Divide and Conquer, LDPC Codes, Turbo Codes, Joint Equalization and Decoding

ÖZ

ITERATIF TOPLA-ÇARP ALGORITMASINDA ALT AG̃ YAKLAŞIMI

BAYRAMOĞLU, Muhammet Fatih
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Buyurman BAYKAL
Yardımcı Danışman: Yrd. Doç. Dr. Ali Özgür YILMAZ

Eylül 2005, 75 sayfa

Topla-çarp algoritması verilen bir tümleşik olasılık yoğunluk işlevinden marjinal olasılık yoğunluk işlevlerini bulmak için kullanılabilir. Topla-çarp algoritması rastgele değişkenlerin birbirlerine olan bağıntılarını gösteren bir faktör ağ üzerinde çalışır. Topla-çarp algoritması, döngü içeren faktör ağlar için kullanılamaz. Bu tür ağlar için iteratif topla-çarp algoritması kullanılmaktadır.

Döngü içeren bir faktör ağ, döngü içermeyen alt-ağlara bölünebilir. Topla-çarp algoritması döngü içermeyen bu alt-ağlarda çaliştırılır ve sonuçlar iteratif bir yöntemle birleştirilirse bütün faktör ağ için bir sonuç elde edilebilir.

Bu yöntem algoritmanın karmaşıklığını ve sonucun doğruluğunu azaltmadan yakınsama hızını ciddi ölcüde arttırabilir.

Bu tezde sunulan, araştırmamızın bir yan ürünü de semboller-arası girişim

faktör ağlarında mesaj hesaplamayla ilgili bir yaklaştırımdır. Bu yaklaştırımın karmaşıklığı komşu sayısı ile üstel olmak yerine doğru orantılıdır. Bu yaklaştırımı ve alt ağ fikrini kullanarak tasarladığımız bütünleşik kod çözme-denkleştirme (turbo denkleştirme) algoritmamız düşük karmaşıklığa sahip olmasının yanında iyi sonuç verdi.

Anahtar sözcükler: Topla-çarp algoritması, parçala ve fethet yöntemi, LDPC kodlar, turbo kodlar, bütünleşik kod çözme ve denkleştirme

ACKNOWLEDGMENTS

I would like to express my thanks to my supervisor Prof. Dr. Buyurman Baykal for his guidance and support during the preparation of this study.

I would also like to thank to my coadvisor Asst. Prof. Dr. Ali Özgür Yılmaz for his understanding, support, and valuable ideas.

I wish to express my thanks to my family, especially to my brother Etka, for their support and understanding.

Lastly but mostly, I would like to thank my wife Neslihan for her all kind of support, understanding and being a so lovely wife.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND MOTIVATION

Obtaining the marginal probability density function of a random variable from a given joint density function is an important problem in communication as in many other disciplines of science and technology. For some joint density functions which can be factored into smaller density functions, factor-graphs (FGs) and sum-product (SP) algorithm provides a useful and efficient tool for this purpose[1]. Since marginalization is a very common problem in different disciplines of science, this algorithm has been invented many times with different names during the history[11]. The well known BCJR [3, 1] algorithm, Pearl's belief propagation algorithm [14], Gallager's probablistic decoding algorithm of low-density parity-check (LDPC) codes[5] are all instances of the SP algorithm. Two specific instances of the SP algorithm, turbo and LDPC decoding algorithms, increased the interest on the algorithm recently[12].

Sum-product algorithm can be applied to the factor graphs which do not contain any loop. In the original sum-product algorithm, every node calculates its message according to a schedule. However, if the factor graph contain loops then obtaining such a schedule becomes impossible. Therefore, iterative sum-product algorithm is used [1]. Marginal density functions obtained from the iterative SP algorithm is not exact but approximate. A more severe problem of the iterative sum-product algorithm is that convergence of the algorithm is not guaranteed. Moreover, algorithm may require large num-

ber of iterations to converge, which may be prohibitive especially for mobile or delay sensitive applications. Research on the sum-product algorithm is focused on mainly these three problems [1, 2, 12].

1.1 Previous Work

In order to improve the accuracy of the output and increase the convergence rate Yedidia et. al. and McEliece et. al. have been proposed generalized belief propagation or equivalently belief propagation on partially ordered sets (poset belief propagation) algorithms [8, 9, 10]. In their approach they send joint beliefs instead of single beliefs between group of nodes rather than two single nodes. Although these studies achieve their goals, they increase the complexity of a single iteration a lot.

Our definition of a "sub-graph" may look similar to the "region-graph" definition of Yedidia et. al. The most significant difference between two definitions is that, loops may exist within a "region-graph", however, our "sub-graphs" should be loop free. Another difference is that, two region graphs may share a common factor-node. On the contrary, in our algorithm a factor node can be resident in only a single sub-graph. Although the definitions may look like similar, generalized belief propagation algorithm and our sub-graph SP algorithm is very different from each other. In the generalized belief propagation, compound (joint) messages (beliefs) are passed between group of nodes. On the other hand, sets of single messages are passed between sub-graphs in sub-graph SP algorithm. Dealing with joint messages increases both the memory and time complexity. Generalized belief propagation algorithms also increase the convergence rate and accuracy. However, complexity of a single iteration is increased. On the contrary, in

the sub-graph SP algorithm complexity of an iteration is kept the same as the iterative SP algorithm.

Worthen et. al [7] have proposed unified receiver factor graphs. Colavolpe et. al. [13] have dealt with the ISI factor graphs. They had proposed a method for how to obtain a factor graph for ISI channels having girth larger than four. Although their method keeps the computational complexity at the same order of magnitude, memory complexity of their method increases exponentially.

1.2 Motivation

Our focus on this research is devoted to improving both the convergence rate and the accuracy of the SP algorithm. Our main approach is interpreting the whole FG as a combination of small loop-free sub-graphs. We remind that SP algorithm obtains the actual solution on loop-free sub-graphs *non-iteratively*. Then cascade and parallel connections of sub-graphs are formed. These connection types define the information exchange rules between sub-graphs. Through this framework a loopy FG is transformed to cascade, parallel, or mixed connections of loop-free sub-graphs. Results obtained in each iteration is supplied as an input to the next iteration just as in any other iterative algorithm.

The algorithm proposed in this study is a modification of the iterative SP algorithm. As its origin, instances of our algorithm was invented before with different names. However, our algorithm is a general form of all those algorithms and allows designing new iterative algorithms in a systematic way. The two most famous examples of the known instances of our algorithm are the turbo and the LDPC decoding algorithms. Turbo decoding algorithm

can be viewed as a *cascade* connection of two subgraphs. On the other hand, the LDPC decoding algorithm can be viewed as a *parallel* connection of a number of sub-graphs. Our experimental results show that the sub-graphs connected in the cascade form have faster convergence rate than the sub-graphs connected in parallel. That is the reason why turbo codes have faster convergence rate than the LDPC codes. We have experimentally verified that if we connect the sub-graphs of the LDPC decoder in the cascade form convergence rate increases significantly. As a result, this research closes the gap between LDPC codes and turbo codes one step further.

When our work is compared with the previous work which aimed to increase the accuracy and/or the convergence rate, almost all of the methods include combining the messages or the variable nodes in order to deal with the adverse effects of the loops. These methods increase the memory and/or computational complexity of the algorithm exponentially. In our method we just change the message calculation schedule of the factor nodes.

Sub-graph approach also allows deeper understanding of iterative sum-product algorithm. Especially this approach shows clearly and apparently how to make convergence rate faster. Secondly, after our experimental results we now have a better understanding of the factors which degrades the accuracy of the output.

1.3 Organization

This thesis is organized as follows. Chapter 2 is a brief introduction to factor graphs and sum product algorithm. Definition of the sub-graph SP algorithm is given in Chapter 3. We discuss different design choices of the sub-graph SP algorithm in Chapter 4. We present different examples of applications of

the sub-graph SP algorithm together with some simulation results in Chapter 5. Our useful approximation for ISI factor graphs is also explained in this chapter. Final chapter is not a simple conclusion but also includes some important comments on the convergence and accuracy characteristics of the sub-graph SP algorithm.

CHAPTER 2

FACTOR GRAPHS AND SUM-PRODUCT ALGORITHM

In this chapter factor graphs and sum-product algorithm will be summarized, and notation used throughout the thesis will be introduced. For further information about factor graphs and sum-product algorithm [1] and [2] are authoritative references.

2.1 Factor Graphs

Although more general definitions of a factor graph exist in literature, for this thesis we will restrict ourselves to factor graphs for calculating marginal probability density functions from a given joint probability density function.

Definition 2.1. Factor Graph of joint probability density functions: Let $\mathbf{X}$ be a set of random variables $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$. Let the joint density function of the random variables in $\mathbf{X}$, $f_{\mathbf{X}}(X)$, can be factored into functions of X_i, $g_i(X_i)$, where X_is are subsets of X. In other words:

$$f_{\mathbf{X}}(X) = \prod g_i(X_i) \tag{2.1}$$

Then a factor graph is defined as a bipartite graph consisting of factor nodes assigned to each factor function $g_i(X_i)$ and variable nodes assigned to each

random variable $\mathbf{x_i}$. There is an edge between the ith factor node and kth variable node if and only if $x_k \in X_i$.

Note that in this definition $g_i(X_i)$'s do not have to be probability density functions. Also if there exists a priori information about some of the random variables, they are represented by factor functions, $g_i(X_i)$'s.

Example 2.1. As an example, consider a (4,2) code which has the following parity check matrix.

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

Assume that a codeword ,$(\mathbf{x_1 x_2 x_3 x_4})$ is transmitted through a memoryless channel and the vector $(y_1 y_2 y_3 y_4)$ is received. The problem of finding the marginal a posteriori densities of $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_4}$ can be represented with factor graph.

$$f_{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_4}}(x_1, x_2, x_3, x_4 | y_1, y_2, y_3, y_4) = \\ \frac{1}{Z} f_{\mathbf{x_1}}(x_1|y_1) f_{\mathbf{x_2}}(x_2|y_2) f_{\mathbf{x_3}}(x_3|y_3) f_{\mathbf{x_4}}(x_4|y_4) p_1(x_1, x_2, x_3) p_2(x_3, x_4) \quad (2.2)$$

where Z is a normalization constant and $p_1(x_1, x_2, x_3)$ and $p_2(x_3, x_4)$ are the factor functions corresponding to parity check equations. They can be defined as follows:

$$p_1(x_1, x_2, x_3) = \begin{cases} 1, & x_1 \oplus x_2 \oplus x_3 = 0 \\ 0, & x_1 \oplus x_2 \oplus x_3 \neq 0 \end{cases}$$

$$p_2(x_3, x_4) = \begin{cases} 1, & x_3 \oplus x_4 = 0 \\ 0, & x_3 \oplus x_4 \neq 0 \end{cases}$$

where $\oplus$ denotes addition in modulo 2.

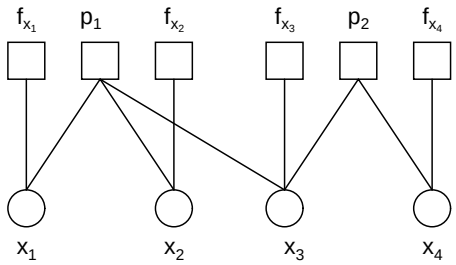


Figure 2.1: Factor graph representing the problem given in Example 2.1. Note that factor nodes are represented with squares and variable nodes are represented with circles

2.2 Sum-Product Algorithm

After we have defined what a factor graph is, now we can define sum-product algorithm. Sum-product algorithm is a message passing algorithm which operates on factor graphs. Messages that are sent by nodes during the sum-product algorithm are some kind of probability density functions. These messages are called as "belief"s sometimes. Similarly the sum-product algorithm may be called "belief propagation". The *belief* that a variable node sends to factor node is its probability density function, calculated by using the messages it has received from all of the neighboring nodes except the target node. On the other hand, a *belief* that a factor node sends to a variable node is the probability density function of the target variable node calculated by using the messages from all the neighboring nodes except the target node. It is important to emphasize that a message sent by a node is independent from the message it has received from the target node.

Conventional sum-product algorithm defines the message calculation schedule of the nodes and the content of the messages formally. Before giving the definition of the sum-product algorithm we should introduce some notations.

In our notation we assume that all of the variable nodes and factor nodes are numbered starting from 1. We also assume that i^{th} variable node represents the ${}^{th}i$ random variable.

- Let v_i denote the i^{th} variable node in factor graph and p_i denote the i^{th} factor node in the factor graph.
- Let $m_{i\to j}(x_i)$ be the message function from v_i to p_j and $n_{i\to j}(x_j)$ be the message function from p_i to v_j.
- Let M_i be the set of the indices of the neighbor nodes of the v_i, and N_i be the set of the indices of the neighbor nodes of the p_i.
- Let X_i be a set of variables which are represented by the neighbors of the i^{th} factor node and $X_{i\setminus j}$ means that all of the variables in X_i except the one represented by v_j.

Then standard sum-product algorithm can be defined as follows [1]:

Definition 2.2. Sum-Product Algorithm (Belief propagation): Given a factor graph of a joint probability density function, sum product algorithm can be defined on that factor graph as follows:

- Message Calculation Schedule
  - Any node p (factor node or variable node) waits for messages from its neighbors until only one neighbor remains yet to send its message.
  - If any node p has received messages from all of its neighbors except neighbor r, p sends a message to r, and starts to wait for a message from r.

- When the node p receives a message from r, then it sends messages to all the remaining neighbors.
- Message passing terminates when two messages pass across all the edges in the factor graph in both directions.

- Message Content:
  - $m_{i\to j}(x_i)$ is calculated by point-by-point multiplication of the incoming message functions to v_i excluding the one coming from p_j, i.e.,
$$m_{i\to j}(x_i) = \prod_{k\in M_i\setminus\{j\}} n_{k\to i}(x_i) \tag{2.3}$$
  - $n_{i\to j}(x_j)$ can be calculated in two steps. In the first step all of the messages received by p_i except the one coming from v_j is multiplied together with the factor function that p_i represents, $f_i(X_i)$, and intermediate function, $\widetilde{p}_i(X_i)$,is obtained. i.e.
$$\widetilde{p}_i(X_i) = f_i(X_i) \prod_{k\in N_i\setminus\{j\}} m_{k\to i}(x_k) \tag{2.4}$$
    In the second step $\widetilde{p}_i(X_i)$ is marginalized on to x_j and $n_{i\to j}(x_j)$ is obtained.
$$n_{i\to j}(x_j) = \sum_{\forall X_{i\setminus j}} \widetilde{p}_i(X_i) \tag{2.5}$$
    If the random variables of the factor graph are of continuous type rather than discrete type then the sum operator in Equation (2.5) can be replaced by the integral operator.

- Finalization:

  Final step of the algorithm is obtaining the marginal density functions of the variables that the factor graph represent. This task can be

accomplished by multiplying all of the received messages at a variable node and then normalizing it as in the below equation.

$$f_{\mathbf{x_i}}(x_i) = \frac{\prod_{k \in M_i} n_{k \to i}(x_i)}{\sum_{x_i} \prod_{k \in M_i} n_{k \to i}(x_i)} \tag{2.6}$$

In an informal fashion sum-product algorithm can be summarized as follows:

- A message from node p to r does not depend on the message node r sends to p. However, it depends on all the other messages that node p receives.
- Messages are calculated when they are able to be calculated, i.e. all of the necessary messages to calculate that message are received.
- The message that is sent from a variable node v to a factor node p, is the *belief* of node v about its density given all the other messages except from node p.
- The message that is sent from a factor node p to a variable node v is the *belief* of node p about the density of the node v given all the other messages except from node v.

2.3 Iterative Sum-Product Algorithm

There is an apparent restriction in the definition of the sum-product algorithm. If the factor graph contains loops this algorithm does not work. We will explain this fact with the help of an example. Figure 2.2 illustrates a factor graph which contains a loop. Let us try to apply message calculation schedule of the SP algorithm. Variable nodes v_3 and v_4 have just one

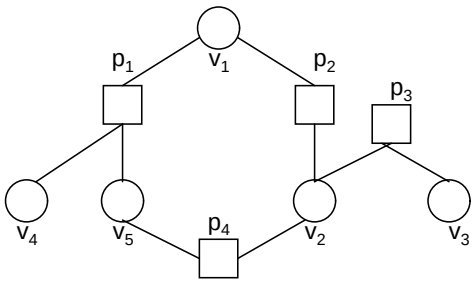


Figure 2.2: An example factor graph which includes a loop

neighbor. Therefore they do not need anything to calculate their messages. Thus they calculate their messages and send to p_3 and p_1 respectively. p_3 has two neighbors and receives the message from one of them. Factor node p_3 calculates its message for v_2 and sends, after receiving its message from v_3. After this point algorithm stalls. None of the nodes resident on the loop receive messages from two of their neighbors. Therefore, neither of them can calculate any message. This phenomenon is valid for all of the factor graphs that contain loop. A node resident on a loop can never receive messages from two of its neighbors with which it shares the loop. As a result a node which lay on a loop can never calculate its messages and the conventional sum-product algorithm does not work.

Solution to this problem is to change the message calculation schedule of the sum-product algorithm. Resulting algorithm is the iterative sum-product algorithm. Message content and the finalization of the iterative sum-product algorithm is the same as the conventional sum-product algorithm. Therefore we only give message calculation schedule of the iterative sum-product algorithm.

Definition 2.3. Iterative Sum-Product Algorithm

- Step 1: All of the variable nodes send uniform density functions as

messages.

- Step 2: All of the factor nodes calculate their messages using the messages they receive at the previous step and send to variable nodes.
- Step 3: All of the variable nodes calculate their messages using the messages they receive at the previous step and send to factor nodes.
- Step 4: Go to step 2, until a fixed number of iterations is reached or a specific termination condition is satisfied.

It should be emphasized that a message received by a node p through an edge e at iteration i, is *independent* from the message sent by p through edge e at iteration $i-1$.

Although convergence of this algorithm is not guaranteed, it has found many application areas. The most important application of this algorithm is probably the decoding algorithm of the LDPC codes.

There exists more complex schedules than the one given in the above definition in the literature. One of those complex schedules is the flooding schedule [15]. In this schedule variable nodes and factor nodes calculate their messages at the same time instant. In the next time instant all of the nodes calculate their messages based on the messages they receive in the previous time instant. This schedule suits parallel implementation better.

If the iterative sum-product algorithm is run on a loop free factor graph, after a certain number of iterations, which is determined by the longest distance between two nodes, exactly the same result of the conventional sum-product algorithm is obtained. If the complexities of the two algorithms are compared it is observed that complexity of the conventional SP algorithm is equivalent to the complexity of the one iteration iterative SP algorithm.

The reason for this fact can be explained as follows; complexity of these algorithms is dominated by the message calculation. For the conventional SP algorithm two messages pass across an edge during the whole algorithm. On the other hand, in the iterative SP algorithm two messages pass across an edge during just one iteration. Therefore, complexity of the conventional SP algorithm is equivalent to the complexity of the one iteration of the iterative SP algorithm. The dual of this statement is that iterative SP algorithm requires some number of iterations to converge, on the other hand conventional SP algorithm converges in just a single iteration.

Our sub-graph sum-product algorithm is actually just a different scheduling to the iterative sum-product algorithm. Our algorithm allows more complex schedules in a very systematic way. Next chapter explains this method in detail.

CHAPTER 3

SUB-GRAPH SUM PRODUCT ALGORITHM

As it is stated in the closing parts of the last chapter conventional SP algorithm can be viewed as a kind of iterative SP algorithm which converges in just a single iteration. However, this algorithm is restricted to only loop free factor graphs. However, many of the factor graphs that we encounter in practice contains loops. For example, decoding factor graph of an LDPC code contains long cycles, an ISI cancellation factor graph contains many short cycles. The idea of sub-graph sum-product algorithm that we propose arises from the question how can those fast convergence characteristics of the conventional SP algorithm can be applied to loopy factor graphs. One answer to this question is given by dividing the factor graph into sub-graphs which do not contain any loops. Since these sub-graphs are loop free, the conventional SP algorithm can be applicable. However, the ways to combine the information obtained from those sub-graphs, should be well defined. This chapter starts by defining the rules of partitioning a factor graph into such sub-graphs. Then definition of the interconnection of these sub-graphs will be discussed.

3.1 Sub-Graphs

The phrase "sub-graph" can be used to refer to any portion of a graph in general. However, in this context we assign a special meaning to "sub-graph". What we mean by a sub-graph is a portion of a factor graph on which conventional SP algorithm can be run.

Before giving a formal definition of the sub-graph, we shall explain its mathematical basis. The factorization equation should be reconsidered for this purpose. Assume that the global function is factored into n functions, and $R_1, R_2, \ldots, R_k$ are mutually exclusive subsets of the set of numbers from 1 to n such that $\bigcup_{\forall i} R_i = \{1, 2, \ldots, n\}$. Then we can modify the Equation (2.1) as below:

$$\begin{aligned} f_{\mathbf{X}}(X) &= \prod_{i=1}^{n} g_i(X_i) \\ &= \prod_{i \in R_1} g_i(X_i) \prod_{i \in R_2} g_i(X_i) \ldots \prod_{i \in R_k} g_i(X_i) \\ &= \prod_{i=1}^{k} r_i(\hat{X}_i) \end{aligned} \tag{3.1}$$

where $\hat{X}_i = \bigcup_{l \in R_i} X_l$ and

$$r_i(\hat{X}_i) = \prod_{l \in R_i} g_l(X_l) \tag{3.2}$$

Equation (3.2) shows that r_i's are functions which can be factored into functions of smaller number of arguments. Thus, marginalization problem of r_i's can be represented with factor graphs. The original factor graph, which represents the marginalization problem of $f_{\mathbf{X}}(X)$ is the superposition of these newly formed small factor graphs, which will be called "sub-graphs" from now on. In other words the problem has been divided into pieces of the same kind. However, a method is required to combine the results of the sub-graphs to

obtain the result of the whole problem. Sub-graph sum product algorithm provides an iterative solution for combining the solutions of the sub-graphs. Another problem is how to solve the marginalization problem for these sub-graphs. Since these sub-graphs are actually factor-graphs, conventional sum-product algorithm or iterative sum-product algorithm can be used to solve this problem. It is reasonable to choose R_i's such that the resulting sub-graphs are loop-free. In this case conventional sum-product algorithm, which is computationally efficient, can be employed to solve the marginalization problem of the sub-graphs. If the resulting sub-graphs contain loops, iterative sum product can be employed in a trivial way. A more interesting option could be applying the sub-graph sum-product algorithm to those sub-graphs which contain loops in a *recursive* fashion. In this chapter we will force sub-graphs to be loop-free.

Definition 3.1. Sub-graph: A sub-graph, represented by S, is a region of a factor graph which does not contain any loop, and if factor node p is included in a sub-graph S all of the variable nodes connected to p should be included in S.

This definition is a direct consequence of Equations (3.1) and (3.2). A factor graph can be partitioned into sub-graphs in many different ways. However, not all of the different partitioning schemes are appropriate. Therefore we should define which partitioning schemes are proper.

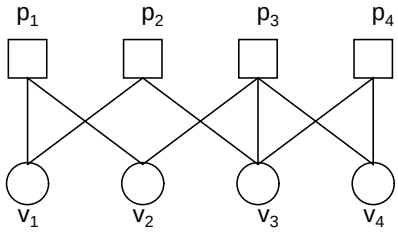
Definition 3.2. Proper partitioning of a factor graph: A partitioning of factor graph is proper if and only if all factor nodes in the factor graph is contained in a sub-graph, and only one sub-graph.

This definition is a result of the fact that R_i's should be mutually exclusive and complete.

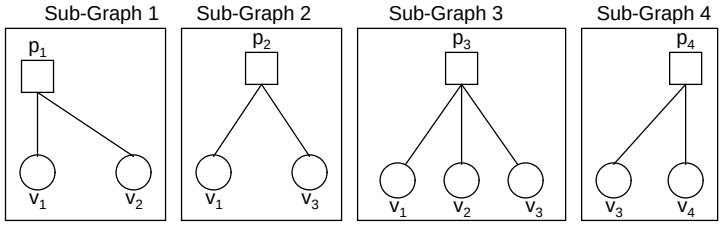
A question that may arise is whether a proper partitioning exists for every factor graph. The answer is positive. First of all, there exists a trivial partitioning, in which every sub-graph is composed of just a single factor node and all of the variable nodes connected to it. In other words every subset $R_i = \{i\}$. Depending on the factor graph, other partitioning schemes may exist. Different partitioning methods will result in different convergence and accuracy characteristics. These subjects will be explained in Chapter 4 and Chapter 5 in detail.

Example 3.1. Figure 3.1 shows an example of partitioning a factor graph into sub-graphs. The trivial partitioning is shown in part (b) of the figure. Part (c) and part (d) of the figure shows different partitioning schemes. We will name these partitioning schemes as *all-accessible partitioning* and *non-accessible partitioning* respectively in Section 4.

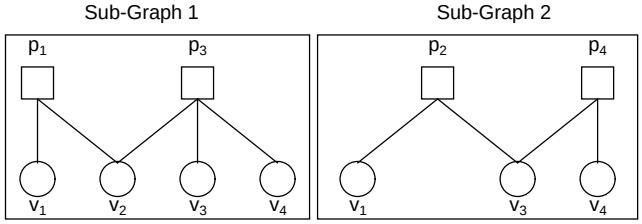
Up to this point we have defined sub-graphs and proper partitioning. In other words we have defined how to divide the problem into smaller pieces. Since we force sub-graphs to be loop-free, we can use the standard sum-product algorithm method to solve the problem for these smaller pieces. The only remaining part is how to construct the solution of the whole problem from the solutions of these sub-graphs. We will use block diagram representation while explaining our sub-graph sum-product algorithm which is intended to combine the results from sub-graphs.



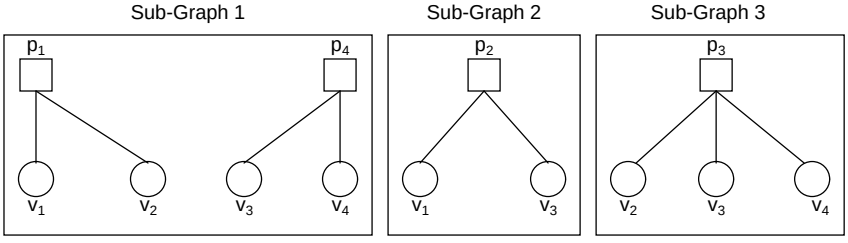
(a) The original factor graph



(b) The trivial partitioning



(c) Another proper partitioning



(d) Another proper partitioning

Figure 3.1: An example of partitioning a factor graph into sub-graphs

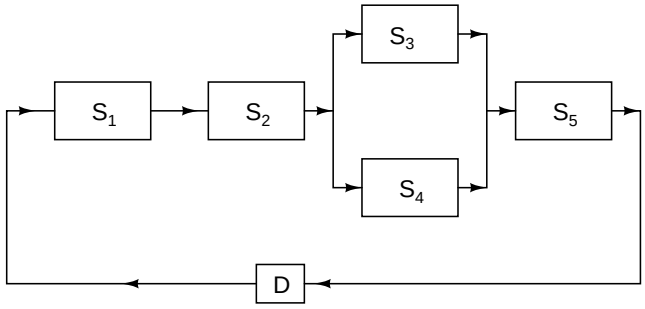


Figure 3.2: A simple flow graph of sub-graphs. In this figure S_is represent subgraphs and D represents a delay element.

3.2 Block Diagram Representation of Sub-Graphs

Block diagrams make the understanding of many systems much simpler. Therefore, we will use block diagrams for representing sub-graphs and sub-graph sum-product algorithm. The blocks are connected in cascade or in parallel or a mixture of these two to form a directed flow graph. Since our algorithm is an iterative one, we should represent the iterations. An iteration will be represented by feeding the output of the flow graph back to input of the flow graph through a delay element. Figure 3.2 shows a simple flow graph of sub-graphs.

Our block diagram representation is a little bit more complex than the block diagram representations that may be encountered in literature. In the usual block diagram representations, blocks have inputs and they generate outputs. On the other hand, sub-graph blocks have also *states* in our representation. In each iteration they store the information that they have generated in order to use it in the next iteration. Another way of saying this is that sub-graph blocks have two inputs and two outputs, and one of the

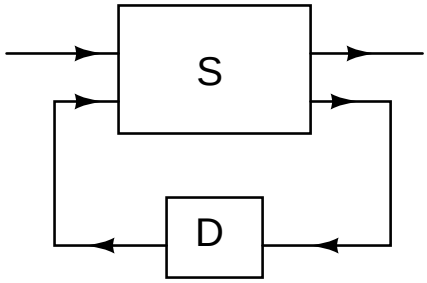


Figure 3.3: Showing the hidden state of sub-graph block

outputs is fed back to the one of the inputs through a delay element. This idea is depicted in Figure 3.3. However, in order to simplify the representation we will hide the state, and assume that it exists inside a subgraph block as in Figure 3.2.

Signals flowing through the edges of these flow graphs are vectors of beliefs about the variable nodes included in the factor graph. Basically input and state of a sub-graph will be used as a priori information and sum-product algorithm will be run. The output of the SP algorithm will determine both the output and the state. Details of this process will be explained in Section 3.3 in detail.

Another important point to be emphasized is that the signals flowing through the edges are beliefs about *all* variables included in the whole factor graph. Possibly, there will exist some sub-graphs which do not include all of the variable nodes. In this case they will have been received some information about a variable which they do not include. In order to solve this problem, we enforce all of the variable nodes to be included in all sub-graphs. However, a variable node included in the sub-graph does not have to be connected to some factor node. Note that this enforcement does not conflict with our definition of sub-graph.

3.3 Structure and Operation of a Sub-Graph Block

In this section we will describe in detail what happens when a sub-graph block is invoked.

The basic idea is that an input received by a sub-graph is a combination of beliefs which are generated by other sub-graphs and the information generated by this sub-graph during the previous iteration. The information generated by other sub-graphs will be used as a priori information for this sub-graph. However, input contains also the information generated by this sub-graph during previous iteration. This information should be separated from the input. If it is not, this causes to converge to a wrong point. Therefore, every sub-graph should store the information that it has generated. The sub-graph will separate the information from the input it received at the next iteration. This idea is similar to the"extrinsic information" idea in turbo decoding literature, Output of a sub-graph is the result of the standard sum-product algorithm.

While explaining the algorithm with the help of block diagram representation, we need some notation for the signals of the flow graph. Remember that these signals are vectors of beliefs, or in other words vectors of probability density functions. We will denote the vectors of beliefs with boldface capital letters with a ($\tilde{}$) sign on top of them. Input to the j^{th} sub-graph will be denoted by $\tilde{\mathbf{I}}_j$, output of that sub-graph will be denoted by $\tilde{\mathbf{O}}_j$, and finally state of that sub-graph will be denoted by $\tilde{\mathbf{X}}_j$. In other words,

$$\tilde{\mathbf{I}} = (b_{in_1}(x_1), b_{in_2}(x_2), \ldots, b_{in_i}(x_i)) \tag{3.3}$$

$$\tilde{\mathbf{O}} = (b_{out_1}(x_1), b_{out_2}(x_2), \ldots, b_{out_i}(x_i)) \tag{3.4}$$

$$\tilde{\mathbf{X}} = (b_{st_1}(x_1), b_{st_2}(x_2), \ldots, b_{st_i}(x_i)) \tag{3.5}$$

where, b_{in}'s represent input beliefs, b_{out}'s represent output beliefs, and b_{st}'s represent state beliefs. Finally, if we apply $\widetilde{\mathbf{I}}$ as input to a sub-graph S having state $\widetilde{\mathbf{X}}$ and output $\widetilde{\mathbf{O}}$ then we represent it with

$$\tilde{\mathbf{I}} \xrightarrow{(S,\tilde{\mathbf{X}})} \tilde{\mathbf{O}}.$$

Sometimes we may drop the state in this representation and write

$$\tilde{\mathbf{I}} \xrightarrow{S} \tilde{\mathbf{O}}.$$

We also need two operations defined on these vectors. One of them is the combination operation and will be denoted by $\otimes$ and the other one is the separation operation and will be denoted by $\oslash$. If we denote jth component of a vector by $(.)_j(x_j)$ then

$$(\tilde{\mathbf{A}} \otimes \tilde{\mathbf{B}})_j(x_j) = \frac{(\tilde{\mathbf{A}})_j(x_j)(\tilde{\mathbf{B}})_j(x_j)}{\sum[(\tilde{\mathbf{A}})_j(x_j)(\tilde{\mathbf{B}})_j(x_j)]} \quad \forall j \tag{3.6}$$

$$(\tilde{\mathbf{A}} \oslash \tilde{\mathbf{B}})_j(x_j) = \frac{(\tilde{\mathbf{A}})_j(x_j)/(\tilde{\mathbf{B}})_j(x_j)}{\sum[(\tilde{\mathbf{A}})_j(x_j)/(\tilde{\mathbf{B}})_j(x_j)]} \quad \forall j \tag{3.7}$$

In other words, combination operation is the normalized point by point multiplication, and separation operation is the normalized point by point division. With the help of this representation, we can define the operation of a sub-graph more clearly and more formally.

The first operation that a sub-graph should perform is to separate the information generated by the other sub-graphs from its input. Let $\tilde{\mathbf{R}}$ denote the information generated by other sub-graphs. Then $\tilde{\mathbf{R}}$ can be obtained by the following equation:

$$\tilde{\mathbf{R}} = \tilde{\mathbf{I}} \oslash D(\tilde{\mathbf{X}}) \tag{3.8}$$

where $D(.)$ is the delay operator.

Then the elements of $\tilde{\mathbf{R}}$ will be used as a priori information for this sub-graph. Recall that in Chapter 2 we have stated that a priori information for

a random variable is nothing but a factor function, thus it can be represented by a factor node. Therefore, artificial factor nodes will be included in the sub-graphs, one for each variable node, in order to represent the a priori information. An artificial factor node representing the belief $(\tilde{\mathbf{R}})_j(x_j)$ will be connected to the j^{th} variable node only. Since artificial factor nodes are connected to a single variable node, it is guaranteed that sub-graphs are still loop-free.

Now sub-graphs are ready to operate the standard sum-product algorithm. After sum product algorithm operates, we obtain b_{out}'s. Note that, the output of the standard sum-product algorithm is not called *beliefs*, they are marginal density functions. However, we call the output of the sum-product algorithm operating on a sub-graph as beliefs, since the sub-graph is not the complete factor graph but a portion of it. Therefore, obtained results may not be true, and calling them as beliefs is reasonable. These b_{out}s are used for obtaining the output vector, $\tilde{\mathbf{O}}$, as in Equation (3.4). The only remaining vector that is not explained is the state vector, $\tilde{\mathbf{X}}$. Remember that, the purpose of the vector $\tilde{\mathbf{X}}$ is to hold the information generated by the sub-graph. Now think of the j^{th} element of the vector $\tilde{\mathbf{O}}$. As Equation (2.6) shows $(\tilde{\mathbf{O}})_j(x_j)$ is the multiplication of the incoming messages from factor nodes and a normalization constant. One of these factor nodes is the artificial factor node representing $(\tilde{\mathbf{R}})_j(x_j)$. Then we can write $(\tilde{\mathbf{O}})_j(x_j)$ as

$$(\tilde{\mathbf{O}})_j(x_j) = \frac{1}{Z}(\tilde{\mathbf{R}})_j(x_j) \prod_i n_{i\to j}(x_j) \tag{3.9}$$

The information generated in this sub-graph is the $\prod_i n_{i\to j}(x_j)$ part. This is the part that the vector $\tilde{\mathbf{X}}$ should hold. It can be simply obtained by the following equation.

$$\tilde{\mathbf{X}} = \tilde{\mathbf{O}} \oslash \tilde{\mathbf{R}} \tag{3.10}$$

Figure 3.4 depicts the structure and operation of a sub-graph block schematically.

3.4 Connection Types of Sub-Graphs

When the sub-graph sum-product algorithm is represented with block diagrams we should also define the connection rules between these sub-graph blocks. Basically there are two connection types, which are cascade and parallel connection.

3.4.1 Cascade Connection

Cascade connection type is very simple and obvious. Basically, this connection type consists of connecting the $\tilde{\mathbf{O}}$ vector of a sub-graph to $\tilde{\mathbf{I}}$ vector of the next sub-graph. We will denote connecting two sub-graphs in cascade with the $\rightarrow$ sign.

When sub-graphs connected in cascade they form a chain of sub-graphs. Chains can be composed of any number of sub-graphs. We will name chains with letter C. For instance if we form a chain C by connecting sub-graphs S_1 and S_2 in cascade such that S_1 operates before S_2 as in Figure 3.5, it will be denoted by $C = S_1 \rightarrow S_2$. Chains can also include parallel connection of chains, which will be defined in Section 3.4.2.

3.4.2 Parallel Connection

Parallel connection is more complex than the cascade connection. Figure 3.6 shows a simple parallel connection of two chains. This figure shows that when chains are connected in parallel, there exists a signal fork and a signal

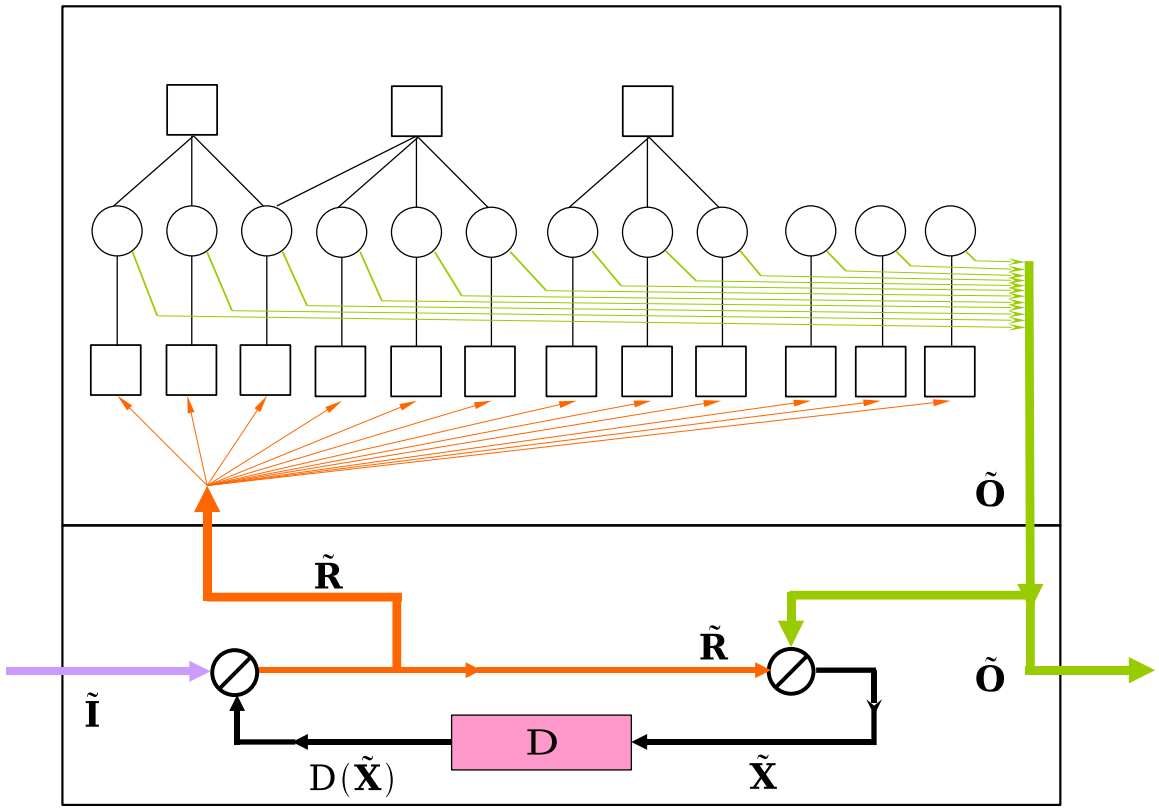


Figure 3.4: Structure of a sub-graph block

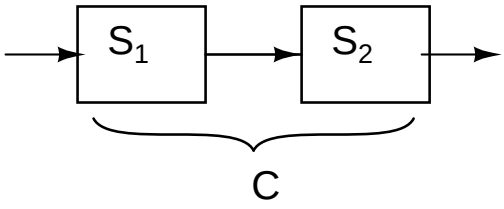


Figure 3.5: Cascaded connection of two sub-graphs to form a chain

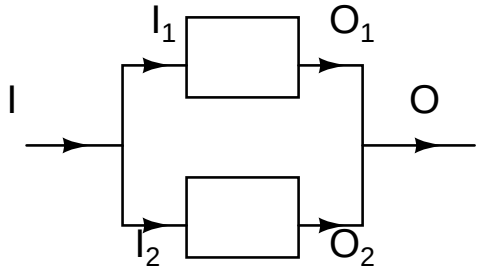


Figure 3.6: An example parallel connection of sub-graphs

junction. Signal forks are common to almost all block diagram representations. They mean simply that all forked signals are equal to the input signal, which is the same as our case. On the other hand signal junction points on other signal flow diagram representations usually makes some operations, such as addition or multiplication. In our sub-graph flow diagrams signal junction points also perform an operation, but this is not shown on the diagram extrinsicly, since it is always the same operation. In the sub-graph flow diagrams junction points does not combine *any* two signals but they combine two signals which are originated from same fork point.

We denote the parallel connection of two sub-graphs or chain of sub-graphs with $\|$ sign. For example if we connect S_1 and S_2 in cascade and S_3 in parallel to these two, then we represent it with $(S_1 \rightarrow S_2) \parallel S_3$.

Definition 3.3. Parallel Connection of Chains: Let two chains C_1 and C_2

are connected in parallel. Let these two chains have inputs $\widetilde{\mathbf{I}}_1$ and $\widetilde{\mathbf{I}}_2$ and output $\widetilde{\mathbf{O}}_1$ and $\widetilde{\mathbf{O}}_2$ respectively. Let $\widetilde{\mathbf{I}}$ be the signal before the fork point and $\widetilde{\mathbf{O}}$ be the signal after the junction point, i.e.

$$\widetilde{\mathbf{I}}_1 \xrightarrow{C_1} \widetilde{\mathbf{O}}_1$$

$$\widetilde{\mathbf{I}}_2 \xrightarrow{C_2} \widetilde{\mathbf{O}}_2$$

and

$$\widetilde{\mathbf{I}} \xrightarrow{C_1 \| C_2} \widetilde{\mathbf{O}}$$

Then $\widetilde{\mathbf{I}}_1$,$\widetilde{\mathbf{I}}_2$, and $\widetilde{\mathbf{O}}$ is given by

$$\widetilde{\mathbf{I}}_1 = \widetilde{\mathbf{I}} \tag{3.11}$$

$$\widetilde{\mathbf{I}}_2 = \widetilde{\mathbf{I}} \tag{3.12}$$

$$\widetilde{\mathbf{O}} = \widetilde{\mathbf{O}}_1 \otimes \widetilde{\mathbf{O}}_2 \oslash \widetilde{\mathbf{I}} \tag{3.13}$$

Theorem 3.1. $\|$ *operation is associative. In other words, if*

$$\widetilde{\mathbf{I}} \xrightarrow{(C_1 \| C_2) \| C_3} \widetilde{\mathbf{O}}$$

then

$$\widetilde{\mathbf{I}} \xrightarrow{C_1 \| (C_2 \| C_3)} \widetilde{\mathbf{O}}$$

.

Proof: It is obvious that all of the chains have the same input $\widetilde{\mathbf{I}}$*. Let* $\widetilde{\mathbf{I}} \xrightarrow{C_1} \widetilde{\mathbf{O}}_1$, $\widetilde{\mathbf{I}} \xrightarrow{C_2} \widetilde{\mathbf{O}}_2$, $\widetilde{\mathbf{I}} \xrightarrow{C_3} \widetilde{\mathbf{O}}_3$, $\widetilde{\mathbf{I}} \xrightarrow{C_1 \| C_2} \widetilde{\mathbf{O}}_4$, *and* $\widetilde{\mathbf{I}} \xrightarrow{C_2 \| C_3} \widetilde{\mathbf{O}}_5$ *. Then ;*

$$\widetilde{\mathbf{O}}_4 = \widetilde{\mathbf{O}}_1 \otimes \widetilde{\mathbf{O}}_2 \oslash \widetilde{\mathbf{I}} \tag{3.14}$$

$$\widetilde{\mathbf{O}}_5 = \widetilde{\mathbf{O}}_2 \otimes \widetilde{\mathbf{O}}_3 \oslash \widetilde{\mathbf{I}} \tag{3.15}$$

Let $\widetilde{\mathbf{I}} \xrightarrow{(C_1 \| C_2) \| C_3} \widetilde{\mathbf{O}}_6$ *and* $\widetilde{\mathbf{I}} \xrightarrow{C_1 \| (C_2 \| C_3)} \widetilde{\mathbf{O}}_7$. *We should prove that* $\widetilde{\mathbf{O}}_6 = \widetilde{\mathbf{O}}_7$.

$$\begin{aligned} \widetilde{\mathbf{O}}_6 &= \widetilde{\mathbf{O}}_4 \otimes \widetilde{\mathbf{O}}_3 \oslash \widetilde{\mathbf{I}} \\ &= \widetilde{\mathbf{O}}_1 \otimes \widetilde{\mathbf{O}}_2 \oslash \widetilde{\mathbf{I}} \otimes \widetilde{\mathbf{O}}_3 \oslash \widetilde{\mathbf{I}} \end{aligned} \tag{3.16}$$

and

$$\begin{aligned} \widetilde{\mathbf{O}}_7 &= \widetilde{\mathbf{O}}_1 \otimes \widetilde{\mathbf{O}}_5 \oslash \widetilde{\mathbf{I}} \\ &= \widetilde{\mathbf{O}}_1 \oslash \widetilde{\mathbf{I}} \otimes \widetilde{\mathbf{O}}_2 \otimes \widetilde{\mathbf{O}}_3 \oslash \widetilde{\mathbf{I}} \\ &= \widetilde{\mathbf{O}}_1 \otimes \widetilde{\mathbf{O}}_2 \oslash \widetilde{\mathbf{I}} \otimes \widetilde{\mathbf{O}}_3 \oslash \widetilde{\mathbf{I}} \end{aligned} \tag{3.17}$$

Since Equation (3.16) is equal to Equation (3.17) proof has been completed.

Theorem 3.1 allow us to connect any number of chains in parallel.

3.5 Result of the Sub-Graph Sum-Product Algorithm

Sub-graph sum-product algorithm is an iterative algorithm. After some certain number of iterations or when a specific termination condition is satisfied, the algorithm stops. After the algorithm stops, output vector of the last sub-graph, i.e. the one just before the delay element, gives the result of the sub-graph sum-product algorithm. Elements of this vector can be considered as the marginalized probability density functions of the random variables.

Our experimental results show that when the sub-graph sum-product converges, output vector of each sub-graph is equal to its input vector. This means that any vector at the main branch of the algorithm can be considered as the output vector. However, this requires convergence of the algorithm. If iterations are stopped for a specific reason, for instance all of the parity check

equations are satisfied in LDPC decoding problem, then only the output of the last sub-graph should be taken as the result of the algorithm.

3.6 Summary of the Sub-Graph Sum-Product Algorithm

In this chapter the sub-graph sum-product algorithm is explained with all details. Before closing the chapter the sub-grah sum-product algorithm will be summarized.

For a given factor graph, the first step of the sub-graph sum-product algorithm is partitioning the sub-graph into loop-free sub-graphs. Secondly, a flow graph sub-graphs should be constructed. Then this flow should be operated for each iteration. Finally the result of the algorithm is obtained as in Section 3.5.

CHAPTER 4

VARIATIONS OF THE SUB-GRAPH SUM-PRODUCT ALGORITHM

We have to decide on some parameters while designing a specific sub-graph sum-product algorithm. These parameters include how to partition the factor graph into sub-graphs and how to connect these sub-graphs. These variations are explained in this chapter.

4.1 Partitioning a Factor Graph into Sub-Graphs

Deciding on which factor node is included in which sub-graph is one of the most important problems in designing a sub-graph sum-product algorithm for a *given factor-graph*. Remember that this process should obey the rules given in definitions 3.1 and 3.2. It is apparent that there exist many different solutions for this problem. It is a very difficult task to choose the *best* solution in this case, since there are too many of them. Even determining the number of possible choices of sub-graphs is a difficult problem. Moreover, we do not have an efficient method to evaluate and compare the different solutions other than simulation.

On the other hand the way of partitioning into sub-graphs affects the accuracy and the convergence characteristics of the algorithm considerably. Therefore, we should somehow analyze this problem.

Our approach for this case is to analyze some specific sub-graph partitioning strategies which will provide us a deeper understanding of the sub-graph sum-product algorithm. We think that the most important criterion is the *accessibility* property of sub-graphs. What we mean by accessibility is how many different factor nodes can be accessed from a factor node in a sub-graph. The two extreme type of sub-graphs are all-accessible and non-accessible sub-graphs. We have made our simulations with these two type of sub-graphs.

4.1.1 All-Accessible Sub-Graphs

In an all-accessible sub-graph partitioning there exists a path between *any* two factor nodes included in that sub-graph. Again, there may be several different choices of sub-graph partitioning for a given factor graph such that every resulting sub-graph is all-accessible. We did not delve into the subject of how to choose among all-accessible solutions. We have designed a simple and efficient algorithm to find all-accessible sub-graph partitioning from a given factor graph.

Definition 4.1. All-accessible sub-graph construction algorithm:

- Step 1 (Initialization):
  - Sub-step 1 : Make a list of factor nodes.
  - Sub-step 2 : Construct an initial empty sub-graph.
  - Sub-step 3 : Pick a factor node from the factor node list and add it to the sub-graph together with the variable node neighbors of it, and delete the factor node from the factor-node list.
- Step 2 :Pick a factor node from the factor node list.

- Step 3 :Find a suitable sub-graph for the factor node.
  - Sub-step 1 : Set $i = 1$.
  - Sub-step 2 : Check if the factor node is connected to a variable node included in the i^{th} sub-graph. If the answer is negative go to sub-step 4.
  - Sub-step 3 : Check if the factor node is connected to at most one variable node included in the i^{th} sub-graph.
    - If the result is positive add factor node to the i^{th} sub-graph together with the variable node neighbors of it, and delete it from the factor node list and go to step 5.
    - If the result is negative go to sub-step 4.
  - Sub-step 4 : Increment i until i reaches the number of sub-graphs.
- Step 4 :Construct a new sub-graph and add the factor node to this newly formed sub-graph together with the variable node neighbors of it.
- Step 5 :Go to step 2 until factor node list becomes empty.

4.1.2 Non-Accessible Sub-Graphs

Non-accessible sub-graphs are the opposite of the all-accessible sub-graphs. In non accessible sub-graphs there are no paths between any two factor node. Similar to the all-accessible sub-graphs there may be several different solutions for forming non-accessible sub-graphs. We have designed a similar algorithm to that defined in definition 4.1.

Definition 4.2. Non-accessible sub-graph construction algorithm:

- Step 1 (Initialization) :
  - Sub-step 1 : Make a list of factor nodes.
  - Sub-step 2 : Construct an initial empty sub-graph.
  - Sub-step 3 : Pick a factor-node from the factor node list and add it to the sub-graph together with the variable node neighbors of it, and delete from the factor-node list.
- Step 2:Pick a factor node-from the factor node list.
- Step 3:Find a suitable sub-graph from the factor node.
  - Sub-step 1 : Set $i = 1$.
  - Sub-step 2 : Check if the factor node is connected to a variable node included in i^{th} sub-graph. If the answer is negative add the factor node to this sub-graph together with the variable node neighbors of it, and delete it from the factor node list. Go to step 5.
  - Sub-step 3 : Increment i until i reaches the number of sub-graphs.
- Step 4 :Construct a new sub-graph and add the factor node to this newly formed sub-graph together with the variable node neighbors of it, and delete it from the factor node list.
- Step 5 :Go to step 2 until factor node list is empty.

The algorithms that we have defined in definitions 4.1 and 4.2 are guaranteed to find a proper partitioning.

4.2 Connecting Sub-Graphs

Another important design parameter is deciding on the connection pattern of sub-graphs. Two different connection types of sub-graphs are available while constructing a flow graph of sub-graphs. These two connection types are cascade and parallel connection as explained in Chapter 3. These two connection types allows to construct lots of different types of sum-product algorithms.

Our experimental results show that convergence rate of the cascade connection is *usually* better than parallel connection. However, connecting sub-graphs in cascade form may degrade the accuracy of the output. These experimental results will be presented in Chapter 5 in detail.

4.3 Non-Homogeneous Flow Graph of Sub-Graphs

For some problems, using a non-homogeneous flow-graph of sub-graphs may yield better results. Joint decoding and equalization problem may be an example for this case. We call a flow graph non-homogeneous if it contains more than one instance of a sub-graph. Instances of a sub-graph may exist more than once in a flow graph of sub-graphs, if some conditions is satisfied.

First condition is same sub-graph may not be encountered in parallel chains. In other words, if chains C_1 and C_2 are connected in parallel, or some larger chains which include C_1 and C_2 are connected in parallel these two chains cannot include same sub-graph. If this condition is violated, an instance of the same sub-graph will use the output of another instance of the

same sub-graph as a priori information. This will lead to convergence to a wrong point.

Second condition is that all instances of a sub-graph should share the same state. An instance which is invoked later, should use the state which is determined by another instance which was invoked previously. If this condition is violated, the result will be the same as in the first condition.

CHAPTER 5

APPLICATIONS AND NUMERICAL RESULTS

We will present different applications of sub-graph sum-product algorithm and numerical simulation results in this chapter. The possible applications of sub-graph sum-product algorithm are not restricted to the applications that will be presented in this chapter. The sub-graph sum-product algorithm is a so general algorithm that it can be applied to any problem that can be represented by a factor graph.

This chapter starts with showing that the well known iterative sum-product and turbo decoding algorithms are instances of sub-graph sum-product algorithm. After these sections we will present decoding of LDPC codes with different forms of our sub-graph sum-product algorithm. Finally we will apply sub-graph sum-product algorithm to a joint equalization-decoding scheme.

5.1 Turbo Decoding Algorithm as an Instance of Sub-Graph Sum-Product Algorithm

Turbo codes, which are discovered by Berrou et. al. at 1993 [4], is a very big step in coding theory. Turbo codes have showed that near Shannon

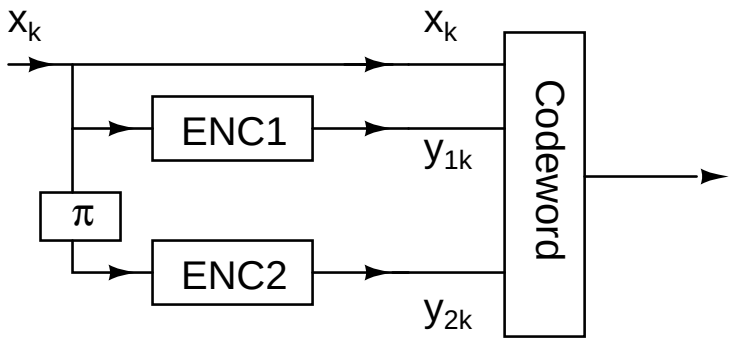


Figure 5.1: Block diagram of a parallel concatenated turbo encoder

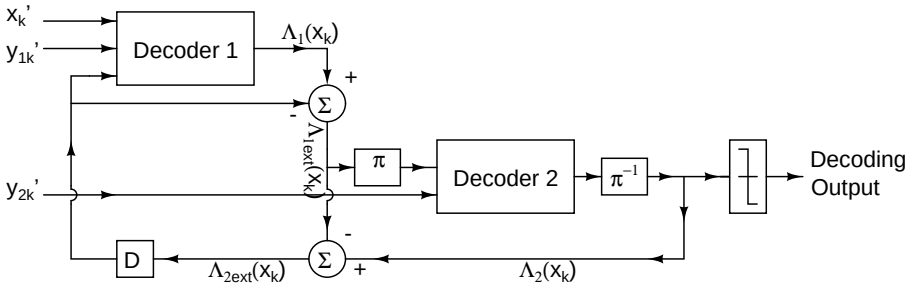


Figure 5.2: Block diagram of a parallel concatenated turbo decoder. (Delay element, which is shown by D in the figure, represents an iteration of the decoder.)

limit coding is possible with concatenating two convolutional encoders and a simple iterative decoder.

Figure 5.1 depicts the block diagram of the encoder of a parallel concatenated turbo code. Input symbols, x_k's, are first fed to a convolutional encoder, $ENC1$, and y_{1k}'s are obtained. Input symbols are also fed to a second convolutional encoder, $ENC2$, through an interleaver π and y_{2k}'s are obtained. Then these three symbol streams are combined, probably with some puncturing, and the codeword is obtained.

In the turbo decoder literature usually log-likelihood ratios are used. Log-likelihood ratio of x_k, $\Lambda(x_k)$, is defined as:

$$\Lambda(x_k) = \log \frac{Pr(x_k = 1)}{Pr(x_k = 0)} \tag{5.1}$$

Figure 5.2 shows the decoder of a parallel concatenated turbo code. As it can be seen from the figure, turbo decoder consists of two separate decoders that exchange log-likelihood ratios between them iteratively. Decoder 1 calculates $\Lambda_1(x_k)$'s using x'_k's, y'_{1k}'s, and $\Lambda_{2ext}(x_k)$'s. The symbols x'_k's and y'_{1k}'s are the received symbols when x_k's and y_{1k}'s are passed through a memoryless channel. These received symbols supply the channel information to Decoder 1. $\Lambda_{2ext}(x_k)$'s are the information generated by Decoder 2 and is used as a priori information by Decoder 1. Berrou et. al. showed that $\Lambda_1(x_k)$ is addition of three terms [4]. First term is information generated by the Decoder 1 using the channel information (x'_k and y'_{1k}'s) supplied to it. Second term is the information generated by using the $\Lambda_{2ext}(x_i)$'s, such that $i \neq k$, through the structure of the code. Final term is directly the $\Lambda_{2ext}(x_k)$. First term is common in every decoder. Second term enhances the performance of the decoder a lot. However, final term makes the decoder to converge to a wrong point. Therefore it should be subtracted as in Equation (5.2).

$$\Lambda_{1ext}(x_k) = \Lambda_1(x_k) - D(\Lambda_{2ext}(x_k)) \tag{5.2}$$

$\Lambda_{1ext}(x_k)$ is called *extrinsic* information generated by Decoder 1. A similar argument is true for Decoder 2, and $\Lambda_{2ext}(x_k)$ is obtained in a similar way as in Equation (5.3).

$$\Lambda_{2ext}(x_k) = \Lambda_2(x_k) - \Lambda_{1ext}(x_k) \tag{5.3}$$

Note that there is no delay operator in Equation (5.3), however, a delay operator exists in Equation (5.2). The reason is that Decoder 1 uses the $\Lambda_{2ext}(x_k)$'s which are generated in the previous iteration.

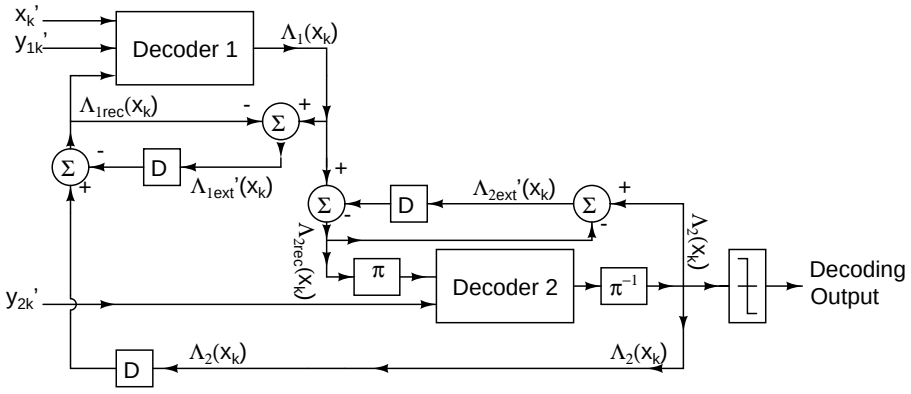


Figure 5.3: Modified turbo decoding schematic

After a brief introduction to turbo codes and turbo decoding, now we can start to show that turbo decoding is an instance of sub-graph sum-product algorithm. As a first step, we modify the schematic diagram of turbo decoding ,which is shown in Figure 5.2, as in Figure 5.3.

We shall start with showing that the decoder shown in Figure 5.3 is equivalent to the decoder shown in Figure 5.2. We have represented the outputs of the decoders $\Lambda_i(x_k)$ in both figures. We have removed the signals $\Lambda_{i,ext}(x_k)$, $i = 1, 2$, in Figure 5.3, and add new signals namely $\Lambda_{i,rec}(x_k)$ and $\Lambda'_{i,ext}(x_k))$ for $i = 1, 2$. In order to prove that these two schematic diagrams are equivalent we should show that inputs of the decoders in these two different schematic diagrams are equal. In other words, we should show that $\Lambda_{1rec}(x_k) = D(\Lambda_{2ext}(x_k))$ and $\Lambda_{2rec}(x_k) = \Lambda_{1ext}(x_k)$ for all k. We begin with two assumptions such that

$$\Lambda'_{1ext}(x_k) \stackrel{?}{=} \Lambda_{1ext}(x_k) \quad \forall k \tag{5.4}$$

$$\Lambda'_{2ext}(x_k) \stackrel{?}{=} \Lambda_{2ext}(x_k) \quad \forall k \tag{5.5}$$

Figure 5.3 depicts that

$$\Lambda_{1rec}(x_k) = D(\Lambda_2(x_k)) - D(\Lambda'_{1ext}(x_k)) \quad (5.6)$$

$$\Lambda_{1rec}(x_k) = D(\Lambda_2(x_k) - \Lambda'_{1ext}(x_k)) \quad (5.7)$$

$$\Lambda_{1rec}(x_k) = D(\Lambda_2(x_k) - \Lambda_{1ext}(x_k)) \quad (5.8)$$

$$\Lambda_{1rec}(x_k) = D(\Lambda_{2ext}(x_k)) \quad (5.9)$$

We have obtained Equation (5.8) by using the assumption given in Equation (5.4) and Equation (5.9) by using Equation (5.3). This completes the first part of the proof.

Similarly starting from the Figure 5.3 we can complete the second part of the proof.

$$\Lambda_{2rec}(x_k) = \Lambda_1(x_k) - D(\Lambda'_{2ext}(x_k)) \quad (5.10)$$

$$\Lambda_{2rec}(x_k) = \Lambda_1(x_k) - D(\Lambda_{2ext}(x_k)) \quad (5.11)$$

$$\Lambda_{2rec}(x_k) = \Lambda_{1ext}(x_k) \quad (5.12)$$

The last thing we should do before completing the proof is validating the assumptions. Figure 5.3 and 5.2 show that

$$\Lambda'_{1ext}(x_k) = \Lambda_1(x_k) - \Lambda_{1rec}(x_k) \quad (5.13)$$

$$= \Lambda_1(x_k) - D(\Lambda_{2ext}) \quad (5.14)$$

$$= \Lambda_{1ext}(x_k) \quad (5.15)$$

$$\Lambda'_{2ext}(x_k) = \Lambda_2(x_k) - \Lambda_{2rec}(x_k) \quad (5.16)$$

$$= \Lambda_2(x_k) - \Lambda_{1ext}(x_k) \quad (5.17)$$

$$= \Lambda_{2ext}(x_k) \quad (5.18)$$

After showing that Figures 5.2 and 5.3 are two different schematic representations of the same turbo decoding algorithm, we should show that

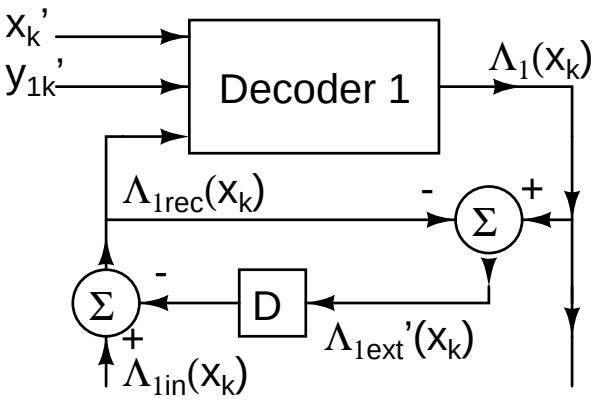


Figure 5.4: Zoomed portion of Figure 5.3

Decoder 1 and Decoder 2 are sub-graphs. We zoom to a region of Figure 5.3 that is shown in Figure 5.4.

If Figure 5.4 is inspected carefully it can be seen that each component structure of Decoder 1 together with its peripherals looks like the structure of a sub-graph block that was represented in Figure 3.4. A significant difference between two diagrams is that, vectors of probability density functions flow inside a sub-graph structure, on the other hand vectors of log-likelihood ratios flow through the edges of the turbo decoder structure. Log-likelihoods represent only a special case of random variables which is the binary random variable. However, our representation and formulation can be applicable to any discrete type of random variable including the binary random variables. Separation operation in our probability density function domain is the generalized form of the subtraction operation in the log-likelihood domain. We will prove this argument in Theorem 5.1.

Theorem 5.1. *Subtraction operation in log-likelihood domain is a special case of the separation operation in probability density function domain. In other words; let* $\mathbf{x_1}$ *be a random variable having probability density func-*

tion $f_{\mathbf{x_1}}(x_1)$ *and* $\mathbf{x_2}$ *be a random variable having probability density function* $f_{\mathbf{x_2}}(x_2)$*. If* $\mathbf{x_3}$ *is a random variable having probability density function* $f_{\mathbf{x_3}}(x_3) = f_{\mathbf{x_1}}(x_1) \oslash f_{\mathbf{x_2}}(x_2)$ *then*

$$\Lambda(x_3) = \Lambda(x_1) - \Lambda(x_2)$$

Proof: Let

$$f_{\mathbf{x_1}}(x_1) = \begin{cases} a, & x = 0 \\ b, & x = 1 \end{cases}$$

and

$$f_{\mathbf{x_2}}(x_2) = \begin{cases} \frac{1}{c}, & x = 0 \\ \frac{1}{d}, & x = 1 \end{cases}$$

Then

$$\begin{aligned} f_{\mathbf{x_3}}(x_3) &= f_{\mathbf{x_1}}(x_1) \oslash f_{\mathbf{x_2}}(x_2) \\ &= \frac{f_{\mathbf{x_1}}(x_1)/f_{\mathbf{x_2}}(x_2)}{\sum_{x=0,1} f_{\mathbf{x_1}}(x_1)/f_{\mathbf{x_2}}(x_2)} \\ &= \frac{f_{\mathbf{x_1}}(x_1)/f_{\mathbf{x_2}}(x_2)}{ac+bd} \\ &= \begin{cases} \frac{ac}{ac+bd}, & x = 0 \\ \frac{bd}{ac+bd}, & x = 1 \end{cases} \end{aligned} \tag{5.19}$$

Then

$$\begin{aligned} \Lambda(x_3) &\triangleq \log \frac{Pr(x_3 = 1)}{Pr(x_3 = 0)} \\ &= \log \frac{f_{\mathbf{x_3}}(1)}{f_{\mathbf{x_3}}(0)} \\ &= \log \frac{ac}{bd} \\ &= \log \frac{a}{b} - \log \frac{d}{c} \\ &= \Lambda(x_1) - \Lambda(x_2), \end{aligned} \tag{5.20}$$

which completes the proof.

We can easily make a one-to-one matching between the signals represented in Figure 5.4 and Figure 3.4. The signals $\Lambda_{1in}(x_k)$, $\Lambda_{1rec}(x_k)$, $\Lambda_{1ext'}(x_k)$, and $\Lambda_1(x_k)$ in Figure 5.4 can be matched to the signals $\widetilde{\mathbf{I}}$, $\widetilde{\mathbf{R}}$, $\widetilde{\mathbf{X}}$, and $\widetilde{\mathbf{O}}$ in Figure 3.4 respectively.

Up to this point we have modified the turbo decoder structure without altering its characteristics. Then we have showed that subtraction operation in log-likelihood ratio domain is a special case of separation operation in probability density function domain. We should also show that component decoders, i.e. Decoder 1 and Decoder 2, employs sum-product algorithm on a loop free factor graph. However, this is done many times in literature. Component decoders in a turbo decoder employs the BCJR algorithm [4, 3]. It is shown in literature in many times that BCJR algorithm is an instance of the sum-product algorithm on a loop-free factor graph [1, 18].

When BCJR algorithm of the component decoders is represented by a factor graph, x'_k's, y'_{1k}'s, and $y'_{2k}s$'s affect the factor functions of the factor nodes in that factor graph. When a new set of data is received by the decoder, the factor functions of the factor nodes are changed. However, structure of the factor graph remains the same.

An important detail in this structure is that x'_k's are applied to the Decoder 1 only in Figure 5.2. This means that the factor nodes corresponding to x'_k's exist only in the factor graph structure of Decoder 1. This fact complies with our definition of proper partitioning, which is given in Definition 3.2. Proper partitioning rule states that a factor node can be a node of only one sub-graph. This rule is satisfied by applying x'_k's to the Decoder 1 only. Applying the x'_k's to only Decoder 1 is not a modification to the turbo decoding algorithm we made, but it is the original form of the algorithm [4].

As a final step, representation of the turbo decoder structure with sub-

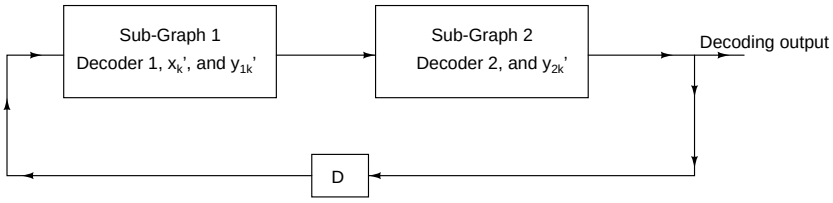


Figure 5.5: Representation of the turbo decoder structure with sub-graphs

graphs is shown in Figure 5.5. In this representation we embed the interleaver and deinterleaver blocks that exist in Figure 5.3 in the second sub-graph. As it can be seen from the Figure 5.5 that the turbo decoding algorithm is just a simple instance of the sub-graph sum-product algorithm.

In this section we have showed how the decoding algorithm of the parallel concatenated turbo codes is an instance of sub-graph sum-product algorithm. Using very similar but a little bit more complicated arguments we can show that decoding of serial concatenated turbo codes and turbo equalization algorithms are also instances of sub-graph sum-product algorithm.

It should be noted that, we did not design the sub-graph sum-product algorithm to be a generalization of the turbo decoding algorithm. While devoloping the sub-graph sum-product algorithm, we have realized that turbo decoding algorithm is an instance of the sub-graph sum-product algorithm. However, we have inspired a lot from the idea of *"extrinsic information"* during the development of the sub-graph sum-product algorithm. The idea of *"extrinsic information"* and turbo codes is a big step in coding theory. Therefore, inventors of the turbo decoding algorithm deserve a great appreciation.

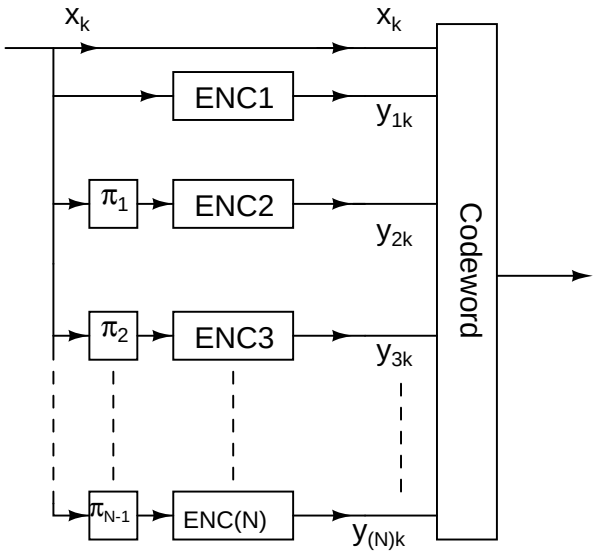


Figure 5.6: Schematic of a multiple concatenated turbo encoder

5.1.1 Possible Improvements on Turbo Codes Using Sub-Graph Sum-Product Algorithm

After putting the turbo decoding algorithm on to sub-graph sum-product algorithm basis, we can build more floors on top of the turbo decoding algorithm using the sub-graph idea. For instance, connecting the sub-graphs in parallel may be investigated.

A more interesting option may be using multiple concatenated turbo codes and decode them with the sub-graph sum-product algorithm. Multiple concatenated turbo codes are the turbo codes which consist of more than two component codes, and their encoder consists of more than two convolutional encoders as shown in Figure 5.6.

Multiple concatenated turbo codes are proposed [16] in 1995. Their decoding algorithm is different than the turbo decoding algorithm. By using the sub-graph methodology we can propose very simple decoding algorithm

for multiple concatenated turbo codes. Decoders of the component codes can be considered as sub-graphs similar to the Section 5.1. Then these sub-graphs can be connected in cascade or in parallel form to construct a decoder of multiple concatenated turbo code.

Probably the multiple concatenated turbo codes together with the decoder that we propose in the previous paragraph, will perform better than the standard turbo codes of the same rate. A standard turbo decoder has a factor graph structure composed of two BCJR factor graphs connected through an interleaver. On the other hand, a multiple concatenated turbo code composed of N component codes has a factor graph structure consisting of N BCJR factor graphs connected through $N-1$ interleavers. In order to preserve the code rate, the multiple concatenated turbo code should be punctured more than a standard turbo code. Puncturing corresponds to removing some of the factor nodes in the BCJR factor graphs, consequently breaking some loops or making the length of some loops longer. In other words the factor graph structure of a multiple concatenated turbo decoder contains less number of loops which are longer than the loops on standard turbo decoder. This fact will definitely increase the performance of the multiple concatenated code.

The random coding idea may be an other way of showing that performance of multiple concatenated turbo codes is better than the standard turbo codes. The output samples of a more punctured multiple concatenated turbo code are obviously less correlated than the output samples of the standard turbo code. Being less correlated means being more random, hence a better code.

The turbo codes are already very good codes which approach the Shannon limit. Therefore, multiple concatenated turbo codes are not expected to

improve BER-SNR characteristics of the codes. However, multiple concatenated turbo codes may improve the error floor characteristics of the turbo codes.

This sub-section is a future research proposal. Verifying the arguments in this subsection experimentally, is out of the scope of this thesis. If the multiple concatenated turbo codes together with the sub-graph sum product algorithm lowers the error floor of the turbo codes then the gap between turbo codes and LDPC codes will be closed one step further.

5.2 Iterative Sum-Product Algorithm as an Instance of Sub-Graph Sum-Product Algorithm

The iterative sum-product algorithm can be considered as an instance of the sub-graph sum-product algorithm. Proving this idea is much more simpler than showing that turbo decoding algorithm is an instance of the sub-graph sum-product algorithm.

One of the most important properties of the iterative sum-product algorithm is every factor node operates in parallel. If a sub-graph sum-product algorithm is constructed with a parallel connection of non-accessible sub-graphs, then all of the factor nodes in the sub-graph sum-product algorithm may also operate in parallel. Therefore parallel connection of non-accessible sub-graphs seems to be a candidate for the equivalent of the iterative sum-product algorithm.

Let v_i be the i^{th} variable node in a factor graph which contain loops. Remember from Section 2.2 that M_i represents the set of the indices of the

neighbors of v_i. Assume that iterative sum-product algorithm is running on this factor graph. The message function from v_i to p_j, where $j \in M_i$, at the t^{th} iteration is given by

$$m_{i \to j,\ t}(x_i) = \prod_{k \in M_i \setminus \{j\}} n_{k \to i, t-1}(x_i) \tag{5.21}$$

Now we shall calculate the same message in the sub-graph sum-product algorithm which has a flow graph of parallel connection of non-accessible sub-graphs. Since sub-graphs are non-accessible all of the neighbors of v_i lie in different sub-graphs. During the $(k-1)^{th}$ iteration all of these factor node neighbors send their messages to the instances of v_i. Each one of the sub-graphs store a copy of this message in their state. Then these messages are combined at the junction point of the flow graph. Let $\bar{m}_{i,k-1}(x_i)$ be the message function about the i^{th} variable node after the combination operation. Then $\bar{m}_{i,k-1}(x_i)$ is given by:

$$\bar{m}_{i,k-1}(x_i) = \frac{\prod_{k \in M_i} n_{k \to i, t-1}(x_i)}{\sum_{\forall x_i} \prod_{k \in M_i} n_{k \to i, t-1}(x_i)} \tag{5.22}$$

$$= \frac{1}{Z_1} \prod_{k \in M_i} n_{k \to i, t-1}(x_i) \tag{5.23}$$

The function $\bar{m}_{i,k-1}(x_i)$ will be distributed to each sub-graph. Then inside the sub-graphs, each sub-graph will separate the message function which they hold in their states from $\bar{m}_{i,k-1}(x_i)$. Remember that separation operation is nothing but a normalized point by point division. Result of the separation operation will determine the message that factor nodes receive from i^{th} variable node. For instance, for the j^{th} factor node it is given by the following equations.

$$m_{i \to j,\ t}(x_i) = \frac{\bar{m}_{i,k-1}(x_i) / n_{j \to i, k-1}(x_i)}{\sum_{\forall x_i} \bar{m}_{i,k-1}(x_i) / n_{j \to i, k-1}(x_i)} \tag{5.24}$$

$$= \frac{1}{Z_1} \frac{1}{Z_2} \prod_{k \in M_i \setminus \{j\}} n_{k \to i, t-1}(x_i) \tag{5.25}$$

The only difference between two message functions which are calculated in Equation (5.21) and Equation (5.25) is the normalization constants Z_1 and Z_2. It is not a big problem since these normalization constants will be added to the results of the iterative sum-product algorithm at the finalization step. Therefore, the messages calculated in Equation (5.21) and Equation (5.25) can be considered as equivalent. Hence, iterative sum product algorithm is equivalent to an instance of the sub-graph sum-product algorithm which is constructed by parallel connection of non-accessible sub-graphs.

5.3 Different LDPC Decoding Strategies with Sub-Graph Sum-Product Algorithm and Simulation Results

The iterative sum-product algorithm is the default LDPC decoding algorithm in literature. We are able to propose different LDPC decoding algorithms by the help of sub-graph approach. There exists thousands of partitioning choices for a factor graph of a given LDPC code. Moreover, for the chosen partitioning there exists again hundreds of different types of flow graphs. Analyzing all of these different choices is almost impossible. Therefore, we have conducted our simulations with the extreme types of LDPC decoders. As mentioned in Chapter 4, there are two extreme types of sub-graph partitioning, namely all-accessible and non-accessible. In terms flow graphs, we have only analyzed the all-parallel and all-cascade connected sub-graphs. As a result we have constructed and simulated four different kinds of LDPC decoders. Our first decoder is composed of parallel connection of non-accessible sub-graphs. Note that this decoder is equivalent to the standard LDPC de-

coder as shown in Section 5.2. Second decoder is the cascade connection of non-accessible sub-graphs. Our third and fourth decoders are cascade and parallel connection of all-accessible sub-graphs. In other words we simulate and compare *three new LDPC decoding algorithms* with the standard LDPC decoding algorithm.

We have used a randomly generated LDPC code of length 2000 in our simulations. The code is a regular LDPC code, having row weight 6 and column weight 3. Hence, the code rate is $R = \frac{1}{2}$. We have used the same code for all of the four decoders.

All of the four decoders are allowed to make at most 200 iterations. If all of the parity check equations are satisfied before achieving 200 iterations then the iteration process is terminated. The resulting decoded codeword is compared with the transmitted codeword. If the decoded codeword was not equal to the transmitted codeword then it would mean that the decoder made an undetected error. However, we did not observed any undetected errors during our simulations for all of the four decoders. All of the errors were detected in which the decoder reaches 200^{th} iteration.

Encoding of LDPC codes is a difficult and separate research problem. Therefore, encoders are not used while conducting a research about the performance of LDPC codes. Instead it is assumed that an all zero codeword is transmitted, which guarantees that all of the parity check equations are satisfied. Since LDPC codes are linear codes, probability of error when the all zero codeword is transmitted, is equal to the probability of error when any codeword is transmitted. We have adopted this approach and always used the all zero codeword.

During the rest of the thesis, we have used abbreviated names for decoders. We have used the abbreviations "PAR-NON","CAS-NON","PAR-

ALL", and "CAS-ALL" for the decoders parallel connection of non-accessible sub-graphs, cascade connection of non-accessible sub-graphs, parallel connection of all-accessible sub-graphs, and cascade connection of all-accessible sub-graphs respectively.

We have modulated coded symbols with binary phase shift keying (BPSK) modulation and transmit them through additive white Gaussian noise (AWGN) channel.

In Figure 5.7 we have presented frame error rate (FER) vs. signal to noise ratio (SNR) characteristics of all these four decoders. This figure shows that the decoders which consist of non-accessible sub-graphs perform approximately $2dB$ better than the decoders which consist of all-accessible sub-graphs. Connection type of the sub-graphs does not affect the FER-SNR characteristics much. For a deeper inspection, FER-SNR characteristics of the all-accessible and non-accessible decoders are drawn separately in Figures 5.8 and 5.9 respectively. Both of these figures show that decoders constructed by connecting the sub-graphs in cascade are slightly better than the decoders which are constructed by connecting the sub-graphs in parallel. This result is quite important since one of the new decoders, CAS-NON, is a little bit better than the standard LDPC decoding algorithm which is the PAR-NON.

Figure 5.10 depicts the average number of iterations required to converge for each decoder. While calculating these averages only the correctly decoded codewords are taken into account. Firstly, this figure shows that limiting the number of iterations to 200 for this code is reasonable since at the lowest SNR in which decoder can receive some correct codewords the average number of iterations is at most 40 for all decoders. The second result that can be deduced from this figure is that the convergence rate of the decoders in which sub-graphs are connected in cascade is higher. The number of average

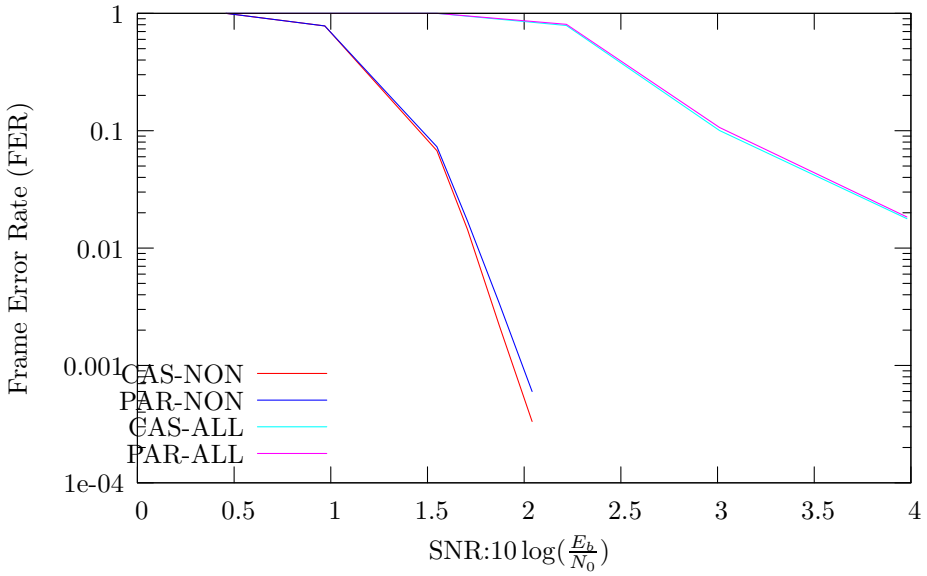


Figure 5.7: FER. vs. SNR characteristics of all of the four decoders.

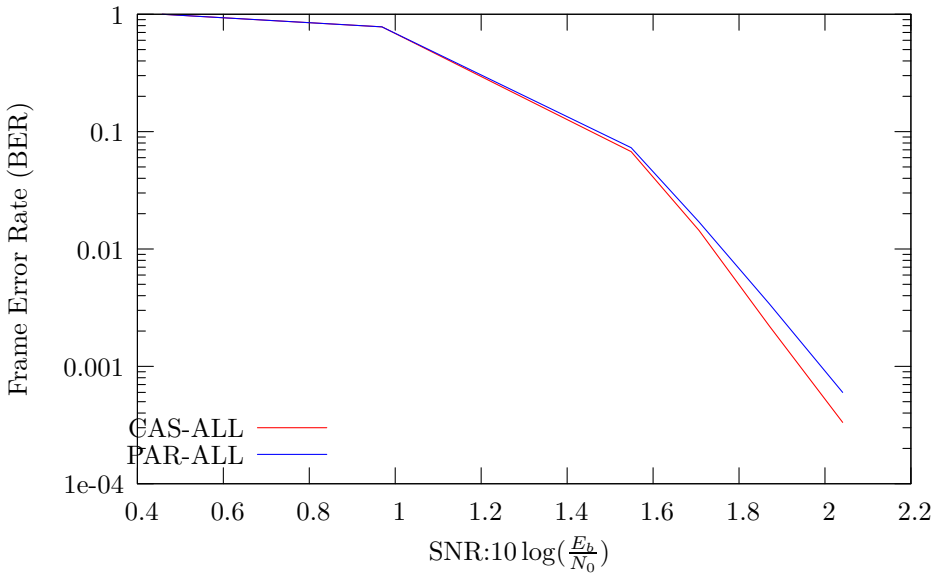


Figure 5.8: FER. vs. SNR characteristics of the non-accessible decoders.

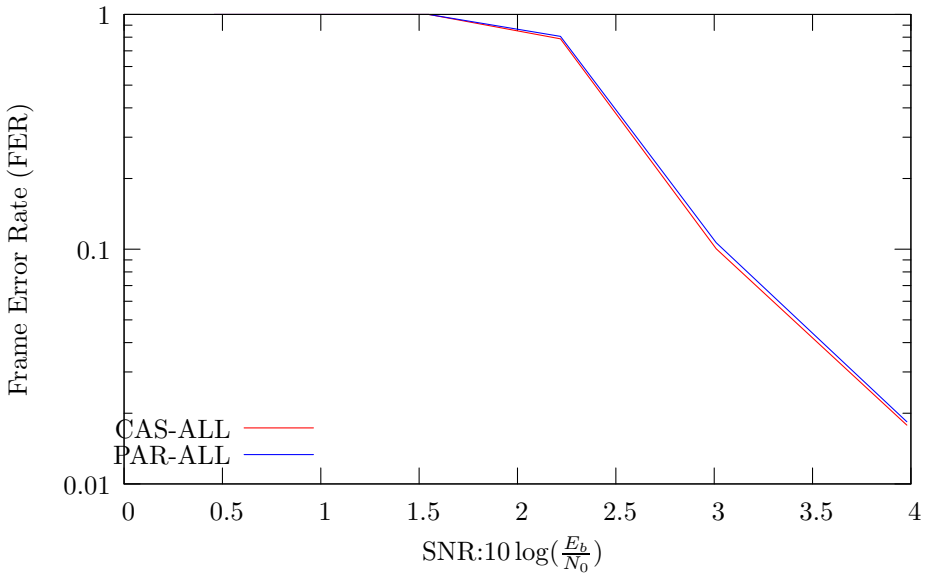


Figure 5.9: FER. vs. SNR characteristics of the all-accessible decoders.

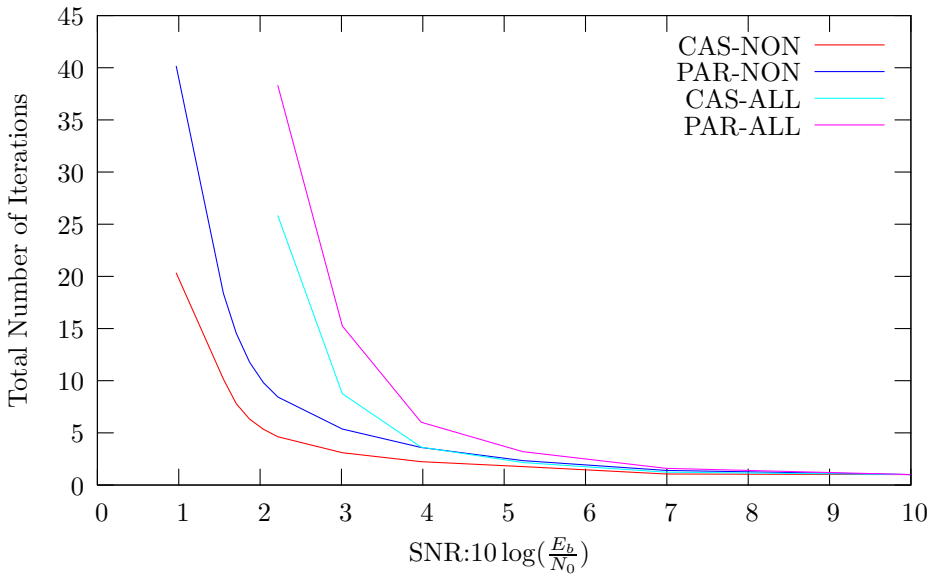


Figure 5.10: Average number of iterations done by decoders at different SNR's.

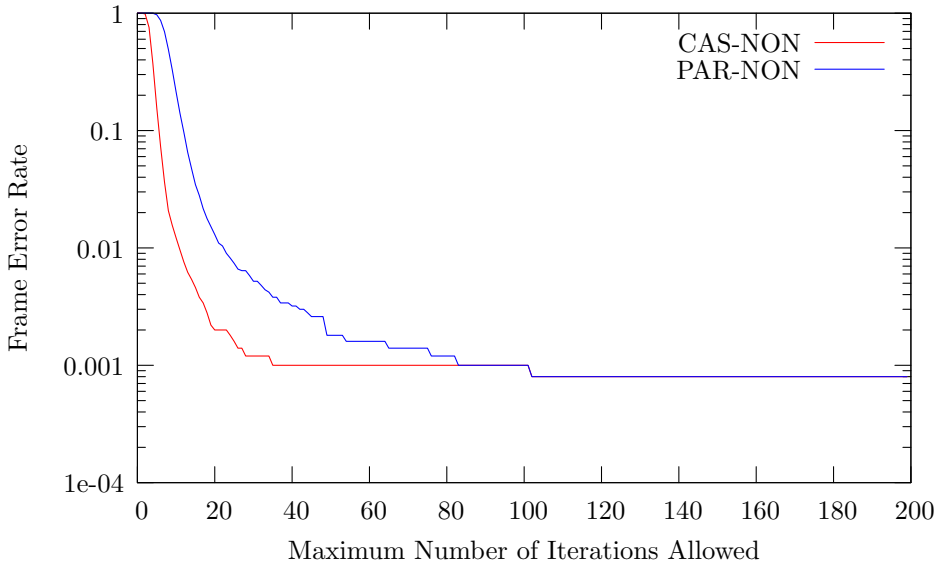


Figure 5.11: FER characteristics of the non-accessible decoders for different maximum allowed number of iterations at $2db$.

iterations required to converge for CAS-NON decoder is nearly half of the standard (PAR-NON) decoder. Figures 5.11 and 5.12 also supports this fact. These figures show the performance of non-accessible decoders at $2dB$ and all-accessible decoders at $4dB$ w.r.t maximum number of iterations allowed. This is a very important result when combined with FER-SNR characteristics. The CAS-NON decoder convergence twice as fast as the PAR-NON (standard) decoder without altering the FER-SNR characteristics.

5.4 Joint Decoding and Equalization with Sub-Graph Sum-Product Algorithm

Another problem that we have applied our sub-graph sum-product algorithm is the joint decoding and equalization problem. This section starts with an

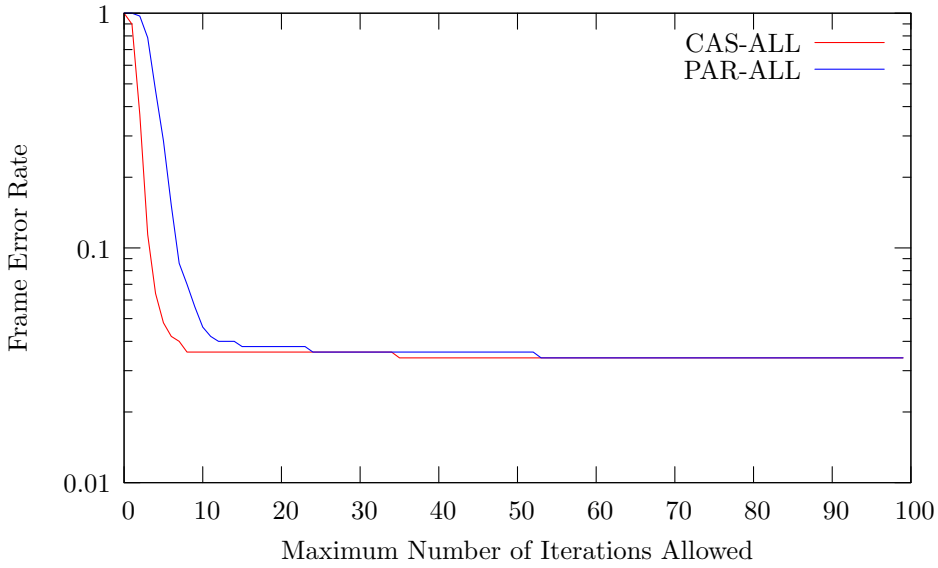


Figure 5.12: FER characteristics of the all-accessible decoders for different maximum allowed number of iterations at $4db$.

introduction to obtaining factor graph for a signal transmission through an inter-symbol interference (ISI) channel.

5.4.1 Factor Graphs of ISI Channels

Assume that an independent identically distributed (i.i.d.) equally likely sequence of elements of a symbol set A, $(x_0, x_1, \ldots, x_n)$, is passed through a discrete time channel with memory. The impulse response of the channel is represented with $(h_0, h_1, \ldots, h_{L-1})$ and $h_i = 0$ for $i < 0, i \geq L$. White Gaussian noise is added to the samples, $\mathbf{n}_i$, at this stage and received symbols y_i's are obtained. In other words

$$y_i = \sum_{k=0}^{L-1} x_{i-k} h_k + n_i \tag{5.26}$$

Finding the marginal a posteriori probability densities of x_i's given y_i's, i.e. $f_{\mathbf{x_i}}(x|y_0, y_1, \ldots, y_{L+n-1})$, is a problem which can be represented with factor graphs. $f_{\mathbf{x_i}}(x|y_0, y_1, \ldots, y_{L+n-1})$ can be obtained from the joint a posteriori density function of x_i's through the marginalization sum as in Equation (5.27).

$$f_{\mathbf{x_i}}(x|y_0, y_1, \ldots, y_{L+n}) = \sum_{\forall(x_0,x_1,\ldots,x_n)|x_i=x} f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0, \ldots, x_n|y_0, \ldots, y_{n+L-1}) \quad (5.27)$$

In order to show that this problem can be represented with factor graphs, we should show that the joint a posteriori density function $f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0, x_1, \ldots, x_n|y_0, y_1, \ldots, y_{L+n-1})$ can be factorized into functions which have smaller number of x_i's as arguments.

Using Bayes Theorem we can obtain

$$f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0, x_1, \ldots, x_n|y_0, y_1, \ldots, y_{L+n-1}) = f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0, x_1, \ldots, x_n) f_{\mathbf{y_0},\mathbf{y_1},\ldots,\mathbf{y_{L+n-1}}}(y_0, y_1, \ldots, y_{L+n-1}|x_0, x_1, \ldots, x_n) \cdot \frac{1}{f_{\mathbf{y_0},\mathbf{y_1},\ldots,\mathbf{y_{L+n-1}}}(y_0, y_1, \ldots, y_{L+n-1})} \quad (5.28)$$

For a given $(y_0, y_1, \ldots, y_{L+n-1})$, $f_{\mathbf{y_0},\mathbf{y_1},\ldots,\mathbf{y_{L+n-1}}}(y_0, y_1, \ldots, y_{L+n-1})$ is constant. Moreover, since $(x_0, x_1, \ldots, x_n)$ is an i.i.d and equally likely sequence $f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0, x_1, \ldots, x_n)$ is constant for any $(x_0, x_1, \ldots, x_n)$. These facts simplify the Equation (5.28) as below:

$$f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0, x_1, \ldots, x_n|y_0, y_1, \ldots, y_{L+n-1}) = \frac{1}{Z} f_{\mathbf{y_0},\mathbf{y_1},\ldots,\mathbf{y_{L+n-1}}}(y_0, y_1, \ldots, y_{L+n-1}|x_0, x_1, \ldots, x_n) \quad (5.29)$$

y_i's are conditionally independent given $x_0, x_1, \ldots, x_n$. Therefore, Equation

(5.29) can be simplified further as follows:

$$f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0,x_1,\ldots,x_n|y_0,y_1,\ldots,y_{L+n-1}) = \frac{1}{Z}\prod_{i=0}^{L+n-1} f_{\mathbf{y_i}}(y_i|x_0,x_1,\ldots,x_n) \quad (5.30)$$

A received sample y_i depends only the past L transmitted samples. Therefore Equation (5.30) can be rewritten as below:

$$f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0,x_1,\ldots,x_n|y_0,y_1,\ldots,y_{L+n-1}) = \frac{1}{Z}\prod_{i=0}^{L+n-1} f_{\mathbf{y_i}}(y_i|x_{i-L+1},x_{i-L+2},\ldots,x_i) \quad (5.31)$$

For a given received symbol y_i, $f_{\mathbf{y_i}}(y_i|x_{i-L+1},x_{i-L+2},\ldots,x_i)$ is a function of $x_{i-L+1},x_{i-L+2},\ldots,x_i$. If this function is denoted by g_i, Equation (5.31) can be written as

$$f_{\mathbf{x_0},\mathbf{x_1},\ldots,\mathbf{x_n}}(x_0,x_1,\ldots,x_n|y_0,y_1,\ldots,y_{L+n-1}) = \frac{1}{Z}\prod_{i=0}^{L+n-1} g_i(x_{i-L+1},x_{i-L+2},\ldots,x_i) \quad (5.32)$$

We have been written the joint a posteriori density function as a product of $L+n$ functions which have at most L arguments by Equation (5.32). Therefore, now we can represent this problem with a factor graph. The factor graph of this problem, consists of n variable nodes, one for each x_i and $L+n$ factor nodes one for each y_i where each factor function $g_i(x_{i-L+1},x_{i-L+2},\ldots,x_i)$ can be calculated by Equation (5.33).

$$g_i(x_{i-L+1},x_{i-L+2},\ldots,x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\sum_{j=0}^{L-1} x_{i-j}h_j)^2}{2\sigma^2}} \quad (5.33)$$

Where σ is the variance of $\mathbf{n}_i$.

Example 5.1. An ISI factor graph: Figure 5.13 shows an example ISI factor graph. In this factor graph the channel has length 3 and block length is 7 symbols. Different possible partitioninigs of the factor graph are also shown in Figure 5.13

Exact Message Passing

After deriving the factor functions, exact message functions can be calculated. In this section we will present the expressions of the exact message functions. We assume that i^{th} variable node represents the i^{th} transmitted symbol, x_i, and the j_{th} factor node represents the factor function $g_j(x_{j-L+1}, x_{j-L+2}, \ldots, x_j)$.

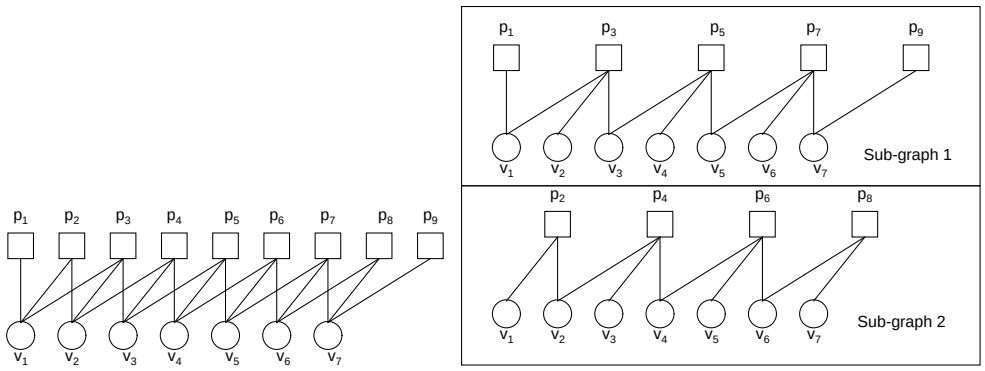
We start with the messages from variable nodes to factor nodes. Due to the structure of the factor graph the i^{th} variable node is connected to only $i^{th}, (i+1)^{th}, \ldots, (i+L-1)^{th}$ factor nodes. If $j \in \{i, i+1, \ldots, i+L-1\}$ the message function from i^{th} variable node to the j^{th} factor node is given by the expression

$$m_{i\to j}(x) = \prod_{k\in\{i,i+1,\ldots,i+L-1\}\setminus\{j\}} n_{k\to i}(x). \tag{5.34}$$

Similarly, j^{th} factor node is connected to the $j^{th}, (j-1)^{th}, \ldots, (j-L+1)^{th}$ variable nodes. If $i \in \{j, j-1, \ldots, j-L+1\}$ then the message function from j^{th} factor node to the i^{th} variable node is given by the expression

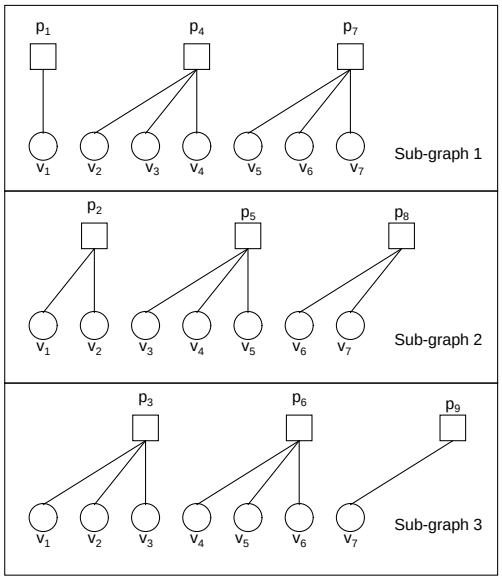
$$n_{j\to i}(x) = \sum_{\forall(x_{j-L+1},x_{j-L+2},\ldots,x_j)|x_i=x} g_j(x_{j-L+1}, x_{j-L+2}, \ldots, x_j). \prod_{k\in\{j-L+1,j-L+2,\ldots,j\}\setminus\{i\}} m_{k\to j}(x) \tag{5.35}$$

Expressions given in Equations (5.34) and (5.35) are direct applications of the sum-product algorithm defined in Definition 2.2.



(a) The factor graph

(b) All-accessible partitioning



(c) Non-accessible partitioning

Figure 5.13: An ISI factor graph corresponding to a channel of length 3, and possible sub-graph partitionings

5.4.2 An Approximation to Message Calculation for ISI Factor-Graphs

In this section we introduce a message calculation simplification in ISI factor graphs. Message calculation in variable nodes has complexity $O(|A|L)$ as can be ssen from Equation (5.34). However, message calculation at the factor nodes has complexity $O(|A|^L)$ due to the marginalization sum in the Equation (5.35). In order to decrease the total complexity, complexity of the message calculation at the factor nodes should be decreased. Therefore, we make an approximation for message calculation at the factor nodes.

We will show the derivation of the approximation for only sending a message to the L^{th} neighbor of an ISI factor node, for the sake of simplicity in the notation. However, our derivation is general, and can be applied to any neighbor.

Remember the factor function $g_i(x_{i-L+1}, x_{i-L+2}, \ldots, x_i)$ that we have derived in Equation (5.33). Note that this function is also a function of a linear combination of its arguments as shown in Equation (5.36). In Equation (5.37) we simplify the argument of the function g_i.

$$\begin{aligned} g_i(x_{i-L+1}, x_{i-L+2}, \ldots, x_i) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\sum_{j=0}^{L-1} x_{i-j}h_j)^2}{2\sigma^2}} \\ &= g_i(\sum_{j=0}^{L-1} x_{i-j}h_j) \qquad (5.36) \\ &= g_i(r) \qquad (5.37) \end{aligned}$$

Now let's calculate the following convolution:

$$
\begin{aligned}
g_i(r) * m_{i-L+1\to i}(\frac{r}{h_{L-1}}) * m_{i-L+2\to i}(\frac{r}{h_{L-2}}) * \cdots * m_{i-1\to i}(\frac{r}{h_1}) = \\
(\sum_{\forall(x_{i-L+1})} g_i(r - h_{L-1}x_{i-L+1}) m_{0\to i}(x_{i-L+1})) \\
* m_{i-L+2\to i}(\frac{r}{h_{L-2}}) * \cdots * m_{i-1\to i}(\frac{r}{h_1}) = \\
(\sum_{\forall(x_{i-L+1},x_{i-L+2})} g_i(r - h_{L-1}x_{i-L+1} - h_{L-2}x_{i-L+2}) \\
m_{i-L+1\to i}(x_{i-L+1}) m_{i-L+2\to i}(x_{i-L+2})) \\
* m_{i-L+3\to i}(\frac{r}{h_{L-3}}) * \cdots * m_{i-1\to i}(\frac{r}{h_1}) = \\
\vdots \\
= \sum_{\forall(x_{i-L+1},x_{i-L+2},\ldots,x_{i-1})} (g_i(r - \sum_{j=1}^{L-1} x_{i-j}h_j) \\
\prod_{j=1}^{L-1} m_{i-j\to i}(x_{i-j})) \quad (5.38)
\end{aligned}
$$

Now if we replace r in the equation above with $-x\ h_0$ we can obtain the message function $n_{i\to i}(x)$ which is defined in Equation (5.35).

$$
\begin{aligned}
n_{i\to i}(x) = \sum_{\forall(x_{i-L+1},x_{i-L+2},\ldots,x_i)|x_i=x} (g(-\sum_{j=0}^{L-1} x_{i-j}h_j) \\
\prod_{j=1}^{L-1} m_{j\to i}(x_j)) \quad (5.39)
\end{aligned}
$$

We have seen in these preceding two equations that a factor node of an ISI factor graph actually convolves the message function it receives with its factor function. The summation operator in front of Equation (5.39) causes the complexity to be $O(|A|^L)$. If we had a simpler method for calculating this convolution, complexity of the message calculation would decrease. We will propose a simple approximation which simplifies obtaining this convolution.

Let $w(r)$ be the result of the convolution that we obtain in Equation (5.38), i.e.:

$$w(r) = \sum_{\forall(x_{i-L+1}, x_{i-L+2}, \ldots, x_{i-1})} (g_i(r - \sum_{j=1}^{L-1} x_{i-j}h_j) \prod_{j=1}^{L-1} m_{i-j\to i}(x_{i-j})) \quad (5.40)$$

Let's make a simple approximation at this step. Assume that $w(r)$ has a Gaussian density form, i.e.:

$$w(r) \approx \frac{1}{\sqrt{(2\pi\sigma_w^2)}} e^{-\frac{(r-\mu_w)^2}{2\sigma_w^2}} \quad (5.41)$$

Making such an approximation is reasonable. Because, we have obtained $w(r)$ by convolving some probability density functions ($m_{i-j\to i}(r/h_j)$'s) and a Gaussian density function ($g(r)$). Central limit theorem states that if these density functions belong to independent random variables -we assume to be so- $w(r)$ will converge to a Gaussian density function.

Just knowing σ_w^2 and μ_w is enough for specifying $w(r)$. Calculating these two parameters is easy. Convolution in probability density functions (p.d.f.) domain corresponds to addition in the random variable domain. Therefore, $w(r)$ is the p.d.f. of addition of some random variables. The first one of these random variables is the Gaussian random variable with p.d.f. $g(r)$. This random variable has mean y and variance σ^2 as can be seen from Equations (5.36) and (5.37). We convolve $g(r)$ with density functions $m_{i-j\to i}(r/h_j)$'s. These p.d.f.'s correspond to random variables $h_j\mathbf{x}_{i-j}$. After identifying random variables whose p.d.f.'s are convolved, then we can easily calculate the parameters specifying $w(r)$. From probability theory we know that, when two random variables are added their means and variances are added, and when a random variable is multiplied with a scalar its mean is multiplied

with that scalar and its variance is multiplied with the square of that scalar. From this basic information, we can calculate σ_w^2 and μ_w as follows:

$$\sigma_w^2 = \sigma^2 + \sum_{j=1}^{L-1} h_j^2 \sigma_{i-j}^2 \tag{5.42}$$

$$\mu_w = y - \sum_{j=1}^{L-1} h_j \mu_{i-j} \tag{5.43}$$

where μ_{i-j} and σ_{i-j}^2 are the mean and variances of $\mathbf{x}_{i-j}$ calculated using $m_{i-j\to i}(x_{i-j})$ as their p.d.f. as in Equations (5.44) and (5.45).

$$\mu_{i-j} = \sum_{\forall x \in A} x \; m_{i-j\to i}(x) \tag{5.44}$$

$$\sigma_{i-j}^2 = \sum_{\forall x \in A} (x - \mu_{i-j})^2 m_{i-j\to i}(x) \tag{5.45}$$

Now calculating the message function that will be sent to L^{th} neighbor is easy.

$$n_{i\to i}(x_i) = w(-h_0 x_i)$$

We have showed the approximation in the above derivations for the L^{th} neighbor only for the sake of simplicity. Our derivation is general. For calculating the message for $L-k^{th}$ neighbor expressions can be generalized as follows:

$$\sigma_w^2 = \sigma^2 + \sum_{j\in\{0,1,\ldots L-1\}\setminus\{k\}} h_j^2 \sigma_{i-j}^2 \tag{5.46}$$

$$\mu_w = y - \sum_{j\in\{0,1,\ldots L-1\}\setminus\{k\}} h_j \mu_{i-j} \tag{5.47}$$

and

$$n_{i\to i-k}(x_{i-k}) = w(-h_k x_{i-k}). \tag{5.48}$$

Means μ_{i-j}'s and variances σ_{i-j}^2 are calculated as in Equations (5.44) and (5.45) again.

In this new, approximate method we got rid of the marginalization sum that exist in Equation (5.38). Instead, we should calculate $L-1$ means and variances. Calculating mean and variance of discrete random variable, given its p.d.f. has complexity $O(|A|)$ where $|A|$ is the alphabet size. In this procedure we do this $L-1$ times for calculating a message. Therefore complexity reduces to $O(|A|L)$ from $O(|A|^L)$ which is an important reduction.

However, one should not forget that we made this approximation based on an assumption. We have assumed that $w(r)$ is almost Gaussian, which requires that $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are independent. Therefore, this approximation could not be used in a pure ISI factor graph, since ISI factor graphs contain short cycles. However, this approximation may be useful in a joint equalization-decoding scheme, in which dependency of the neighbors of the same factor node reduces due to existence of the code factor graph.

5.4.3 Simulation Results

Our aim in these simulations was not implementing an optimum joint decoding-equalization scheme or finding the best code for a given channel. Such studies exist in literature. Our main aim was evaluating our approximation for ISI factor graphs.

While simulating joint equalization-decoding schemes, our first observation was that the regular LDPC codes having column weight 3 were not as successful as in the pure decoding problem. We have observed that the codes having average column weights around 2.5 were more successful. Therefore, we have used a code of length 1000 and average column weight 2.5 and rate $R = \frac{1}{2}$. The code used in the simulation is randomly generated. However, in order to make the whole factor graph 4-cycle free, we have prevented the variable nodes which are connected to same ISI factor node to be connected

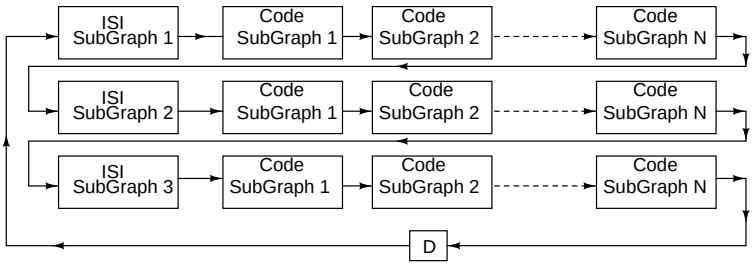


Figure 5.14: The flow graph of the joint decoder-equalizer

to same parity check factor node.

We have modulated the symbols with the binary phase shift keying (BPSK) modulation. The channel we have used in the simulations has three taps. The channel coefficients are $(0.408, 0.817, 0.408)$.

We have used a non-homogeneous flow graph structure for joint decoding-equalization problem. The flow-graph we have used is depicted in Figure 5.14. Both ISI and code sub-graphs are non-accessible type in order to obtain better performance. We have simulated also a turbo equalizer using the same LDPC code as its code in order to use it as a benchmark. The turbo equalizer employs BCJR algorithm as its equalizer and cascade-connected sub-graph sum-product algorithm as its decoder with same sub-graph partitioning. The flow graph that is shown in Figure 5.14 has made 9 iterations during the simulations. If the flow graph makes 9 iterations, each code sub-graph is invoked 27 times. In order to make fair comparison between the benchmark turbo equalizer, turbo equalizer conducted 27 iterations.

Figure 5.4.3 shows the simulation results of our approximate message passing algorithm. On the $\sim 10^{-3}$ frame error rate level, which corresponds to $\sim 10^{-5}$ bit error rate level, our approximate message passing algorithm has a performance very similar to the exact message passing algorithm. Both of the algorithms are around $0.3dB$ worse than turbo equalizer algorithm which

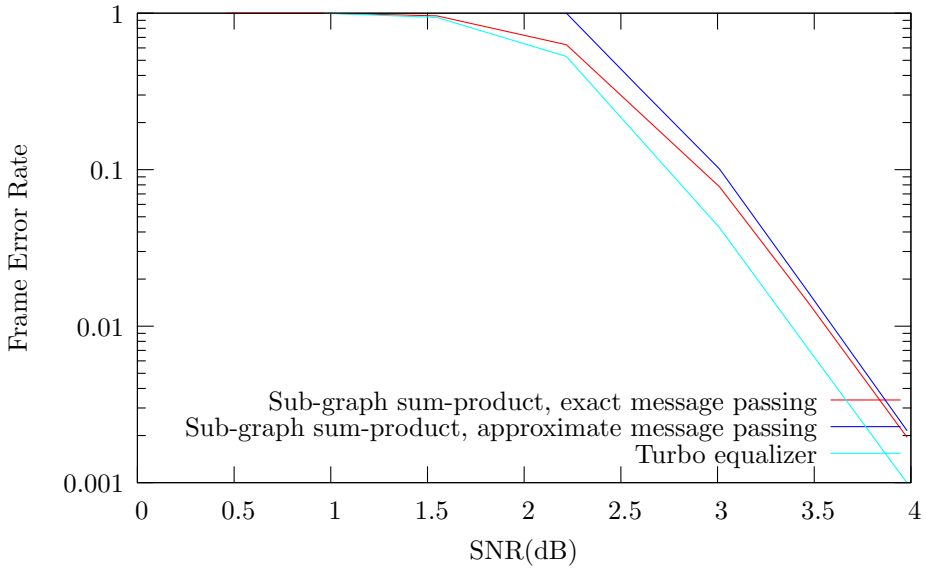


Figure 5.15: Frame error rate of three different joint decoder-equalizers

employs the optimum equalizer with complexity $O(A^L)$. These results show that our approximation works well when used with a code.

CHAPTER 6

DISCUSSION AND CONCLUSION

In this chapter we will summarize and try to explain some of the results that we have obtained in the thesis. We should note that, ideas that we will present in this chapter are just conjectures, they are not proven to be facts yet.

6.1 Why Convergence Rate Increases?

We have shown in Section 5.3 that CAS-NON decoder converges in less number of iterations than the standard LDPC decoder. In this section, this fact will be explained qualitatively.

If there exists a path between a factor node and a variable node in a factor graph, then the factor node has some information about the variable node. For instance there is a path between p_1 and v_3 in Figure 6.1. This

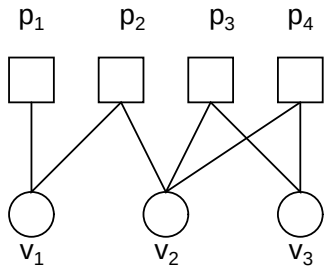


Figure 6.1: A simple factor graph with a loop

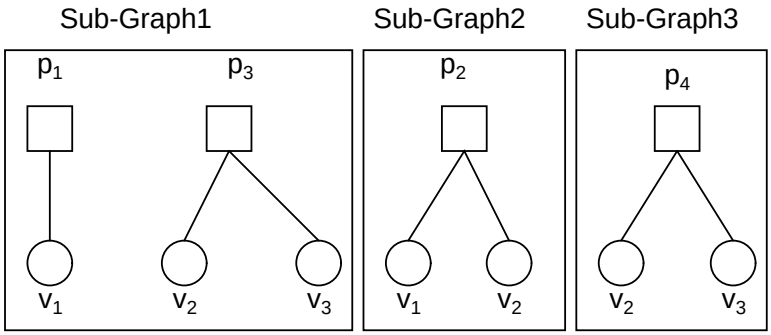


Figure 6.2: Partitioned form of the factor graph that is shown in Figure 6.1

means that p_1 has some valuable information for v_3. If iterative sum-product algorithm runs on this factor graph, v_3 receives its information from p_1 at the third iteration. However, if we partition this factor graph into sub-graphs as shown in Figure 6.2 and connect those sub-graphs in cascade, v_3 receives a portion of information from p_1 through p_4 during the first iteration, and the remaining portion of the information through p_3 in the second iteration. This example shows that sub-graph sum-product algorithm may increase the message propagation speed.

In general, in iterative sum-product algorithm, an information travels at most two edges during an iteration through its target . However, in sub-graph sum-product algorithm such a limitation does not exist. If the factor graph is carefully partitioned then an information originating from a factor node can even travel the whole graph.

The factor graph of a given problem is fixed. Therefore, we cannot make two nodes closer. However, we can increase the speed of message propagation as in the example above in order to increase the convergence rate.

The speed of message propagation is not the sole factor which determines the convergence rate. If the speed of message propagation was the only factor

which determines the convergence rate then CAS-ALL and PAR-ALL LDPC decoding algorithms should have higher convergence rates. However, as we have observed in Section 5.3 this is not the case. Probably these algorithms' poor performance in accuracy, affects their convergence performance as well. From this idea we may conclude that partitioning into sub-graphs increase the convergence rate if the accuracy is not degraded.

6.2 Why All-Accessible Sub-Graphs Have Worse Accuracy?

Message calculation at factor nodes depends on an assumption that incoming messages are independent. An all-accessible sub-graph makes many variable nodes to depend on each other. Therefore, the factor nodes which operate after an all-accessible sub-graph will generate erroneous output messages. This fact will degrade the accuracy of a flow graph composed of all-accessible sub-graphs.

While showing that turbo decoding algorithm is an instance of sub-graph sum-product algorithm we have said that each component decoder runs the BCJR algorithm. BCJR algorithm is an instance of sum-product algorithm tuns on an all-accessible factor-graph. Although, turbo decoding algorithm consists of all-accessible sub-graphs, it is known that turbo decoding algorithm has good performance. This fact seems to conflict with our idea. However it is not the case because of the structure of the BCJR factor graph.

The BCJR factor graph has a fixed repetitive structure which is shown in Figure 6.3. This structure is first proposed by Wiberg [18] in his Ph.D. dissertation and explained in [1]. There is a different kind of node in this structure which are represented by double circles. These nodes are called "hidden state

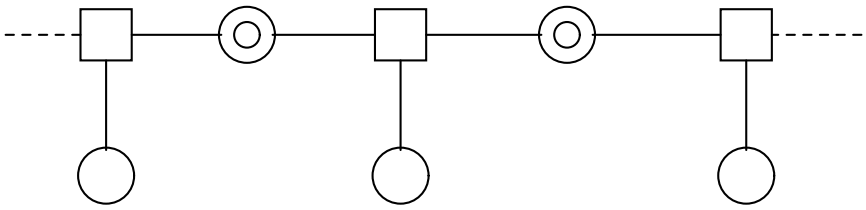


Figure 6.3: Structure of a BCJR factor graph

nodes". Actually they are variable nodes which represent some joint random variables. What is important for our case is that in the BCJR factor graph there exist at least four edges between any two variable nodes. Therefore, even if these factor graphs are connected, they are loosely connected. We believe that this is the reason why turbo decoding algorithm is so successful even if it consists of two connected sub-graphs.

This idea supports our proposal on multiple concatenated turbo codes. Since multiple concatenated turbo codes will be more punctured, the loosely connected factor graph of the BCJR algorithm will be broken and sub-graphs will look like more non-accessible. Hence, hopefully their performance will increase.

6.3 Conclusion

The most important contribution of this thesis is the sub-graph sum-product algorithm which determines an efficient schedule for factor graphs having loops. By using the sub-graph idea we develop a decoding algorithm for LDPC codes which converges twice as fast as the standard LDPC decoding algorithm without sacrificing from BER performance. This decoder is the second important contribution of the thesis. Final important contribution is

an approximation for message calculation in ISI factor graphs. This approximation transforms the complexity from exponential order to linear order while decreasing the performance only 0.3dB.

6.4 Future Work

After this work, multiple concatenated turbo codes should be revisited. It is very probable that better turbo codes can be developed by using the ideas presented in this thesis.

Sub-graph sum-product algorithm can be employed in receivers of multi-carrier, multi-user, or multi-antenna systems.

Better algorithms could be developed for partitioning a factor graph into sub-graphs. By better partitioning better convergence rates can be obtained.

Some analytical methods can be developed for analyzing the sub-graph sum-product algorithm. Especially partitioning a factor graph into sub-graphs should be analyzed in an analytic manner.

REFERENCES

[1] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor Graphs and the Sum-Product Algorithm", IEEE Transactions on Information Theory, vol.47, No.2,pp.498-519 February 2001

[2] H. A. Loeliger, "An Introduction to Factor Graphs", IEEE Signal Processing Magazine, Vol. 21, Issue 1,pp.28-41 Jan. 2004,

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J.Raviv, "Optimal decoding of linear codes for minimizing symbol error rate" IEEE Transactions on Information Theory, vol. IT-20, pp.284-287, Mar. 1974

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit error correcting coding and decoding: Turbo Codes" in Proc. 1993 IEEE Int. Conf. Communications, Geneva, Switzerland, May. 1993, pp. 1064-1070

[5] R. G. Gallager, "Low-density Parity-check" Codes, M.I.T. Press, 1963.

[6] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices", IEEE Transactions on Information Theory, vol.45, pp.399-431, Mar. 1999

[7] A. P. Worthen and W. E. Stark, "Unified Design of Iterative Receivers Using Factor Graphs", IEEE Transactions on Information Theory, vol.47, No.2, pp. 843-849 February 2001

[8] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding Belief Propagation and its Generalizations", "http://www.merl.com", January 2002

[9] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing Free Energy Approximations and Generalized Belief Propagation Algorithms", "http://www.merl.com", 2002

[10] R. J. McEliece and M. Yildirim, "Belief Propagation on partially ordered sets" in *Mathematical Systems Theory in Biology, Communication, Computation, and Finance*. New York: Springer Verlag, pp.275-299, 2002

[11] D. J. C. MacKay, "Information Theory, Inference, and Learning Algorithms", Cambridge University Press, Cambridge, UK, 2003

[12] B. J. Frey and D. J. C. MacKay, "A Revolution: Belief Propagation in Graphs With Cycles" in *Advances in Neural Information Processing Systems 10*, MIT Press, Cambridge,MA, 1998

[13] G. Colavolpe and G. Germi, "On the Application of Factor Graphs and the Sum-Product Algorithm to ISI Channels", IEEE Transactions on Communications, Vol. 53, No.5 May 2005

[14] J. Pearl, "Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference",Morgan Kauffman Publishers, San Mateo, CA, 1998

[15] F. R. Kschischang and B. J. Frey,"Iterative decoding of compound codes by probability propagation in graphical models" IEEE J. Select. Areas Commun., vol. 16, pp. 219-230, Feb. 1998

[16] D. Divsalar and F. pollara, "Multiple Turbo Codes for Deep-Space Communications" TDA Progress Report 42-121, pp.66-77, May 1995

[17] R. Morelos-Zaragoza, "The Art of Error Correcting Codes", John Wiley & Sons, West Sussex, England, 2002

[18] N. Wiberg, "Codes and Decoding on General Graphs", Ph. D. Thesis, Department of Electrical Eng. Linköping University, Linköping, Sweden, 1996