

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Decoder Design and Decoding Models for Joint Source-Network Coding

Susana Pereira Bulas Cruz

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: João Barros (Associate Professor, Ph.D.)

Co-supervisor: Gerhard Maierbacher (M.Sc.)

July 2010

A Dissertação intitulada

“DECODER DESIGN AND DECODING MODELS FOR JOINT SOURCE-NETWORK
CODING”

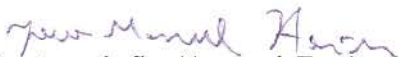
foi aprovada em provas realizadas em 22/Julho/2010

o júri



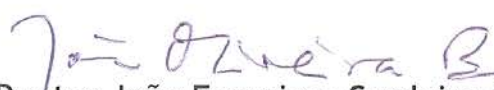
Presidente Professor Doutor Sílvio Almeida Abrantes Moreira

Professor Auxiliar do Departamento de Engenharia Electrotécnica e de
Computadores da Faculdade de Engenharia da Universidade do Porto



Professor Doutor João Manuel Freitas Xavier

Assistente do Instituto Superior Técnico de Lisboa da Universidade Técnica de
Lisboa



Professor Doutor João Francisco Cordeiro de Oliveira Barros

Professor Associado do Departamento de Engenharia Electrotécnica e de
Computadores da Faculdade de Engenharia da Universidade do Porto (Orientador).

O autor declara que a presente dissertação (ou relatório de projecto) é da sua
exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente
autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou
inspirados em trabalhos de outros autores, e demais referências bibliográficas
usadas, são correctamente citados.



Autor - Susana Pereira Bulas Cruz

Faculdade de Engenharia da Universidade do Porto

Abstract

The aim of this work is to study decoder design and decoding models for joint source-network coding. The key idea of network coding is to replace routing operations by coding operations in networks with several sinks, combining packets at intermediate network nodes to make the data distribution more flexible and efficient. A sink node can obtain its intended data using the received packets and processing them in order to extract the required information. If, additionally, there is correlation between the sources, source coding operations can reduce the amount of data for transmission, taking the correlation into account. However, performing joint source-network coding requires, in general, the usage of high complexity decoders. The main goal of the thesis is to provide algorithms that construct decoding models for a given scenario in an automatic fashion, and allow for an efficient decoder implementation.

We are interested in a sensor network scenario, in which several sources collect information and send it to a number of sinks through the network. We assume that the sensor network topology and the source model are given and can be described by a graphical model. The goal is to find a graphical model representing the coding operations performed within the network. By combining the source model with the network model, a graphical source-network coding model is constructed, which is then used for the decoder implementation.

Our results show that this approach yields an effective source-network coding and decoding mechanism with manageable complexity.

Sumário

O objectivo deste trabalho é estudar modelos de descodificação para codificação conjunta de fonte e rede. A ideia chave da codificação em rede é substituir as operações de roteamento por operações de codificação em redes com diversos nós receptores. A combinação de pacotes nos nós intermédios da rede torna a distribuição de dados mais flexível e eficiente. Um nó receptor pode obter os dados desejados usando os pacotes recebidos e processando-os de forma a extrair a informação necessária. Se adicionalmente existir correlação entre as fontes, operações adequadas de codificação de fonte possibilitam a redução da quantidade de dados para transmissão, tendo em conta a correlação existente. No entanto, realizar operações de codificação conjunta de fonte e rede requer, em geral, a utilização de descodificadores de elevada complexidade. O principal objectivo desta tese é fornecer algoritmos capazes de construir modelos de descodificação para um dado cenário de forma automatizada, permitindo uma implementação eficiente do descodificador.

Neste projecto estamos interessados num cenário de redes de sensores, em que diversas fontes colecionam informação e a enviam para um certo número de nós receptores através da rede. Assumimos que a topologia da rede de sensores e o modelo de fonte são conhecidos e podem ser descritos por um modelo gráfico. O objectivo é encontrar um modelo gráfico que represente as operações de codificação efectuadas na rede. Através da combinação do modelo de fonte com o modelo de rede, um modelo de codificação de fonte e rede é construído e usado na implementação do descodificador.

Os resultados mostram que esta metodologia produz mecanismos de codificação e descodificação efectivos e de complexidade limitada.

Acknowledgments

I would like to thank my supervisor, Prof. João Barros, for the opportunity he gave me to join his great team, and for the guidance that he provided along the work.

I would also like to thank my co-supervisor, Gerhard Maierbacher, for his helpful advice and continuous support.

Contents

1	Introduction	1
1.1	Source-Network Coding for Sensor Networks	1
1.2	Related Work	3
1.2.1	Distributed Source Coding	3
1.2.2	Network Coding	5
1.2.3	Factor Graphs and the Sum-Product Algorithm	7
1.2.4	Joint Source-Network Coding	7
1.3	Main Contributions	8
2	Preliminaries	11
2.1	System Setup and Problem Formulation	11
2.1.1	Notation	11
2.1.2	Source Model	11
2.1.3	Encoder Setup	12
2.1.4	Network	13
2.1.5	Decoder	13
2.1.6	Problem Statement	14
2.2	Graphical Models	15
2.2.1	Factor Graphs as Graphical Models	15
2.2.2	Source-Optimized Clustering	16
3	Subgraph Construction	17
3.1	Algorithm for Two Sources	17
3.2	Extension to S Sources	19
3.2.1	Possible Generalizations	22
3.3	Linear Optimization	23
4	Coding Approach	25
4.1	Network Model	25
4.2	Decoding Model and Algorithm	28
4.2.1	Factor Graph Representation	28
4.2.2	Sum-Product Algorithm	29
4.3	Code Design	31
4.3.1	Random Mapping Function	31
4.3.2	Sequential Mapping Function	32
4.3.3	Joint Source-Network Coding Mapping Function	32

5	Results	35
5.1	Working Example	35
5.2	Complexity Analysis	39
5.2.1	Complexity of Factor Graph Nodes	39
5.2.2	Number of Factor Graph Nodes	39
5.2.3	Overall Complexity	40
5.3	Performance Analysis	41
6	Conclusions and Future Work	45
A	Prototype Implementation of Subgraph Construction Method	47
	References	51

List of Figures

1.1	Correlated source coding configuration used in [1]	3
1.2	Slepian-Wolf admissible rate region corresponding to Fig. 1.1	4
1.3	Butterfly Network Example	6
2.1	System model	13
2.2	Factor graph nodes	15
2.3	Source-optimized clustering example	16
3.1	Rate region and virtual node and edges for two correlated sources	20
3.2	Rate region and virtual node and edges for two uncorrelated sources	20
3.3	Virtual nodes and edges for four uncorrelated sources	21
3.4	Virtual nodes and edges for six correlated sources in four uncorrelated clusters	22
4.1	Suitable and unsuitable network examples	25
4.2	Network example	27
4.3	Representation of the network model example	27
5.1	Illustrated example: source-optimized clustering	36
5.2	Illustrated example: original and simplified networks	36
5.3	Illustrated example: network model representation	38
5.4	Illustrated example: decoding factor graph	38
5.5	Original and simplified networks	42
A.1	Content of matrix \mathbf{A} representing the inequality constraints	49
A.2	Content of matrix \mathbf{A}_{eq} representing the equality constraints	49

List of Tables

5.1	Flow rate values	37
5.2	Illustrated example: optimization results for sink t_1	37
5.3	System SNR in dB for a strong correlation between the sources ($\beta = 0.25$)	43
5.4	System SNR in dB for a weak correlation between the sources ($\beta = 1$)	44

Chapter 1

Introduction

1.1 Source-Network Coding for Sensor Networks

Distributed compression and cooperative forwarding are key towards ensuring the energy efficiency of sensor networks. This class of networks consists of multiple distributed sensors that collect information at diverse locations and have severe limitations in terms of power and computation. Commonly measured parameters are temperature, humidity, pressure, wind direction and speed, illumination intensity, vibration intensity, sound intensity, power-line voltage, chemical concentrations, pollutant levels and vital body functions. Sensor networks depend on many components working together in an efficient, cooperatively, comprehensible and trustworthy manner. The sensors in these applications may be small or large, and the networks may be wired or wireless [2]. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance, but they are now used in many industrial and civilian application areas [3]. Current and potential applications of sensor networks include: military sensing, physical security, air traffic control, traffic surveillance, video surveillance, industrial and manufacturing automation, distributed robotics, environment monitoring, and building and structures monitoring [2].

Although sensor networks are a vast area, wireless networks of microsensors, which have emerged as a fundamentally new tool for monitoring inaccessible environments, probably offer the most potential in changing the world of sensing [4]. These specific networks consist of a large quantity of autonomous detection stations called sensor nodes, randomly deployed in the area of interest, each of which should be a small, lightweight, portable, low-power and low-cost wireless sensor. They are distinguished from traditional sensors by strict limitations on system bandwidth and sensor energy resources. These constraints motivate the use of data compression at each sensor. The output of all these sensors is sent to one or more central points, called base stations or sinks, for joint decoding and further processing.

The approach being studied in this work consists in concentrating more efforts in the decoder at the sinks while keeping the independent encoders for different sources as simple as possible. This is justified by their severe constraints on energy consumption. However, this implies a very

high decoder complexity, making it difficult to find a feasible decoder implementation for a given scenario.

In this research we consider multiple correlated sources that aim to send their data to various sinks through the network. In a correlated source scenario, data from one source already contains some information about other sources, reducing the amount of data required for transmission. Distributed source coding operations can be performed at each source to achieve this aim, as theoretical results show that, even if the encoding processes are separately and independently performed, the correlation between the sources can still be exploited [1]. This approach is adequate to wireless sensor networks as the data collected by the sensors is usually strongly correlated, due to the spatial and temporal proximity of the measurements.

In our network scenario, information collected by sensors (source nodes) is packed and sent to other sensors which retransmit it (intermediate nodes) until it reaches the destinations (sink nodes). The amount of data moved successfully from a source to a sink in a given time period, called throughput, is an important characteristic of any network. Network coding [5] is a key aspect to reach the maximum throughput in a network with several sources sending packets to several sinks. The key idea is to combine packets inside the network by performing coding operations in the intermediate nodes, instead of traditional routing operations, to make the data distribution more flexible and efficient. A sink node processes the packets it receives in order to reconstruct the information originally intended for that sink.

It is possible to perform distributed source coding separate from network coding but this approach is not optimal for certain scenarios [6]. The solution is using joint source-network coding. However, this approach implies high computational complexity in joint decoding operations [7], making the decoder implementation infeasible for a large number of nodes.

The main goal of this thesis is to provide adequate algorithms which allow the construction of decoding models for joint source-network coding problems in an automatic fashion, based on previous work on practical joint source-network decoding [8].

To achieve this aim, we intend to obtain graphical models for both source and network. It is assumed in this work that the topology of the considered sensor network scenario and the source model is given and fully described by a graphical model. Then, given a data transmission scheme of interest, a graphical model representing the network coding operations must be found. With the combination of these two models it is possible to derive a graphical source-network coding model, which can be used for an adequate and efficient decoder implementation. The main focus of this project is to obtain methods for the construction of those graphical models with an efficient approach, given the data transmission scheme. There is also interest in evaluating the impact of the underlying communication protocols on the decoder performance.

Although wireless sensor networks are the most immediate and known application for the work being developed, there may also be other areas that can potentially benefit and make progress with the proposed solutions, such as information dissemination in peer-to-peer networks.

This document contains this section with a brief explanation of the dissertation project, namely its motivation, scope and objectives. The next section 1.2 is a review of related work, divided into

four subsections related to the main topic that fall into different areas of investigation: distributed source coding, network coding, factor graphs and joint source-network coding. In the first section of Chapter 2, entitled system setup and problem formulation, we present a research scenario and formulate the problem under analysis. The second section contains a brief explanation of the graphical models used and a description of the source-optimized clustering method applied in the source model. In Chapter 3 we describe the subgraph construction method we use to obtain the optimal flow rates for our multicast scenario. Chapter 4, entitled coding approach, presents our network model and joint source-network model factor graph. It also includes in the last section a description of the three functions we developed for our code design. Chapter 5 contains the results of our work. The first section presents an illustrated example of the whole process. The second section consists in a complexity analysis and is followed by a third section with a performance analysis. Finally, Chapter 6 contains the conclusions of this dissertation and draws some directions for future work.

1.2 Related Work

1.2.1 Distributed Source Coding

We consider a scenario in which information originated from correlated sources flows through a network to a number of sinks, where appropriate estimation of the source data should be achieved.

An approach to this kind of problems is based on the idea of distributed source coding. Seminal work on this area has been developed by Slepian and Wolf [1] back in 1973, namely on noiseless coding of correlated information sources. The authors studied the information theoretical limits of distributed source coding, sometimes referred to as the Slepian-Wolf bound, suggesting that two isolated source encoders may compress data as efficiently as if they were communicating with each other. Slepian and Wolf did this by generalizing certain well-known results on noiseless coding of a single discrete information source to the case of two correlated sources albeit with separate encoders. The encoder of each source operates without knowledge of the other source. The decoder uses the outputs from both encoders and takes advantage of the correlations between the sources.

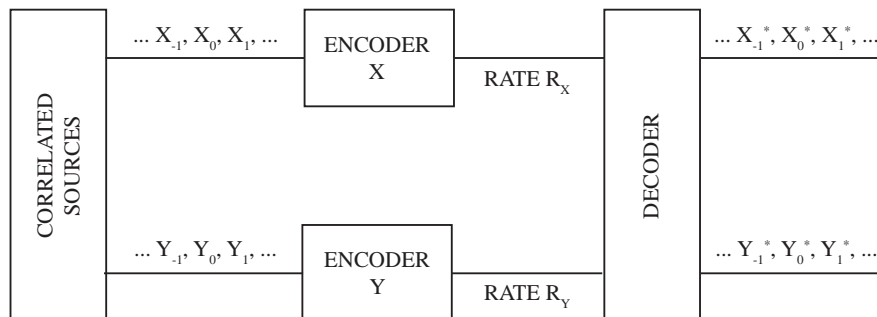


Figure 1.1: Correlated source coding configuration used in [1]

Slepian and Wolf used the configuration showed in Fig. 1.1. According to the Slepian-Wolf coding theorem, the set of all ordered pairs $[R_X, R_Y]$, called the achievable rate region for the distributed sources X and Y , which respectively generate the sequences $\dots, X_{-1}, X_0, X_1, \dots$ and $\dots, Y_{-1}, Y_0, Y_1, \dots$, is limited by $R_X \geq H(X|Y)$, $R_Y \geq H(Y|X)$ and $R_X + R_Y \geq H(X, Y)$, where R_X and R_Y represent the rates of the encoded sequences (Fig. 1.1) and H is the entropy. Fig. 1.2 shows the Slepian-Wolf admissible rate region graphically for joint decoding of the two sources X and Y .

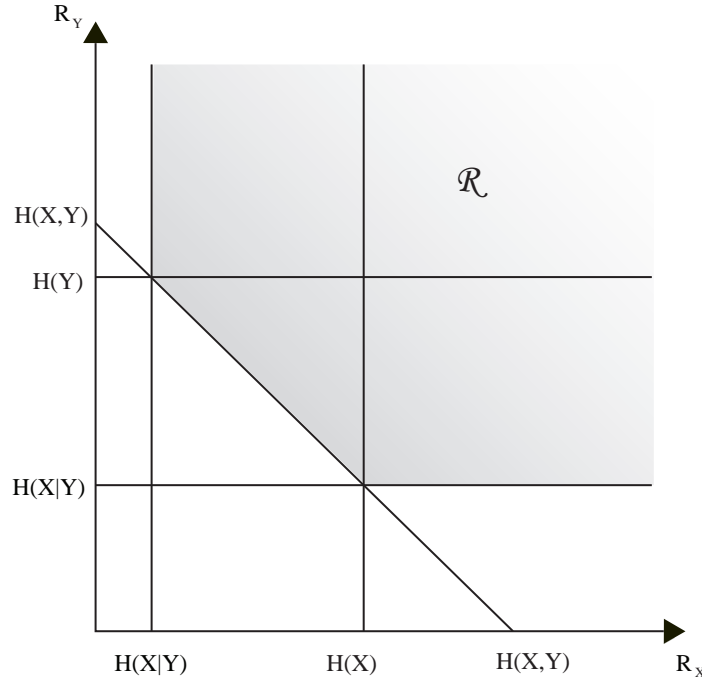


Figure 1.2: Slepian-Wolf admissible rate region corresponding to Fig. 1.1

In 1975, Cover [9] extended this bound to the case of more than two correlated sources, showing that the Slepian-Wolf theorem still holds without change for an arbitrary number of ergodic processes and countably infinite alphabets.

The approach first proposed by Slepian and Wolf for two encoders is not constructive, suggesting that more work should be dedicated to practical methods and low-complexity schemes. The first work on low-complexity methods has been published by Flynn and Gray in 1987 [10]. It is founded on Slepian and Wolf's proposal but presents it as an engineering problem that involves sensing an environment and making estimates based on the phenomena sensed. The distributed sensing systems described in [10] contain four most important elements: a target (with unknown attributes, whose values the system should be able to infer), an environment (containing the targets), a sensor and a decision mechanism. The observations, which are communicated to the decision-maker over channels of limited capacity, must be separately encoded so that the target attributes can be estimated with minimum distortion. The authors answer two main questions related to this problem: (i) what communication rates and distortions can be achieved if the complexity in the encoder is unlimited and (ii) how can the encoder be designed for good performance if it has

to be a quantizer. Their results are of extreme importance to this project, whose aim is to design and implement a practical scheme for decoding information coming from continuous sequences generated by independently coded sources, as their work is one of the first practical approaches to construct distributed source codes for this scenario.

There are also other authors who have contributed with distributed source coding solutions inspired by Slepian and Wolf's work. In 2004, Xiong et al. [11] wrote a paper with an overview of the research results in this area, focusing on the application to sensor networks. In this paper, special attention is given to LDPC (low-density parity-check) codes, turbo codes and Wyner-Ziv coding (see [11] and references therein).

1.2.2 Network Coding

Another important aspect of sensor networks is the communication. Usually data travels through a network based on routing operations. Routing in traditional networks transmits information by a method known as store-and-forward. Data is sent by a source node to each destination node through intermediate nodes, which forward a copy of the packets onto each output link that leads to at least one of the destinations. In this scenario, there is no data processing at the intermediate nodes, except for data replication. In this paradigm, packets traveling in the network are somehow regarded as cars routed in a traffic flow scheme [12]. In [5], this routing approach leads to a view of information within the network as a "fluid" that can only be routed or replicated.

In a unicast scenario, it is possible to obtain the optimal solution for the maximum amount of flow passing from the source to the sink with the Max-Flow Min-Cut Theorem. In simple words this theorem [13] states that the maximum amount of flow passing from the source to the sink is equal to the minimum cut capacity (e.g. in the fluid approach would be the flow in the pipe with the minimum diameter).

The multicast scenario is more complicated to analyze and optimal solutions generally cannot be based on traditional routing operations. The first work to find an optimal solution for the multicast scenario was authored by Ahlswede et al. and published in 2000 [5]. They introduced a new network information flow approach which states that information can be coded at the intermediate nodes. These coding operations have the potential to increase throughput and, even in the worst case, they could not decrease the performance of a network because they would perform precisely the same actions as the routers.

As referred to in [12], this approach was in part built on the idea that transmitting adequate clues about data can actually be more useful than sending the data directly. This concept was introduced by Claude E. Shannon, who launched a revolution by founding information theory with the publication of his landmark paper "A Mathematical Theory of Communication" in 1948 [14].

Ahlswede et al. [5] followed this idea in the context of communication networks, as they proposed a model where the receiver is able to reconstruct the source data, as soon as it obtains a sufficient number of clues, each of which contains part of the original data from determined sources. In this approach, the sink does not need to receive all the evidence emitted because

analyze P2P file distribution, wireless networks, ad-hoc sensor networks, network tomography and network security.

1.2.3 Factor Graphs and the Sum-Product Algorithm

When we have to deal with a problem involving global functions of many variables, it is often useful to exploit the way in which the global function factors into a product of simpler local functions, each of which only depending on a subset of variables. Kschischang et al. [18] define factor graphs as bipartite graphs that allow us to visualize this factorization. Factor graphs have the advantage of being universal allowing us to represent any set of variables and functions. A factor graph has one node for every variable, called variable node, and one for every local function, called factor node. There is also an edge which connects a variable node to a given factor node, if and only if that variable is an argument of that function.

Message-passing algorithms are used to calculate global functions based on local functions only. In [18], the authors present a simple algorithm to do so, which can be used in factor graphs, called the sum-product algorithm. Factor graphs allow one to simplify and visually describe a very large class of problems. Many efficient algorithms are special cases of the sum-product algorithm applied to factor graphs, namely the FFT (Fast Fourier Transform), Viterbi Algorithm, Forward-Backward Algorithm, Kalman Filter and Bayesian Network Belief Propagation. The sum-product algorithm can encompass an enormous variety of practical algorithms despite being based on a single simple computational rule. The rule according to which the sum-product algorithm operates, as presented in [18], is the following: "The message sent from a node v on an edge e is the product of the local function at v (or the unit function if v is a variable node) with all messages received at v on edges other than e , summarized for the variable associated with e ".

It is also worth pointing out that factor graphs can be divided into factor graphs with cycles and cycle-free factor graphs. The sum-product algorithm may be applied to both cases, however, an exact result is, in general, only obtained for factor graphs without cycles. For more details on factor graphs and the sum-product algorithm please refer to [18].

1.2.4 Joint Source-Network Coding

In 2006, Ramamoorthy et al. [6] wrote a paper on separating distributed source coding from network coding considering the problem of distributed source coding of multiple sources over a network with multiple receivers, which seek to reconstruct all of the original sources. There is a solution to this problem proposed previously by Ho et al. [19], using random network coding, that comes at the potentially high cost of jointly decoding the source and the network codes. In fact, the joint approach makes the process in the decoder much more complex, since network coding may destroy the structure in the encoder that allows tractable decoding. One specific case for which this has a high probability of happening is precisely the use of random network coding referred to in the solution presented in [19]. Furthermore, it is demonstrated by some authors, e.g. Effros et al.

2003 [20], that there exist conditions under which this source-channel separation in nonmulticast networks fails. In [6] the authors demonstrated that failure for multicast networks.

Considering this, Ramamoorthy et al. [6] studied the problem of separation between source coding and network coding and investigated the conditions under which separation holds. These works raised the question whether there is a gain in doing source and network coding jointly. The conclusion of this work indicated that joint source-network coding would be a better solution than a separate approach. However, due to the decoder high complexity, approaches based on the joint scenario were not feasible, as explained by Coleman et al. [7] in 2005.

So, work progressed in the direction of devising a low-complexity approach that could work for a large number of encoders. In 2006, Barros and Tüchler [21] proposed a low-complexity decoder design solution, which offers a trade-off between complexity and end-to-end distortion. The authors justify their approach by arguing that, as complexity grows exponentially with the number of sensors in a standard implementation of the optimal decoder-based on minimum mean-square error estimation, this approach is unfeasible for large-scale sensor networks. They propose instead a decoding algorithm based on factor graphs to model the correlation between the source data, that uses the sum-product algorithm, making complexity grow linearly with the number of nodes. As opposed to algorithms whose complexity grows exponentially with the number of nodes, this algorithm is scalable, which makes it usable in networks with a large number of nodes.

Recent work related to this thesis was developed by Maierbacher and Barros [22], [23]. The first paper [22], published in 2007, proposes a framework for constructing index assignments that have very low-complexity while performing very close to theoretical bounds. In the second one [23], published in 2009, the authors present a scalable solution for the distributed source coding problem in large-scale sensor networks, in which correlated data has to be separately encoded and transmitted to a common receiver with a rate-distortion constraint. Their low-complexity solution is based on index assignment, source-optimized clustering and sum-product decoding.

In 2009, Maierbacher, Barros and Médard [8] analyzed multiple correlated sources scenario, where data is transmitted over a network to more than one sink, using source-network coding. They implement a practical source-network decoder for the case of three sources and two sinks, representing the overall system by a factor graph, on which they run the sum-product algorithm.

Despite these important works and their contributions towards achieving a general decoding solution, there is still no practical implemented decoder that works for a large-scale sensor network scenario, with a high number of correlated sources.

1.3 Main Contributions

In this work, we propose a joint source-network decoding model for large-scale networks and arbitrary scenarios. We show that this decoder can be implemented with feasible complexity, because its complexity grows linearly with the size of the network. Our decoding model is created by combining in a single graph the source and the network models. In order to construct a graphical model to represent the network, we need adequate flow rates for multicasting information. For that

purpose, we extend the subgraph construction method for two correlated sources proposed by Lee et al. [24] to an arbitrary number of correlated sources, by combining it with a source optimized-clustering method.

We also present and analyze three different coding strategies, including a systematic coding scheme for joint source-network decoding we developed.

Chapter 2

Preliminaries

2.1 System Setup and Problem Formulation

2.1.1 Notation

We start by introducing our notation. Random variables are always denoted by capital letters, e.g. U , where its realizations are denoted by the corresponding lowercase letters, e.g. u . Vectors are denoted by bold letters and, if not stated differently, assumed to be column vectors, e.g. $\mathbf{u} = (u_1, u_2, \dots, u_S)^T$ and $\mathbf{U} = (U_1, U_2, \dots, U_S)^T$. The expression $\mathbf{0}_S = (0, 0, \dots, 0)^T$ is the length- S zero vector. Matrices are denoted by bold capital letters, e.g. \mathbf{A} . It is always clear from the context, or stated explicitly, if a bold capital letter refers to a vector of random variables or to a matrix. Index sets are denoted by capital calligraphic letters, e.g. \mathcal{S} , unless otherwise noted, where the set's cardinality is referred to by the usage of vertical bars, e.g. $|\mathcal{S}|$. We follow the convention that variables indexed by a set denote a set of variables, e.g. if $\mathcal{S} = \{1, 2, 3\}$ then $u_{\mathcal{S}} = \{u_1, u_2, u_3\}$, and use the same concept to define vectors of variables, e.g. $\mathbf{u}_{\mathcal{S}} = (u_1, u_2, u_3)^T$. Furthermore, the entries of a vector are referred to by specifying its index within parentheses, e.g. $\mathbf{u}(0)$ refers to the first and $\mathbf{u}(S-1)$ to the last entry of the length- S vector \mathbf{u} .

2.1.2 Source Model

We consider a system-setup, illustrated in Fig. 2.1, with S correlated sources U_1, U_2, \dots, U_S . Each of those sources, indexed by $s \in \mathcal{S}$, $\mathcal{S} = \{1, 2, \dots, S\}$, is observed by a continuous-valued source sample $u_s(t)$ at time instant t . To simplify the approach, when regarding the source model, we analyze only one time instant, dropping the time index t by considering only spatial correlations between measurements and not their temporal dependence.

The output symbols are collected in the vector $\mathbf{u} = (u_1, u_2, \dots, u_S)^T$, $\mathbf{u} \in \mathbb{R}^S$. We could assume that the joint probability density function (PDF) $p(\mathbf{u})$ is known but as we will need the correlation matrix later we will assume that at each time instant t the vector of source samples \mathbf{u} is one realization of a S -dimensional Gaussian random variable distributed according to $\mathcal{N}(\mathbf{0}_S, \mathbf{R})$. We

set the correlation matrix to

$$\mathbf{R} = \begin{bmatrix} 1 & \rho_{1,2} & \cdots & \rho_{1,S} \\ \rho_{2,1} & 1 & \cdots & \rho_{2,S} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{S,1} & \rho_{S,2} & \cdots & 1 \end{bmatrix},$$

such that the individual source samples u_n , $n \in \mathcal{N}$, have zero mean $E\{u_n\} = 0$, unit variance $Cov\{u_n, u_n\} = 1$ and are correlated with u_m , $m \neq n$, $m \in \mathcal{N}$, according to the correlation coefficient $\rho_{n,m} = Cov\{u_n, u_m\}$.

2.1.3 Encoder Setup

Each encoder consists of two stages: a scalar quantizer followed by a mapping function, as described below and showed in Fig. 2.1.

2.1.3.1 Quantizer

The encoder's first stage performs a scalar quantization on the observed source samples $\mathbf{u} \in \mathbb{R}$, approximating the infinite set of real-values source samples by a finite number of reconstruction levels. These levels are indexed by the quantization indices $i_s \in \mathcal{I}_s$, $\mathcal{I}_s = \{0, 1, \dots, |\mathcal{I}_s| - 1\}$. They are obtained by applying the *quantization function* $q_s : \mathbb{R} \rightarrow \mathcal{I}_s$ on each u_s such that $i_s = q_s(u_s)$. This quantization function q_s can be uniform or non-uniform. In this specific case, as the quantizer input values are originated from a non-uniform, Gaussian source, a non-uniform Lloyd-Max quantizer is used. It was chosen to minimize the distortion because it determines the optimum decision and reconstruction levels when the optimization criterion used is the MSE (mean square error) criterion [23], which we describe later in the decoder section. During quantization, an input value u_s is mapped onto the index i_s if it falls into the interval $\mathcal{B}_s(i_s) \subseteq \mathbb{R}$ between the decision levels $b_s(i_s)$ and $b_s(i_s + 1)$ such that $b_s(i_s) < u_s \leq b_s(i_s + 1)$. The obtained quantization index i_s is then associated with the reconstruction level $\tilde{u}_{s,i_s} \in \tilde{\mathcal{U}}_s$, $\tilde{\mathcal{U}}_s = \{\tilde{u}_{s,0}, \tilde{u}_{s,1}, \dots, \tilde{u}_{s,|\mathcal{I}_s|-1}\}$, representing all source samples u_s falling into the quantization region $\mathcal{B}_s(i_s)$. The reconstruction levels \tilde{u}_{s,i_s} are chosen to be the centroid of the quantization region $\mathcal{B}_s(i_s)$.

2.1.3.2 Index Assignment

At the second stage of each encoder an index assignment is performed, in which the quantization index $i_s \in \mathcal{I}_s$ is mapped onto the codeword $w_n \in \mathcal{W}_s$, $\mathcal{W}_s = \{0, 1, \dots, |\mathcal{W}_s| - 1\}$, by the *mapping function* $a_s : \mathcal{I}_s \rightarrow \mathcal{W}_s$ such that $w_s = a_s(i_s)$. By choosing codeword alphabets $\mathcal{W}_1, \mathcal{W}_2, \dots, \mathcal{W}_S$ smaller than the number of quantization levels in $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_S$, i.e. $|\mathcal{W}_s| < |\mathcal{I}_s|$, data compression can be achieved. The challenge is to find an appropriate mapping function that makes it possible to compress the information without a significant increase in the distortion.

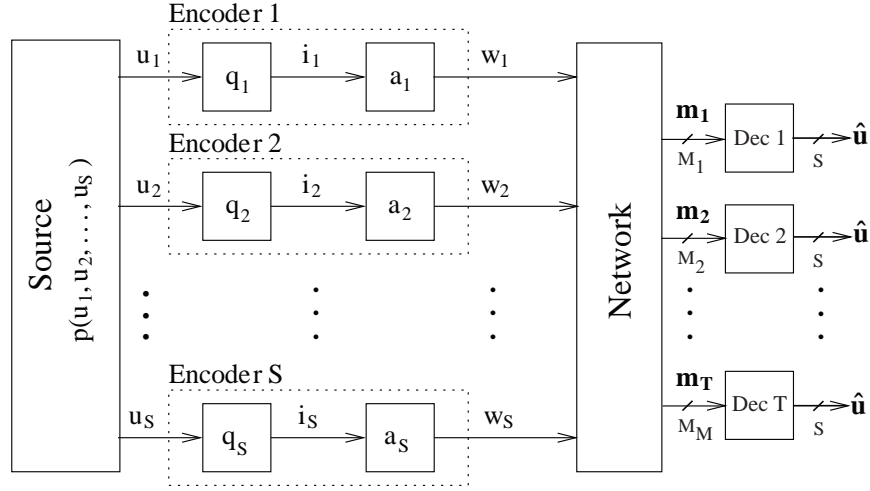


Figure 2.1: System model

2.1.4 Network

We consider a network with N nodes, represented by the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = v_n : n \in \mathcal{N}$ is the set of nodes v_n uniquely identified by the indices $n \in \mathcal{N}$, $\mathcal{N} = \{1, 2, \dots, N\}$ and \mathcal{E} is the set of directed edges $e_{k,l} = (v_k, v_l)$ where $k \in \mathcal{N}$ identifies the start node v_k and $l \in \mathcal{N}$, $l \neq k$, identifies the destination node v_l . Each edge has a capacity $c_{k,l}$ that is the maximum amount of data that may be sent through that edge from the node v_k to the node v_l in each time unit. In the network there are source nodes, which generate information (collect it from the environment), sink nodes, which receive information, and ordinary network nodes, which participate in transmitting the information from the sources to the sinks. As already defined in the source model section, the set $\mathcal{S} \subseteq \mathcal{N}$ identifies the source nodes within the network. The set of sink nodes is identified by the set $\mathcal{T} \subseteq \mathcal{N}$, $\mathcal{T} = \{1, 2, \dots, T\}$. In this work, we assume that every node in the network is able to perform coding operations and can be used to convey data from the sources to the sinks. Each sink contains a decoder which seeks to recover the original data sent by all the sources.

2.1.5 Decoder

Not considering the network and assuming one link from all sources to all sinks in perfect channels with error-free transmissions, each of the S messages received by the T decoders would be the exact codeword $w_{s,s} \in \mathcal{W}_s$, originated by each encoder after the index assignment stage, resulting in the vector of codewords $\mathbf{w} = (w_1, w_2, \dots, w_S)^T \in \mathcal{W}^S$, $\mathcal{W}^S = \prod_{s=1}^S \mathcal{W}_s$. In this simple case each decoder would use this received codeword vector \mathbf{w} and available knowledge of the source statistics \mathbf{u} to form the estimate $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2, \dots, \hat{u}_S)^T$, $\hat{\mathbf{u}} \in \mathbb{R}^S$, of the originally observed source samples $\mathbf{u} \in \mathbb{R}^S$. The corresponding decoding function is defined as $d : \mathcal{W}^S \rightarrow \mathbb{R}^S$ such that $\hat{\mathbf{u}} = d(\mathbf{w})$.

However, as we assumed the network nodes have code capabilities, the packets that leave each source node, can be combined in the intermediate nodes and originate a new packet. Thus, the

vectors of messages $\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_T$ that arrive at each of the T decoders at the respective sinks can have different alphabet sizes, respectively M_1, M_2, \dots, M_M , as shown in Fig. 2.1. The vector of messages $\mathbf{m}_t, t \in T$, is a function of the codewords \mathbf{w} .

In classical approaches, each decoder has to operate to recover the original messages sent by the sources, which are the codewords \mathbf{w} , and only then it can make the estimates with the procedure described above. That means the decoder has to perform decoding operations to reverse both the coding operations performed in the source encoders and the network coding operations. A separate approach is not always possible so we will use a joint source-network coding scheme. This joint source-network decoding approach results in a high decoder complexity [7].

Regarding each of the T sinks, we assume that the decoder at the considered sink node has: (i) access to the received packets $\mathbf{m}_t(\mathbf{w})$, (ii) a-priori knowledge about the source statistics $p(\mathbf{u})$ and (iii) a-priori (or transmitted, e.g. in the packet header knowledge about which network nodes and edges were traversed by the received packets as well as the traversed nodes' coding functions and the traversed edges' transition probabilities.

We consider the MSE $E\{\|\hat{\mathbf{U}} - \mathbf{U}\|^2\}$ between the estimates $\hat{\mathbf{U}} = (\hat{U}_1, \hat{U}_2, \dots, \hat{U}_N)^T$ and source samples $\mathbf{U} = (U_1, U_2, \dots, U_N)^T$ as the fidelity criterion to be minimized by the decoder.

The optimal estimate $\hat{u}_n(\mathbf{m}_t(\mathbf{w}))$ for a given message vector $\mathbf{m}_t(\mathbf{w})$ can be obtained by conditional mean estimation (CME) such that

$$\hat{u}_n(\mathbf{m}_t(\mathbf{w})) = E\{U_n | \mathbf{m}_t(\mathbf{w})\} = \sum_{i_n=0}^{|\mathcal{S}_n|-1} E\{U_n | i_n\} \cdot p(i_n | \mathbf{m}_t(\mathbf{w})) = \sum_{i_n=0}^{|\mathcal{S}_n|-1} \tilde{u}_{n,i_n} \cdot p(i_n | \mathbf{m}_t(\mathbf{w})), \quad (2.1)$$

which has a high complexity (for more details please refer to [23]).

2.1.6 Problem Statement

Under the system model described above (Fig. 2.1), our main goal is to provide algorithms that allow us to construct a feasible decoder for joint source-network coding problems in large-scale networks with an arbitrary high value of N . As we pointed out before, the decoder for these scenarios is usually characterized by its high complexity [7]. As shown in [21], the complexity of the standard implementation of the optimal MMSE (minimum mean-square error) decoder grows exponentially with the size of the network. We intend to use graphical models to represent our source and network scenarios by constructing factor graphs where we can run the sum-product algorithm [18], making that complexity grow linearly with N . Based on previous work on practical joint source-network decoding carried out by Maierbacher et al. [8], our aim is to devise a graphical source-network coding model that can be directly used for an efficient decoder implementation. The main focus of this work is to derive methods on how to construct those graphical models in an efficient way given the data transmission scheme. We intend to evaluate the impact of the underlying communication protocols onto the decoder performance by implementing a working prototype.

2.2 Graphical Models

In this section we provide a brief introduction to the graphical representations used in our models. For these graphical representations we chose the factor graphs described in detail in [18], which can be associated with summary propagation algorithms that operate by passing messages (summaries) along the edges of the graph. The summary propagation algorithm we used is the sum-product algorithm, also known as belief propagation or probability propagation.

2.2.1 Factor Graphs as Graphical Models

Factor graphs are used in probabilistic modeling. They are used to represent probability distributions since conditional and unconditional independence of random variables is expressed in terms of a factorization of their joint probability mass or density function.

A factor graph is a bipartite graph, $\mathcal{G}(\mathcal{V}, \mathcal{F}, \mathcal{E})$, where the vertices \mathcal{V} represent the variables, the vertices \mathcal{F} represent the factors, and undirected edges \mathcal{E} are connected between \mathcal{V} and \mathcal{F} .

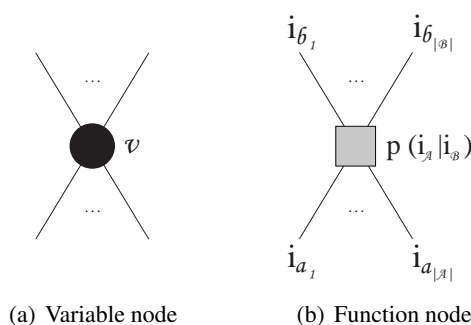


Figure 2.2: Factor nodes representation: (a) variable node and (b) function node.

The graphical representation adopted in this dissertation for the two types of nodes of our factor graphs is as follows.

Variable nodes correspond to the system variables. They are represented by a circle, in this dissertation a black filled circle, as showed in Fig. 2.2(a).

Function nodes, also called factor nodes, characterize the local functions, whose arguments are subsets of the variables. They are represented by squares, in our case grey filled squares, as illustrated in the example of Fig. 2.2(b).

If a variable is an argument of a function, there is an edge connecting the variable node that corresponds to that variable to the function node which represents that function. The nodes have usually incoming and outgoing edges and the *degree* of a node is the total number of edges.

There are two types of factor graphs: factor graphs with cycles and cycle-free factor graphs, also called factor trees. A factor graph is a tree if there is only one path between any two nodes. These factor graphs have special characteristics that can be exploited in the message passing algorithms. This topic will be developed with more detail later on, when we present the factor graph decoding algorithm used in the source-network model.

2.2.2 Source-Optimized Clustering

In our source model, we use the source-optimized clustering method described in detail in [23]. We choose this model because its clustering and coding strategy can be easily combined with a factor graph model for a joint decoder as described in [21]. This source model is based on a partition of the entire set of encoders into subsets named *clusters*, with the aim of reducing the optimization effort on the encoder side by considering only the statistical dependencies within certain subsets of sources, that is, within each cluster. As described in [23] the method selected to find adequate clusters is based on Kullback-Leibler distance (KLD).

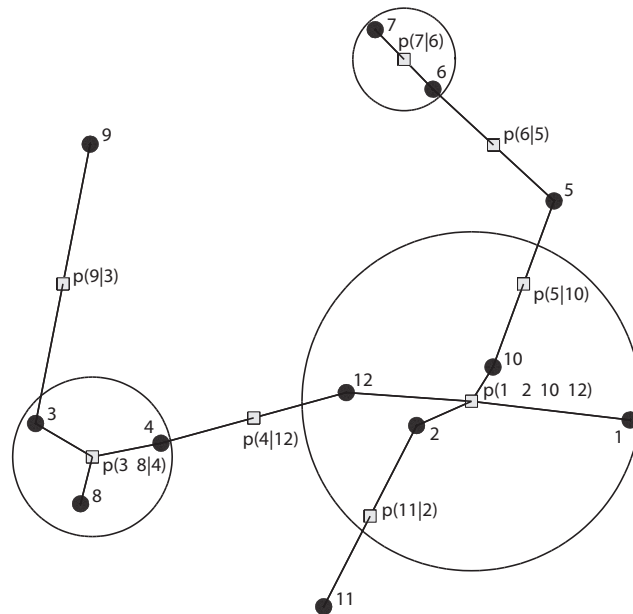


Figure 2.3: Source-optimized clustering example

Fig. 2.3 shows an example of this source-optimized clustering model for 12 source nodes with a maximum cluster size of 4 nodes. In the example there are 6 clusters assuming that single nodes can be considered as one-element clusters.

Chapter 3

Subgraph Construction

In this chapter we will present a subgraph construction method. The resulting subgraph is constructed from the network that defines the communication links, which is a directed graph, defined by its nodes and directed edges. We use this subgraph construction method for two main reasons. The first one is that we need to calculate the flow rates for each edge to multicast the information through the network, in order to construct the decoder factor graph functions that represent the coding operations in the network nodes. This method provides a linear optimization formulation for allocating the edge flow rates in the network. The other reason is that the model for the entire original network is very complex due to the high number of paths, and therefore impracticable to use. This method allows us to use a simpler graph specific for each sink, i.e. a subgraph with only the relevant paths for each sink. In Appendix A we present our prototype implementation of the subgraph construction method for an arbitrary number of correlated sources described in section 3.2.

3.1 Algorithm for Two Sources

Subgraph construction consists in choosing the network resources by selecting edges and the corresponding flow rates to support a multicast connection [24]. The aim is to find the minimum-cost subgraph, i.e. the minimum flow rate for each edge that allows us to transmit the information from the sources to the sinks.

Although the method is explained in detail in [24], we will present a short overview in order to better describe our implementation.

We begin by presenting the problem formulation from [24]. We have the following inputs: a graph $G = (N, A)$, the edge weights $w_{ij} : (i, j) \in A \rightarrow \mathbb{R}^+$, the edge capacities $c_{ij} : (i, j) \in A \rightarrow \mathbb{R}^+$, the set of two source nodes S , the sources X_i generated at s_i with rate $H(X_i)$, the joint rate $H(X_1, X_2)$, and the set of receiver nodes T .

Let

$$G^* = (N^*, A^*),$$

$$N^* = N \cup S^*,$$

$$A^* = A \cup E,$$

where

$$E = \{(S^*, j) | j \in S\}$$

$$c_{S^*j} = H(X_j)$$

$$w_{S^*j} = 0.$$

As described above, we augment the graph by adding a virtual source S^* and a virtual edge from S^* to each actual source. The virtual source S^* and the virtual edges are illustrated in Fig. 3.1(b). The overall rate from the virtual source to each of the receivers is denoted by R and set to the joint entropy of the sources.

The desired minimum-cost subgraph is found using the following linear optimization problem from [24].

Minimize $\sum w_{ij}z_{ij}$, $z_{ij} \in A^*$ subject to

$$c_{ij} \geq z_{ij}, \quad \forall (i, j) \in A, \quad (3.1)$$

$$c_{ij} \geq x_{ij}, \quad \forall (i, j) \in E, \quad (3.2)$$

$$z_{ij} \geq x_{ij}^{(t)} \geq 0, \quad \forall (i, j) \in A^*, t \in T, \quad (3.3)$$

$$\sum_{\{j|(i,j) \in A^*\}} x_{ij}^{(t)} - \sum_{\{j|(j,i) \in A^*\}} x_{ji}^{(t)} = \sigma_i^{(t)}, \quad \forall i \in N^*, t \in T, \text{ where } \sigma_i^{(t)} = \begin{cases} R & \text{if } i = S^* \\ R & \text{if } i = t \\ 0 & \text{otherwise.} \end{cases} \quad (3.4)$$

The desired subgraph is z_{ij} for $z_{ij} \in A$ and is found by solving this minimization problem: the goal is to minimize the cost of the subgraph. The desired subgraph cost is calculated as the summation of the products of the flow rates by the weights of the edges, such that $\sum w_{ij}z_{ij}, z_{ij} \in A$.

The weights of the edges can be set to any value we choose. We can set them all equal or we can use a function of some quantity of interest, such as the distance, latency or energy. The edges that have lower cost are more likely to be included in the subgraph and have higher flow rates because the purpose is to minimize the total cost of the subgraph. The virtual source S^* and the two virtual edges added to the graph (distinguished from the original ones by the dashed lines in which they are represented) are an abstraction which allow for the determination of the amount of

source data that must be transmitted by each source, while fulfilling the Slepian-Wolf conditions. This aspect will be explained later on.

The variables x_{ij} represent the flow rate of the edge (i, j) considering each sink at a time. Variables z_{ij} correspond to the global flow rate of each edge (i, j) in our multicast network. The first two restrictions, (3.1) and (3.2), of the algorithm stated before exist to ensure that the flow rates z_{ij} and x_{ij} do not exceed the capacities c_{ij} of the corresponding edges, respectively. The third restriction (3.3) sets z_{ij} as the maximum value of all the x_{ij} and guarantees that all the flow rates are non-negative values. The last restriction (3.4) tracks the data through the network, considering each sink at a time, calculating the sum of each node data, adding the incoming flow rates and subtracting the flow rates of the data that leaves the node. The result of the sum is set to zero, except for the virtual source node S^* and the current sink, which receives an amount of data R and does not forward it. We assume that every other node just forwards the exact amount of data it receives. It is important to emphasize that this behavior is not the one from our multicast network, but from a network with just one sink because we consider just one sink at a time.

When we solve this minimization problem for a given multicast network with one of the linear optimization algorithms mentioned in section 3.3, we get for each directed edge (i, j) a value for z_{ij} , which is the global optimal flow rate that allows all the data to be conveyed from both sources to all the sinks, and several x_{ij} values (as many as the number of sinks considered) that represent the optimal flow rate for edge (i, j) considering only one specific sink and assuming that all the other nodes are ordinary nodes. Although the global values are the ones that allow us to obtain the global subgraph, the particular values for each sink will also be useful in our implementation to extract a more simple subnetwork to use in the construction of the network model for each sink.

It is fundamental to understand how to guarantee that the Slepian-Wolf conditions for distributed source coding [1] are fulfilled in this algorithm for two correlated sources. A virtual source S^* is added to the graph and connected via virtual edges to each actual source. These two virtual edges have their capacities set to the marginal entropy of the corresponding source as we can see in Fig. 3.2(b). The variable R , the overall rate from the virtual source to each of the sinks, is set to the joint entropy of the two correlated sources, because that is the minimum amount of data that needs to reach each sink to make it possible to reconstruct all the original source values. This way we ensure that the rate for both sources falls inside the Slepian-Wolf rating region [1] as we show in Fig. 3.2(a).

3.2 Extension to S Sources

One of the first problems we had to solve when we started the implementation of our model was the fact that the subgraph construction method we described before is only defined for two correlated sources. We wanted to work with a higher number of correlated sources so we had to adapt the problem formulation to our scenario.

As explained in the previous section, in [24] the scenario with two correlated sources guarantees that the problem meets the Slepian-Wolf constraints for distributed source coding by setting

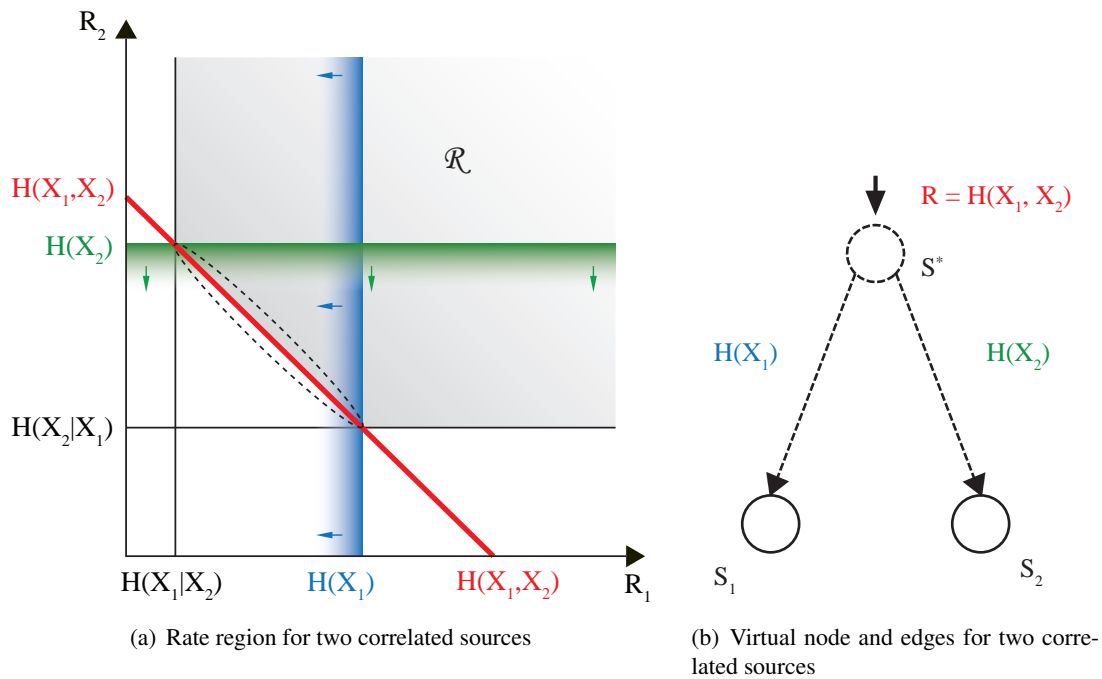


Figure 3.1: Rate region and virtual node and edges for two correlated sources: the capacity constraints and the value that we set to the overall rate in (b) result in the rate region in (a).

the capacity of the virtual edges to the marginal entropy of the corresponding sources and the overall rate R to the joint entropy of the sources as illustrated in Fig. 3.1(b).

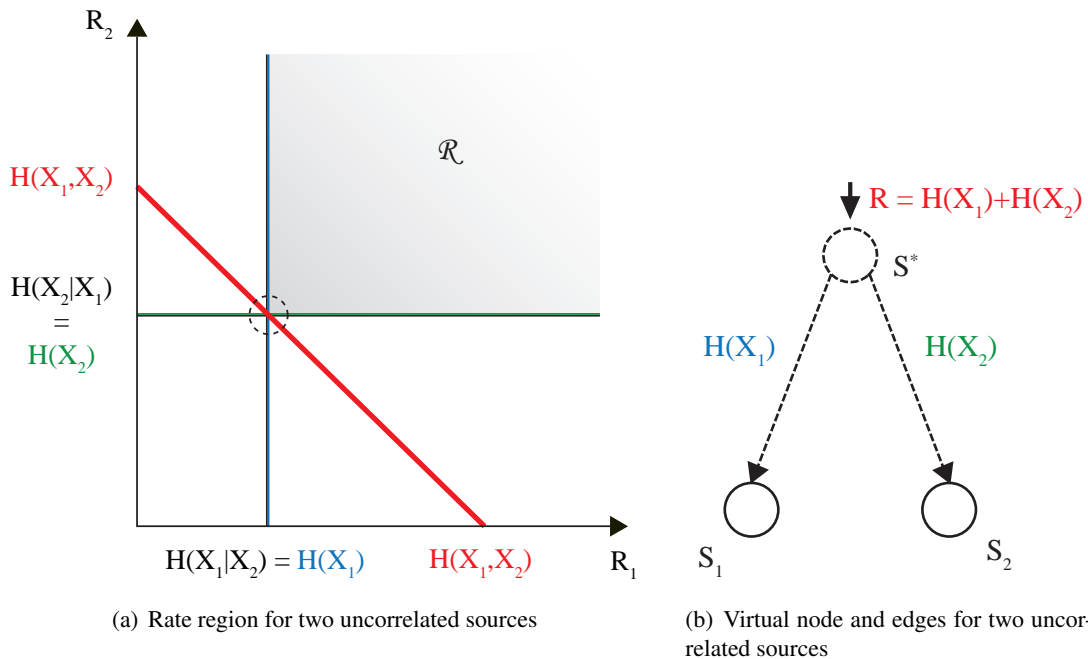


Figure 3.2: Rate region and virtual node and edges for two uncorrelated sources: the capacity constraints and the value that we set to the overall rate in (b) result in the rate region in (a).

On the other hand, if we were considering a case where we had two uncorrelated sources, $H(X_1|X_2) = H(X_1)$ and $H(X_2|X_1) = H(X_2)$. Thus, the possible rate region, as illustrated in Fig. 3.2(a), would be a single point instead of a line because, since the sources do not have correlation to exploit, we would always have to transmit all the information to be able to recover the original data. The only difference in the subgraph construction model formulation would be that the value we set to the overall rate, which is the joint entropy of the sources, is equal to the sum of the marginal entropies (Fig. 3.2(b)). In this case, as there is no correlation between the sources, the model could be used not just for two but for any number of uncorrelated sources as shown in Fig. 3.3.

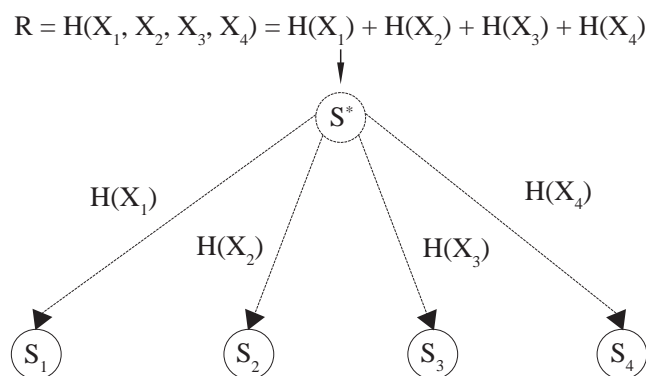


Figure 3.3: Virtual nodes and edges for four uncorrelated sources

We propose a method to increase the number of sources in the subgraph construction model. We combine the two situations concerning correlation between the sources presented before. We can pair the sources using the source-optimized clustering model described in [23]. Because the subgraph construction method described in the previous section is only defined for two correlated sources, we applied the cluster optimization algorithm with a maximum cluster size two. Within each cluster we shall use the model with two correlated sources. Between the clusters, we will assume an uncorrelated model, thus ignoring the correlation and considering only the most important component, namely the correlation inside the clusters.

We will now explain this process in more detail, represented in Fig. 3.4, to modify the subgraph construction model so that we can use any number of correlated sources. We add a virtual cluster node for each cluster created using the source-optimized clustering model with maximum cluster size two. We link each cluster node to the nodes of the actual sources that belong to that cluster and set the capacity of each virtual edge to the marginal entropy of the corresponding source. We create the virtual source S^* as we would in the usual model but instead of connecting the virtual source S^* to the source nodes we connect it to the virtual cluster nodes. The capacity of those virtual edges is set to the joint entropy of the sources from each cluster, which is the marginal entropy of the source if the cluster has size one. The overall rate R is set to the joint entropy of all of the sources, which in this uncorrelated cluster model corresponds to the sum of all the previously mentioned joint entropies of the clusters. It is important to point out that all the virtual edge

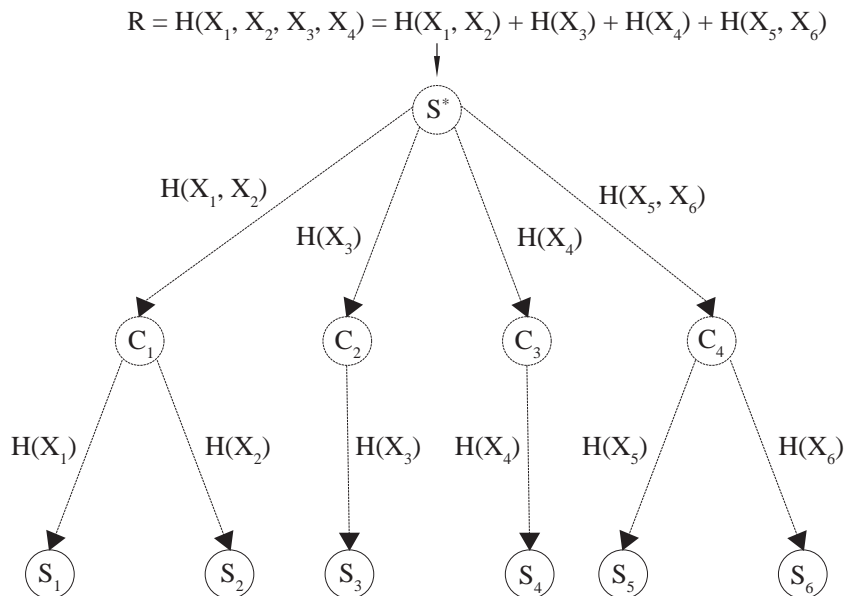


Figure 3.4: Virtual nodes and edges for six correlated sources in four uncorrelated clusters

weights are set to zero because these edges are not to be considered in the minimization function, since they are not real edges.

3.2.1 Possible Generalizations

We considered several other options to solve the problem of the generalization of the subgraph construction method to an arbitrary number of correlated sources. We used clusters of size two because it is simple yet restrictive. We chose to start by exploring the method described in the previous section because, although the other ideas could be seen as alternatives, they would work better as complements to our proposed solution.

The first natural approach is to use virtual nodes recursively with no more than two outgoing edges. We would start at a virtual source S^* which would be connected to two children nodes, each of which could also have two children nodes and so on until the number of children nodes would be equal to the intended number of source nodes. This approach has two main drawbacks, one related to the implementation and the other related to the overall complexity. The first one would be how to guarantee that the flow rates obtained with this method lie within the admissible rate region. The other would be the high number of virtual nodes and edges we would have to add to the graph. This would increase the complexity. The final graph for 2^N sources, would have $\sum_{s=1}^N 2^s$ virtual edges and $\sum_{s=0}^N 2^s$ virtual nodes. This approach on its own is probably not the best choice but it could be used in combination with the one we adopted. By ignoring all the correlation between the clusters our model might be oversimplified but, on the other hand, considering all the correlations without exception turns out to be too complex. For future studies, considering a new

model based on the previous two as a trade-off between the correlations used and the complexity obtained seems promising.

The other option we considered and the two options presented previously are not mutually exclusive. In fact they could be combined. The suggestion is to study the theoretical bounds for specific a number of sources. As we have already a solution for two correlated sources, the idea would be to study the differences in the model if applied to three correlated sources instead. Having extended the model for three, we could continue the study increasing gradually the number of correlated sources, depending on the results obtained. It may be possible to find an abstraction for our model that is applicable to an arbitrary number of correlated sources.

Our proposed model works for an arbitrary number of sources with a low additional complexity, thus fulfilling our main goal. Although some further and more detailed study about every method is necessary, we believe that the ideal method would be a combination of the three models, which could be flexible and adapted to each specific configuration.

3.3 Linear Optimization

The subgraph construction method we use is based on a linear optimization problem. This type of problems can be solved using the simplex algorithm, developed by Dantzig in 1949 [25], which finds the optimal solution by progressing along points on the boundary of a polytopal set. Interior point methods can also be applied to find the optimal solution in linear optimization problems, for example Karmarkar method [26], which is a polynomial time algorithm.

Chapter 4

Coding Approach

4.1 Network Model

The parameters which define our network are N , which is the total number of nodes in the network, the set of source nodes $\mathcal{S} = \{1, 2, \dots, S\}$, the set of sink nodes $\mathcal{T} = \{1, 2, \dots, T\}$, the positions of the nodes and the transmission range r . In our case, we distribute the nodes randomly in a unit square but the positions of the nodes can be defined in a different way. Given the positions of the nodes and the value of the transmission range r , we define the network links as the connections between the nodes whose Euclidean distance does not exceed r . We must define a directed graph because we need to know which amount of data travels in each direction of the connections. As we require a directed graph and the links between the nodes are bidirectional, we need to define two edges for each pair of nodes, one in each direction. To each edge (i, j) we assign a capacity c_{ij} and a weight w_{ij} .

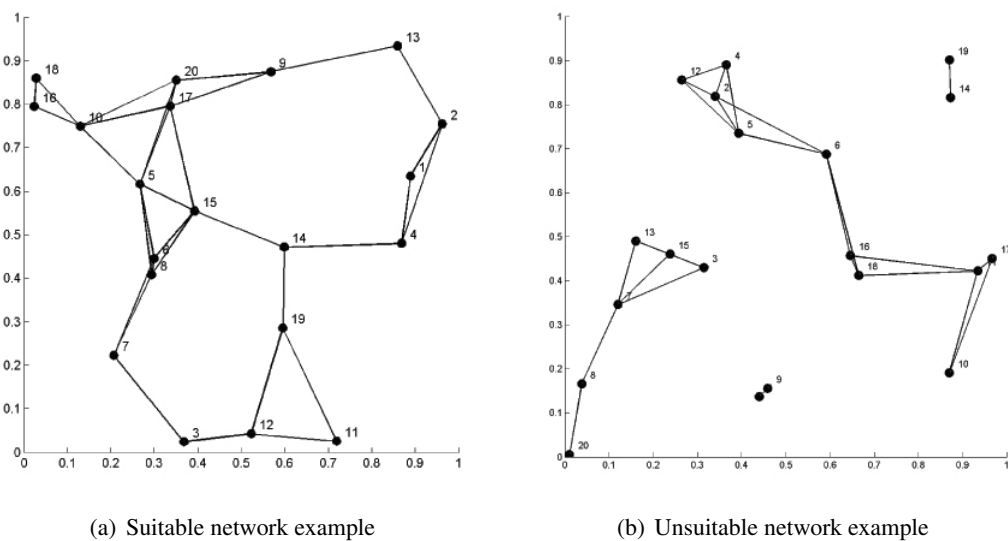


Figure 4.1: Examples of networks with 20 randomly distributed nodes and a 0.3 transmission range: (a) suitable network and (b) unsuitable network.

The network is suitable to be used in our scenario if there is a possible way of transmitting the information from the source nodes to all the sink nodes. It is important to point out that, unless we set the capacity to infinity, it can happen that, although there is a path between a source and a sink, because of the capacity constraints, the resources we have do not suffice to transmit all the information we need. This would also result in a network which is unsuitable for our purposes. In Fig. 4.1 we have examples of suitable 4.1(a) and unsuitable 4.1(b) networks. For a suitable network we intend to find the best path for all the source nodes to convey their data to all of the sink nodes, as well as the optimal flow rates used to do so, which should be within the bounds of the admissible coding rate region characterized in [27]. In order to obtain the optimal flow rates, we will use the subgraph construction method described in detail in [24] and addressed in the previous chapter.

The network model is a representation of the packets' paths through the network. It will allow us to construct the network model factor graph for a specific sink. This representation consists in tracking all the messages that leave the sources until they reach the current sink. We assume that in each time instant each message leaves its current node and arrives at the ones directly reached by that node.

Our network model is represented in a two dimensional array which can have a vector of nodes in each entry. Each line of the array corresponds to a time instant and the columns represent the messages. Different messages are combined into a new message if they arrive at the same node at the same time instant, as we have assumed that every node in the network possesses the capacity of not only forwarding the messages it receives but also of combining them using coding operations. Each line contains all the nodes that receive messages in the corresponding time stamp, starting in the first line in time instant $t=0$ with all the source nodes. The second line, corresponds to $t=1$ and has for each column all the nodes that the corresponding sources can reach directly. If more than one message arrives at a given node at the same time, except the sink, they are combined and regarded henceforth as a single message, thus creating a new column. The same is done for all timesteps t until all the messages reach the sink node.

The result of this procedure is illustrated in Fig. 4.3 for the network showed in Fig. 4.2. In the simple network from Fig. 4.2, we have three source nodes that intend to transmit their data to the sink node. We begin by initializing the source nodes with their messages in time $t=0$. We have three different messages but four packets leaving the sources after that time instant because node 2 can reach two different nodes. In time $t=1$, the message from node 1 reaches node 4, one message from node 2 reaches node 5 and two messages originated from nodes 2 and 3 arrive at node 6, being combined in one single packet from that moment on. Node 4 forwards its message to nodes 7 and 8 and node 5 to nodes 7 and 9. The message from node 6, which is a combination from the original packets from nodes 2 and 3, is sent to the sink in $t=1$, because node 6 is directly connected to node 11, arriving there at $t=2$. At the same time instant, node 8 receives a message from node 4, which it forwards to node 11, reaching the sink in $t=3$. Node 7 combines both messages received in $t=2$ and sends the new message to node 9. Node 9 receives a message from node 5 at $t=2$ and another one from node 7 at $t=3$. It forwards both of them to node 10, which also forwards them to

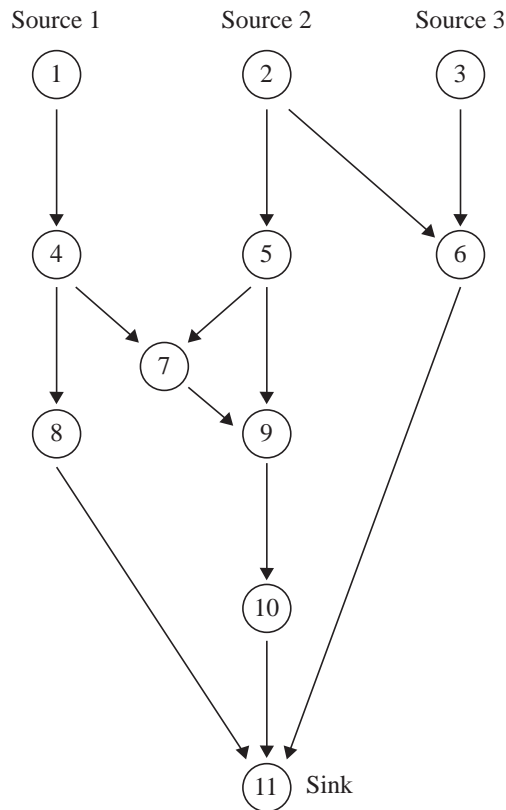


Figure 4.2: Network example

the next node, which is the sink, so the first of those two packets arrives at node 11 at $t=4$ and the last one at $t=5$. Each of the four messages that reach the sink node 11 contains information about one or more of the sources. With the combined information from all of them, the decoder will try to recover the original data sent by all the sources.

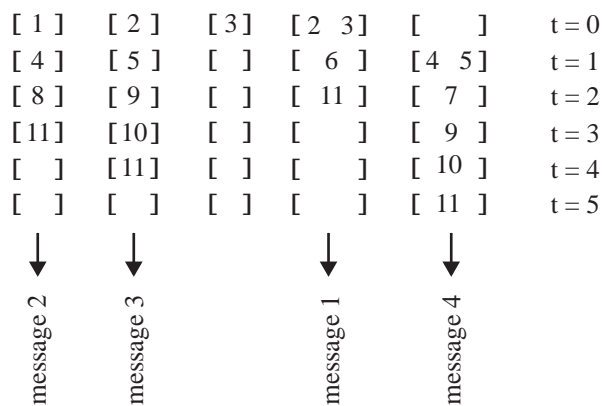


Figure 4.3: Representation of the network model example

In this representation we have the entire path for each of the messages that arrive at the sink, including their origins and all the sequence of nodes visited. Combined with the information of the flow rates for the network edges, this allows us to construct a factor graph that represents our network scenario, as described below.

We start by creating a variable node for each source and another for each message that arrives at the considered sink. Then, for each message, we look at the corresponding column in the network model array and track the path backwards. We start at the last edge (the one that terminates in the sink) and look at the corresponding flow rate, which is the alphabet size of the message's variable node. We proceed to the edge immediately before in the path and compare its flow rate to the previous one. If it is the same, we continue to the subsequent edge with the same procedure. If it has changed, we create a function node with a number of input and output levels equal to the current and previous flow rates, respectively. The outgoing edge of the created function node is connected to the already existent variable message node, and the incoming edge is connected to a newly created variable node. This variable node has alphabet size equal to the current flow rate. If there are no combinations of messages through the path, we repeat this procedure until we reach the last edge (the one that starts at the source node), situation when we connect the incoming edge of the function node to the corresponding variable source node already existent, instead of creating a new variable node. If during the procedure there is a combination of messages in an intermediate node, we create a function node with as many incoming edges as the number of messages combined. We remain with one outgoing edge. We search, in the network model array in the same line of the combination, for all the original messages that originated this one and use the same procedure we were applying to the sink messages to each one of them. This process ends when all the messages are connected to the corresponding variable source nodes, composing a single factor graph that represents the network coding operations for the considered sink.

The design of the coding functions that the network nodes perform is presented in section 4.3.

4.2 Decoding Model and Algorithm

4.2.1 Factor Graph Representation

In previous sections we explained how the source and network models are constructed. Now we will show how to combine both models in a single factor graph to obtain a joint source-network model. As explained in [21] using a factor graph offers the great advantage of allowing for a scalable decoding solution in large-scale networks. The standard implementation of the optimal MMSE decoder is unfeasible in these cases, because the complexity grows exponentially with the number of nodes in the network. The factor graph [18] enables us to use the sum-product algorithm [18], which can make the complexity grow linearly with the size of the network [21].

We construct a decoder factor graph for each sink. The network that is used to construct the network model for each sink is not the original network we defined. With the result of the linear optimization problem, using the optimal values x_{ij} , we construct a subgraph with only the

directional edges which have positive flow rates for a given sink. The resulting subnetwork is much simpler than the original one and specific for each sink, as is the factor graph. Each factor graph is constructed based on the global flow rates determined by the linear optimization problem and the messages' paths obtained from the network model representation for each sink. Thus, we have all the information we need to make the nodes' connections and exploit their correlations. The optimal flow rates calculated with the subgraph construction method are used to determine the factor graph functions dimensions.

The decoder factor graph is constructed by a process similar to the one we described when creating the network factor graph, in the previous section. Beginning with one sink, a variable node is created for each of the messages that reach that sink. We track each message backwards, creating a function node every time there is a combination of messages as well as every time the flow rate is altered, until every message connects to a source node. The only difference is that, after the network model is complete, we add the functions that contain the information about the correlation among the sources within clusters with size two. We create a function node that represents the probability mass function between each two source nodes that belong to the same cluster. The variable nodes' positions are given from the ones in the network so that we can relate the nodes in the factor graph to the ones in the network, since the numeration is completely different. For each sink, we construct a factor graph representing both the network operations and the source correlations, in order to jointly decode the messages that reach that sink.

Using the values of the messages that arrive at each decoder, we initialize the nodes that represent those messages in the corresponding decoder factor graph. We run the decoder factor graph and, after a sufficient number of iterations using the sum-product algorithm as a message-passing algorithm, the messages will reach the source nodes. As the factor graphs constructed for the decoder have cycles, we have to define a criterion to stop the iterations. To guarantee that the messages reach the source nodes before the iterations stop, we set the total number of iterations to the total number of nodes in the factor graph so that the messages reach all the factor graph nodes at least once.

The messages that are obtained in the source nodes from the factor graph after the simulation correspond to the quantized values the decoder calculates for each source. To make the estimates of the original continuous source values, the decoder performs the inverse quantization to the one performed in the corresponding encoder (see Quantizer in System Setup), thus obtaining the reconstruction levels of the corresponding quantization region.

4.2.2 Sum-Product Algorithm

In this section we will present the sum-product algorithm for decoding and how it runs on our factor graphs. It can be used to compute the marginals of all nodes efficiently and exactly, if the factor graph is a factor tree. It can also be run in factor graphs with cycles albeit without guaranteeing convergence to the MMSE. The algorithm involves passing messages on the factor graph. A message is a vector of length L , where L is the number of possible states a node can take.

The message sent from function node f to variable node v is denoted as $\mu_{f \rightarrow v}(v)$.

The message sent from variable node x to function node f is denoted as $\mu_{x \rightarrow f}(v)$.

The messages are defined recursively. In particular, consider a factor f_s that involves (connects to) a particular variable v . Denote the other variables involved in f_s by v_1, v_2, \dots, v_M . Also, let $n(v_m)$ denote the set of factors connected to v_m . To calculate the messages there are the two following rules

$$\mu_{f_s \rightarrow v} = \sum_{v_1} \dots \sum_{v_M} f_s(v, v_1, \dots, v_M) \prod_{m=1}^M \mu_{v_m \rightarrow f_s}(v_m), \quad (4.1)$$

and

$$\mu_{v_m \rightarrow f_s} = \prod_{f \in n(v_m) \setminus f_s} \mu_{f \rightarrow v_m}(v_m). \quad (4.2)$$

The recursion is initialized as follows. When the factor graph is a tree, we can pick an arbitrary node and call it the root. This defines all the leaf nodes, in which all the messages are initialized. If a leaf is a variable node v , its message to a function node f is $\mu_{v \rightarrow f}(v) = 1$. If a leaf is a function node f , its message to a variable node v is $\mu_{f \rightarrow v}(v) = f(v)$.

A node, either function or variable, can send out a message if all the necessary incoming messages have arrived. The factor tree will pass messages once in each direction (towards and away from the root node). The product of the messages into any variable will give the exact marginal distribution for that variable.

When the factor graph contains loops, i.e. is a factor graph with cycles, there is no longer guarantee that the algorithm will even converge. However, applying the sum-product algorithm without regard to the cycles has proven to be a successful strategy [18]. This way of using the sum-product algorithm is known as loopy belief propagation. A more detailed description of the iterative processing of the sum-product algorithm in factor graphs with cycles can be found in [18].

Regarding message-passing schedules, we assume that messages are synchronized with a global discrete-time clock, with at most one message passed on any edge in any given direction at one time. Any such message effectively replaces previous messages that might have been sent on that edge in the same direction.

As described in [18], we say that a node v has a message pending at an edge e if it has received any messages on edges other than e after the transmission of the most previous message on e . Such a message is pending since the messages more recently received can affect the message to be sent on e . The receipt of a message at v from an edge e will create pending messages at all other edges incident on v . Only pending messages need to be transmitted, since only pending messages can be different from the previous message sent on a given edge.

Assuming a schedule in which only pending messages are transmitted, in cycle-free factor graphs the sum-product algorithm will eventually halt in a state with no messages pending, providing the exact function summaries. However, in factor graphs with cycles, it is impossible to reach that state because the transmission of a message on any edge of a cycle from a node v will

trigger a chain of pending messages that must return to v , causing v to send another message on the same edge. This process continues indefinitely, until some stopping criterion is satisfied or until the available time is over. The results obtained in this way are expected to be adequate approximations of the original values.

4.3 Code Design

In order to construct the factor graph for the decoder, the coding functions at the encoders and the intermediate nodes have to be known. Being h the coding operation performed, function h has for inputs the values of the incoming edges of the node where the operation is performed, represented by k_1, k_2, \dots, k_i . Each of these values has alphabet sizes l_1, l_2, \dots, l_i , respectively. The result of performing h is the value k_o that the node forwards to its outgoing edges and has a number of levels represented by l_o , such that $k_o = h(k_1, k_2, \dots, k_i)$, $k_o \in [0, 1, \dots, l_o - 1]$ and $k_j \in [1, 2, \dots, l_j], j \in [1, 2, \dots, i]$.

Although the design of those coding functions is not a decoder specific problem, we propose three different coding strategies. The first one is based on random mappings, the second uses a sequential index assignment strategy and the third one employs a systematic joint source-network coding concept. We shall describe each of them in the next subsections, showing the same two examples of the output mapping table representing h for each one. The mapping table is a matrix with as many dimensions as the incoming edges, i , each dimension with length equal to the number of levels of the corresponding edge, l_1, l_2, \dots, l_i . Each entry of the matrix is the output that corresponds to a unique combination of input values.

The following mapping table examples have two inputs with alphabet sizes $[4, 4]$ ($l_1 = 4$ and $l_2 = 4$) and $[4, 8]$ ($l_1 = 4$ and $l_2 = 8$), respectively. The output alphabets have size 4 ($l_o = 4$) and 8 ($l_o = 8$).

4.3.1 Random Mapping Function

If h is a random function each combination of its input values is determined randomly between 0 and the maximum output value, which is the output alphabet size minus one. This means k_o is a number between 0 and $l_o - 1$, determined in a random way for each combination of k_1, k_2, \dots, k_i . The matrices **A** and **B** are examples of mapping tables for random mapping functions.

For small alphabet sizes this method can lead to situations we want to avoid, such as having output values that are never used, thus wasting bits, or having a high variation in the distribution of the output values, making it more difficult to determine some combinations of inputs over others.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 & 2 \\ 3 & 0 & 2 & 1 \\ 2 & 3 & 2 & 2 \\ 3 & 3 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 2 & 5 & 3 & 1 & 5 & 5 & 7 & 5 \\ 0 & 2 & 3 & 3 & 5 & 1 & 2 & 2 \\ 1 & 7 & 5 & 3 & 2 & 1 & 4 & 4 \\ 6 & 0 & 6 & 5 & 5 & 3 & 2 & 5 \end{bmatrix}$$

4.3.2 Sequential Mapping Function

If h is the sequential function, each combination of the input values is determined in a sequential way starting at 1 and adding one in each value. Afterwards it is calculated each value modulo the maximum output value. The function h can be represented by the operation $p \bmod l_o$, being p the linear index of the mapping table, such that $p \in [1, 2, \dots, l_1 \times l_2 \times \dots \times l_i]$. The sequential mapping function is used to construct the mapping tables **C** and **D**.

The advantage of this function compared to the random one is that we can guarantee that the distribution of the output values is close to uniform, preventing the situations mentioned before where bits could be wasted.

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 1 & 5 & 1 & 5 & 1 & 5 & 1 & 5 \\ 2 & 6 & 2 & 6 & 2 & 6 & 2 & 6 \\ 3 & 7 & 3 & 7 & 3 & 7 & 3 & 7 \\ 4 & 0 & 4 & 0 & 4 & 0 & 4 & 0 \end{bmatrix}$$

4.3.3 Joint Source-Network Coding Mapping Function

This function was developed as a systematic code for joint source-network coding. It has the same advantages of the sequential function as it guarantees that all the output values are equally distributed as long as possible, but it contains extra benefits in the way the coding is done. The matrices **E** and **F** represent two different mapping tables for this function. The performed coding operation is described in detail in the next paragraph.

$$\mathbf{E} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 0 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 1 & 5 & 0 & 4 & 3 & 7 & 2 & 6 \\ 2 & 6 & 3 & 7 & 0 & 4 & 1 & 5 \\ 3 & 7 & 2 & 6 & 1 & 5 & 0 & 4 \end{bmatrix}$$

We start by calculating the minimum value of the input alphabet sizes: $l_{min} = \min(l_1, l_2, \dots, l_i)$. We then produce a matrix with the results of the addition in finite fields of the form 2^f , being f a positive integer, for i dimensions with l_{min} levels each. This operation is equivalent to bitwise exclusive OR operation of the input values so we will call it XOR matrix from now on. We chose it because the number of occurrences of the different values is uniform, and each appears once in each array of each dimension. The matrix **E** is an example of this XOR matrix for two inputs with $l_{min} = 4$. After that, we replicate this XOR matrix in every direction to achieve the correct number of alphabet sizes for all the inputs. When the matrix is first constructed, the resulting alphabet size of the output is the same of the inputs, i.e. l_{min} . If the alphabet size required in the output does not match the one in the matrix, a sum or modulo operation is performed, depending on the need to increase or reduce the number of output levels. If we want a lower alphabet size, we perform the modulo operation on all the values, with the intended alphabet size value l_o , so that every value is between 0 and $l_o - 1$. When we want a higher alphabet size, we first have to calculate how many times the intended alphabet value is higher than the one we currently have: $p = l_o / l_{min}, l_o > l_{min}$. As both l_{min} and l_o can be written in the form 2^f , being f an integer value, p should be an integer value. We divide the matrix in p equally distributed parts and add $l_{min} \times [0, 1, \dots, p - 1]$ to the corresponding part, so that the matrix has all its values between 0 and $l_o - 1$. We subsequently reorder the arrays in the larger dimension, so that the repeated values have the larger possible distance between them in all directions. The purpose of this aim is that the combinations that result in the same value are as distant and different from each other as possible, because with the extra information we get from the correlations we are likely to have an area in which the probability of that value is higher. If the distance between the potential values is sufficiently large compared to the area, we can discard the values that fall outside the expected area and be more accurate in our estimate. If the potential values were very similar, many of them would fall inside the expected area and the information from the correlation would not be very helpful in deciding by one of the values over the others, since they would all have high probabilities of happening.

These are the reasons why we built this type of function and expect it to have a high performance when used for source-network coding operations in a scenario in which the correlation between the sources is explored.

To present an illustrated example of the operation described before, we can explain the procedure and obtain the matrix **F**, showing the resulting matrices of every step. The first matrix, equal to **E**, is the basic XOR matrix for two inputs with alphabet size four and resulting in an output with the same alphabet size. The second matrix is constructed by replicating the first one in the second dimension (i), as the second input has alphabet size eight. By replicating the matrix, the output alphabet size remains four. As we want an alphabet size of eight, which is twice the current value, we add the value of the current alphabet to the second half of the matrix (ii), i.e. we add four to the values in the right side of the matrix. We subsequently reorder the arrays in the larger dimension (iii), so in this case we reorder the columns, as we can see in the resulting matrix.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \xrightarrow{(i)} \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 & 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 & 3 & 2 & 1 & 0 \end{bmatrix} \xrightarrow{(ii)}$$

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \end{bmatrix} \xrightarrow{(iii)} \begin{bmatrix} 0 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 1 & 5 & 0 & 4 & 3 & 7 & 2 & 6 \\ 2 & 6 & 3 & 7 & 0 & 4 & 1 & 5 \\ 3 & 7 & 2 & 6 & 1 & 5 & 0 & 4 \end{bmatrix}$$

In this chapter we proposed a model for the decoder. In the next chapter we will present a working example of our approach and analyze its complexity and performance.

Chapter 5

Results

5.1 Working Example

In this section we will illustrate all the steps of the whole process described in the previous chapters.

We start by defining some parameters that we used in this simulation. One of the parameters we need to define is the number of source nodes considered in the network. In this simulation we set it to 8. Another important aspect is the correlation between the sources. We assume that sensor measurements $\mathbf{u} = (u_1, u_2, \dots, u_S)^T$ are distributed according to a multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}_S, \mathbf{R})$, where the correlation between a pair of sensors u_k and u_l decreases exponentially with the distance $d_{k,l}$ between them, such that $\rho_{k,l} = \exp(-\beta \cdot d_{k,l})$. We set the coefficient describing how correlation decays exponentially with Euclidean distance β to 0.5 to represent correlated sensor measurements. There are also some parameters we need to define for the quantizer. The first one is the type that, as we mentioned before, was set to non-uniform Lloyd-Max quantizer to minimize the distortion as the measurements \mathbf{u} are Gaussian distributed. The second one is the resolution, i.e. number of bits used in each quantizer, which in this simulations was set to 3, resulting in quantizers with 8 levels. We need to define what type of functions to use in the index assignment performed on each encoder after the quantization. In this simulation we use Diophantine index assignments for distributed source coding described in detail in [22]. To represent the communication links, we defined a network with 20 numbered nodes randomly distributed in a unit square area (Fig.5.2(a)). To identify the source nodes and the sink nodes, we classify the first 8 nodes as the sources and the last 3 nodes as the sinks. To define the network edges, we use a 0.3 transmission range, resulting in the network showed in Fig.5.2(a).

In Fig. 5.1 we show the factor graph obtained after source-optimized clustering with a maximum cluster size of 2 nodes, for the 8 source nodes of our illustrated example.

Having the original network defined by its nodes and edges (Fig.5.2(a)), we set a capacity constraint of 8 bits per edge. The reason for this value is that we do not want to have high flow rates in the edges due to communication constraints. On the other hand, very low values of the capacity constraints may be insufficient for transmission. For the weights of the edges we chose to

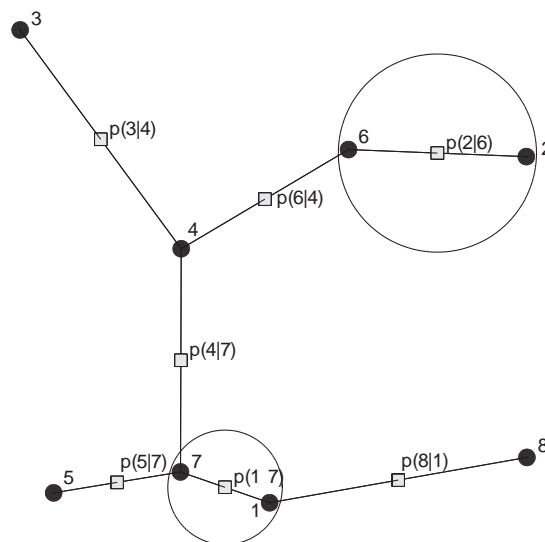


Figure 5.1: Illustrated example: source-optimized clustering

use a function of the distance, although other options would be possible. In this simulation we set the weight of an edge to ten times the Euclidean distance between the two nodes that define that edge, obtaining weights in the range $[0, 3]$.

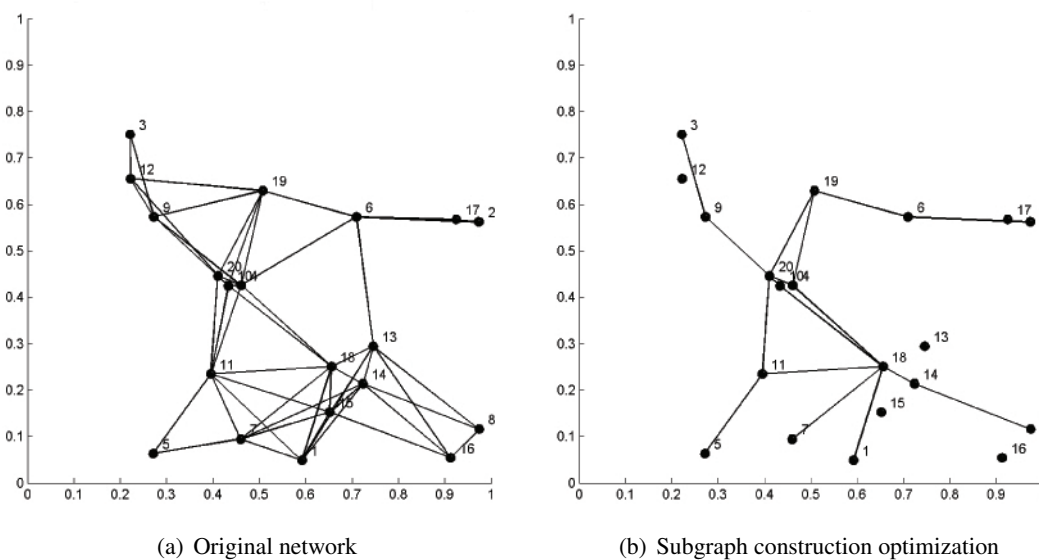


Figure 5.2: Illustrated example: (a) original network with 20 randomly distributed nodes with a 0.3 transmission range and (b) the same network after the subgraph construction optimization for a capacity constraint of 8 bits per edge.

Running the subgraph construction algorithm presented in Chapter 3 on the defined network shown in Fig. 5.2(a) we obtain the optimal flow rates for all the edges. Using those flow rates we can create the simpler network shown in Fig. 5.2(b). This one contains only the edges from the original network that are actually used in the data transmission to the sinks (positive flow rates).

Table 5.1: Flow rate values

Parameter	Value (bits)
Overall flow rate	20.4518
Joint entropy of nodes 2 and 6 (cluter 1)	4.7320
Joint entropy of nodes 1 and 7 (cluster 2)	4.4199
Source 1 flow rate	2.8250
Source 2 flow rate	1.9070
Source 3 flow rate	2.8250
Source 4 flow rate	2.8250
Source 5 flow rate	2.8250
Source 6 flow rate	2.8250
Source 7 flow rate	1.5949
Source 8 flow rate	2.8250

Some relevant flow rate values are presented in Table 5.1.

As an example we will consider the process for the first sink $t_1 \in T$, i.e. the optimization results considering just node 18 as the sink. In Table 5.2 we present the positive optimized flow rates specific for sink node 18, as well as the number of bits and levels actually needed for each of those edges.

Table 5.2: Illustrated example: optimization results for sink t_1

Edge	Flow rate	Number of bits	Number of levels
1 → 18	2.8250	3	8
2 → 6	1.9070	2	4
3 → 9	2.8250	3	8
4 → 18	5.2236	6	64
20 → 4	2.3986	3	8
5 → 11	2.8250	3	8
6 → 19	4.7320	5	32
7 → 18	1.5949	2	4
8 → 14	2.8250	3	8
9 → 20	2.8250	3	8
10 → 18	5.1584	6	64
20 → 10	5.1584	6	64
11 → 18	2.8250	3	8
14 → 18	2.8250	3	8
19 → 20	4.7320	5	32

With the new simplified list of edges, the next step is to construct the network model as described in the first section of Chapter 4. In Fig. 5.3 we have the representation of the network model for sink node 18. We can see that this sink receives 9 messages. Almost all those messages contain information about only one source node, except the combination from nodes 3 and 6 that happens in node 20 in time instant $t=2$.

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[19]	[2]	t=0
[18]	[6]	[9]	[18]	[11]	[19]	[18]	[14]	[9]	[19]	[6]	t=1
[]	[19]	[]	[]	[18]	[]	[]	[18]	[20]	[20]	[19]	t=2
[]	[20]	[]	[]	[]	[]	[]	[]	[4]	[10]	[20]	t=3
[]	[4]	[]	[]	[]	[]	[]	[]	[18]	[18]	[10]	t=4
[]	[18]	[]	[]	[]	[]	[]	[]	[]	[]	[18]	t=5

Figure 5.3: Illustrated example: network model representation

With the network model from Fig. 5.3 and the number of levels for each edge determined from the flow rates from table 5.2, we construct the decoder factor graph using one of the three types of functions described in the previous chapter. The decoder factor graph is shown in Fig. 5.4. It is important to note that there are many nodes printed in the same position because they represent the same nodes in the original network. Nevertheless they are distinct nodes of the factor graph. As an example, in the sink node’s position, there are 9 variable nodes, each of which represents a message that arrives at sink 18. Nodes numbered from 1 to 27 represent the network nodes (1 to 8 the sources and 9 to 18 the sink messages) and the two pairs of nodes 28 and 29 and 30 and 31 are the ones which represent the source correlations inside the two clusters.

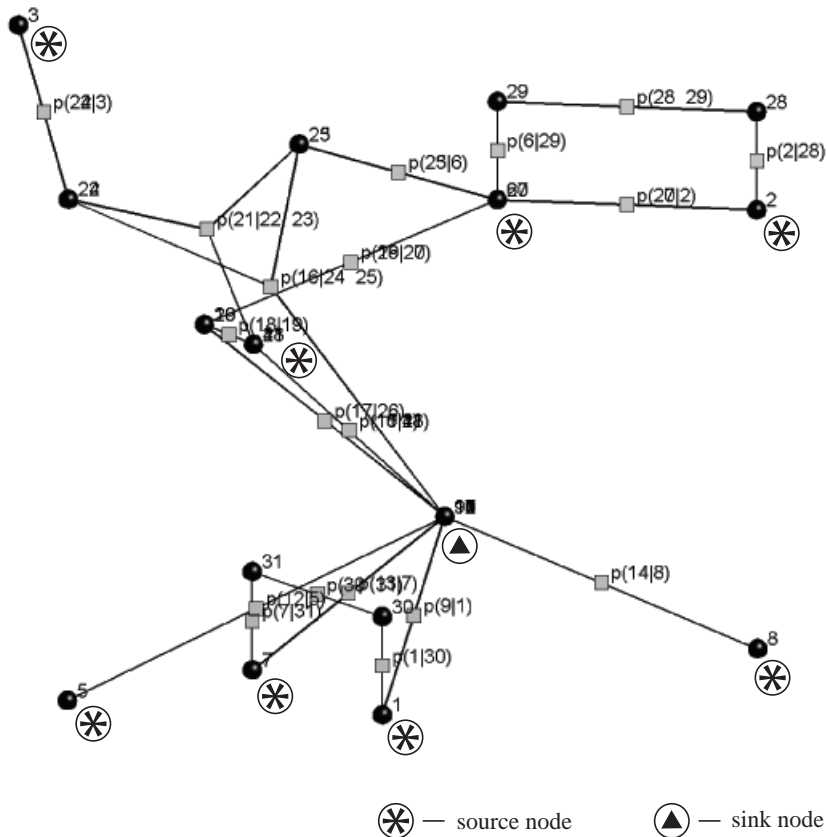


Figure 5.4: Illustrated example: decoding factor graph

The same procedure is also applied to all the other sinks, so that we have a decoder factor graph for each sink.

5.2 Complexity Analysis

Computational complexity theory studies the amounts of resources required for the execution of algorithms. Computational complexity or time complexity is measured by the number of steps that it takes to solve an instance of the problem as a function of the size of the input [28]. Space complexity is equal to the volume of the memory used to solve an instance of the problem as a function of the size of the input [28]. For the complexity analysis we will consider the factor graphs for each sink separately. We will start with the computational and space complexity for the variable and function nodes, using the big O notation [29]. We will then evaluate the number of nodes for both the source and the network models in order to analyze the overall decoding complexity. We study the complexity for one iteration first and then derive the overall complexity for all iterations.

5.2.1 Complexity of Factor Graph Nodes

We start by defining the required parameters for the factor graph nodes and edges. Let d be the degree of a variable or function node and L be the number of levels of an edge, which is upper bounded by 2^c , being c the capacity of the corresponding edge.

Considering a function node, the space complexity is given by the dimension of the PMF (probability mass function) matrix which is $O(L^d)$. Regarding computational complexity, considering the multiplication operations, which are the most relevant, for one outgoing edge we have L^{d-1} multiplications for each value. For all the L values, the computational complexity per outgoing edge is then $O(L^d)$.

Regarding the variable nodes, the space complexity is given by $O(L)$ which is the length of the vector that saves the value of the node. Considering one outgoing edge, the computational complexity is calculated as the normalized sum of all the other edges values so is given by $O((d-1) \times L)$.

Since these complexities are computed per outgoing edge, we have to define the number of outgoing edges for our nodes. As we are looking for an upper bound to the complexity, we can consider the degree d of the node as an estimate for the number of outgoing edges.

5.2.2 Number of Factor Graph Nodes

5.2.2.1 Source Model

In the source model the number of variable nodes is equal to the number of source nodes S . We could use our uncorrelated clusters source model to estimate the number of function nodes as the number of clusters divided by the maximum cluster size but this would be too specific. We will instead use a more general approach that holds for more generic source models, assuming

that the source model can be represented by a tree-like (i.e. cycle free) graph. Thus, the maximum number of function nodes is given by $S - 1$, the maximum number of edges in a tree-like graph with S nodes, since each edge contains exactly one function node.

5.2.2.2 Network Model

We assume that M is the number of messages that arrive at the targeted sink. Usually, the value of M is higher when we decrease the capacity c . If we would set the capacity of the edges to infinity, the value of M would be the minimum possible. Setting the capacity to smaller values, we use alternative paths to reach the sink so the number of messages arriving at the sink increases. In a case where there were no combinations of messages in the intermediate nodes and no alternative paths, the number of messages arriving at the sink would be S . In a real case usually both situations happen, but as one decreases the number of messages (the combination of messages in the intermediate nodes) while the other increases the number of messages (the creation of alternative paths), M is usually similar to S . For now we consider M as a parameter for the complexity analysis.

Considering separately the graph for each message that arrives at the sink, we know that it is a cycle-free graph, which cannot have more nodes than the total number of nodes in the network because the path calculated for each message is optimal, implying it cannot contain repeated nodes. We assume N as the total number of nodes in the network, not the original network but the one obtained by the subgraph construction method presented in Chapter 3. We use N as an upper bound for the number of variable nodes in each message graph. The number of function nodes is upper bounded by the maximum number of edges, which is given by $N - 1$, as the factor graph considering only one message at a time is cycle-free. As the network model includes all the messages that reach the sink, the maximum number of variable nodes is given by $M \times N$ and the maximum number of function nodes is given by $M \times (N - 1)$.

5.2.2.3 Joint Source-Network Factor Graph

The number of nodes in the joint source-network factor graph is the sum of both models. To estimate the total number of variable nodes we can ignore the source model as the network model already includes the source nodes in N . Regarding the number of function nodes in the overall factor graph we can also ignore the source model as $S - 1$ is negligible compared to $M \times (N - 1)$ as N is usually much higher than S .

5.2.3 Overall Complexity

First, we have to estimate the maximum degree of the nodes. Considering the variable nodes, only the source nodes can have a degree higher than two. Concerning source function nodes, the maximum degree is equal to the maximum cluster size, in our case, size two. Regarding the network function nodes, the degree of a node is also two except when there is a combination of messages in a network node. In those cases, the maximum degree would be the maximum number

of messages arriving at a specific node at a specific time plus one (the outgoing edge). This number is not likely to be high because if the capacity was set to infinity there would be almost no alternative paths and consequently the maximum combination of nodes would be the number of sources. If, on the other hand, the capacity is low, we cannot combine many messages in the same node at the same time because of the flow rate constraints. As there is no certain way of calculating the maximum degree of a node we introduce the parameter D as the maximum node degree for both variable and function nodes.

Considering only one iteration we have to sum up the computational complexity of all the nodes in the joint source-network factor graph. Starting with variable nodes, we estimated $M \times N$ nodes with a computational complexity for each node given by $O(D \times (D - 1) \times L) = O(D^2 \times L)$. Considering function nodes we calculated a maximum number of $M \times (N - 1)$ nodes with a computational complexity of $O(D \times L^D)$.

To obtain the overall complexity we have to calculate the total number of iterations performed, which we introduce as I . This value is set to maximum number of iterations needed as we are considering now a factor graph with cycles. The maximum number of iterations is the total number of nodes in the factor graph, given by the sum of the variable and the function nodes, which can be upper bounded by $2 \times M \times N$, so that $I = O(M \times N)$.

The overall complexity is then upperbounded by:

$$\begin{aligned}
 I \times (M \times N \times O(D^2 \times L) + M \times (N - 1) \times O(D \times L^D)) &\simeq \\
 \simeq I \times M \times N \times O(D^2 \times L) + I \times M \times N \times O(D \times L^D) &= \\
 = O(I \times M \times N \times D^2 \times L) + O(I \times M \times N \times D \times L^D) &= \\
 = O(I \times M \times N \times D^2 \times L^D) &
 \end{aligned}$$

The first important conclusion we can get out of this value is that the complexity usually grows linearly with the size of the network N , or in the worst case quadratically. The only parameter that influences the complexity exponentially is L^D . We can limit the value of L by the capacity constraints and the value of D is not usually large as we pointed out before. D can also be limited by protocols constructing the overlay network in the real nodes.

5.3 Performance Analysis

In the previous chapter, we presented our decoder model. It is important to point out that our approach works for arbitrary scenarios. In this analysis, we are only using an exemplary scenario, serving as a proof-of-concept. We will use a network with 30 nodes in this particular case, to verify that our decoder is feasible for large number of nodes.

For our scenario in this analysis we chose the following fixed parameters : 30 network nodes, the number of sources as 20% of network nodes, i.e. 6 source nodes, the number of sinks as 10% of network nodes, i.e. 3 sink nodes, quantizer resolution of 3 bits, corresponding to 8 levels, edge's capacity constraints of 6 bits, corresponding to messages of alphabet size 64 and 20000 samples per source. For the encoder mapping function we used the Diophantine index assignment for distributed source coding described in detail in [22].

The nodes were randomly distributed in a unit square area as illustrated in Fig. 5.5(a). The first six are the source nodes and the last three, i.e. nodes 28, 29 and 30, are the sink nodes. The source-optimized clustering method creates three clusters, grouping nodes 1 and 5, 2 and 4 and 3 and 6 within the same clusters.

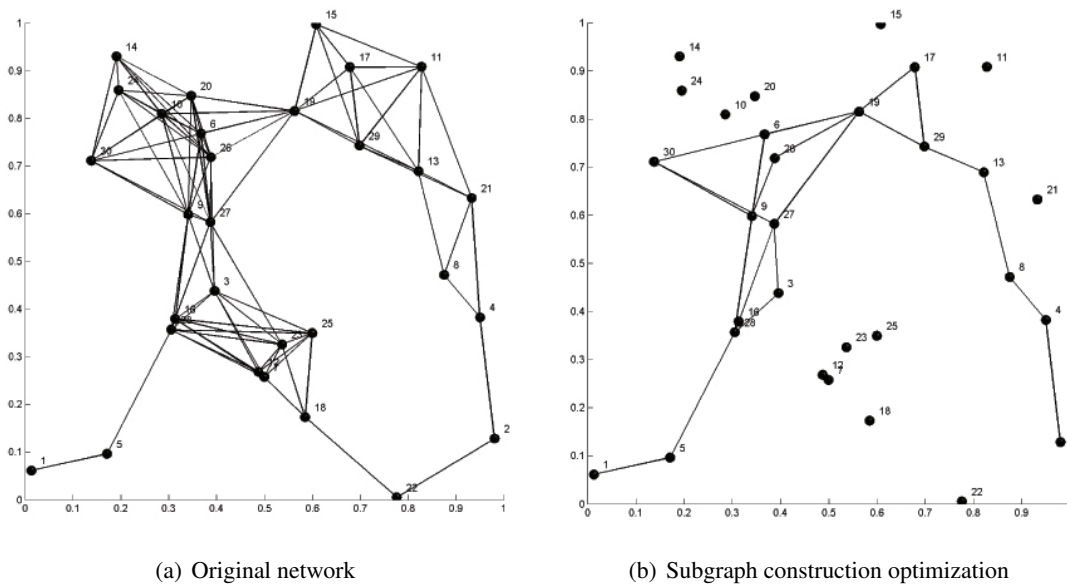


Figure 5.5: Network: (a) original network with 30 randomly distributed nodes with a 0.3 transmission range and (b) the same network after the subgraph construction optimization for a capacity constraint of 6 bits per edge and using a strong correlation scenario with $\beta = 0.25$.

One of the parameters we changed to compare the performance in different cases was the type of function used to perform the coding operations in the intermediate network nodes. We applied the three functions described previously in section 4.3: the random mapping function denoted in the tables by "Rand", the sequential mapping function denoted in the tables by "Seq" and the joint source-network coding mapping function denoted in the tables by "JSNC". The other parameter we altered was β , which is the coefficient describing how correlation decays exponentially with Euclidean distance, to simulate different correlation scenarios.

To evaluate the overall performance of our decoding scheme, we measure the output signal-to-noise ratio (SNR) given by

$$\text{Output SNR} = 10 \cdot \log_{10} \left(\frac{\|\mathbf{u}\|^2}{\|\mathbf{u} - \hat{\mathbf{u}}\|^2} \right) \text{ in dB} \quad (5.1)$$

In our first experiment, we set β to 0.25 to represent strong correlation between the source data. Fig. 5.5(b) shows the results of the subgraph construction method, which yields the transmission rates, for this scenario. The obtained SNR values for the three different mapping functions presented in the previous chapter are showed on Table 5.3. In the tables, the SNR values for all the source-sink pairs are shown and also the total (avareged) value for each sink, i.e. for each decoder. We calculated the 95% confidence interval for these SNR values. For the random mapping the values vary approximately 0.35dB around the presented mean values. Regarding the sequential and the joint source-network coding mapping functions, the variations of the values are approximately 0.25dB.

Table 5.3: System SNR in dB for a strong correlation between the sources ($\beta = 0.25$)

Function	Sink	Sources						Total
		1	2	3	4	5	6	
Rand	1	10.4784	6.6915	7.8733	9.1036	14.5520	7.4397	8.7221
	2	11.4844	5.5647	12.7642	6.6146	14.5520	14.6273	9.3981
	3	11.4033	6.2951	4.1714	5.2648	14.5520	4.9197	6.4959
Seq	1	14.0772	12.9011	14.6625	14.5876	14.5841	11.9301	13.6610
	2	14.0772	12.9011	12.2500	14.5876	14.5841	14.6015	13.7300
	3	14.0772	12.9011	12.2500	14.5876	14.5841	14.6015	13.7300
JSNC	1	14.0772	12.9011	14.6625	14.5876	14.5841	11.9301	13.6610
	2	14.0772	12.9011	12.2500	14.5876	14.5841	14.6015	13.6610
	3	14.0772	12.9011	12.2500	14.5876	14.5841	14.6015	13.7300

Considering weakly correlated sources, we set β to 1.0. The constructed subgraph in this case is similar to the one in Fig. 5.5(b) but also uses the edge from node 26 to node 30, since it needs to transmit more data. The resultant SNR values are shown in Table 5.4. The 95% confidence intervals in the weak correlation case are the following: for the random function we have a variation of approximately 0.32dB around the mean values, for the sequential function 0.16dB and for the joint source-network coding function 0.17dB.

In general, we observe that the weak correlation scenario has higher SNR values. However, it is important to point out that the SNR values in strong and weak correlation cases cannot be compared directly, since also the overall cost of the obtained subgraph in each of the situations is different. The overall cost of the graph is the sum of the edges' costs, which are linearly propotional to their rates (for more details see the description of the subgraph construction method in Chapter ??). In this particular example, for the strong correlation scenario, the total cost of the subgraph is 195.9 and the overall rate, i.e. the total flow rate that reaches each sink, is 16 bits, both values after rounding up every non-interger flow rates since we only allow for integer rates

Table 5.4: System SNR in dB for a weak correlation between the sources ($\beta = 1$)

Function	Sink	Sources						Total
		1	2	3	4	5	6	
Rand	1	8.2345	7.5630	4.5182	7.5893	9.4680	3.5466	6.2958
	2	14.6718	8.6511	14.6070	14.7399	14.6790	14.7043	12.9155
	3	3.8336	10.1837	8.6467	14.7399	5.1001	8.7540	7.2819
Seq	1	14.6360	14.5935	14.6230	14.5538	14.6668	14.5097	14.5968
	2	14.6360	14.5935	14.6230	14.5538	14.6668	14.5097	14.5968
	3	14.6360	14.5935	14.6230	14.5538	14.6668	14.5097	14.5968
JSNC	1	14.6916	14.7033	14.7226	14.6100	14.7235	14.6293	14.6798
	2	14.6916	14.7033	14.7226	14.6100	14.7235	14.6293	14.6798
	3	14.6916	14.7033	14.7226	14.6100	14.7235	14.6293	14.6798

(bit). On the other hand, for the weak correlation scenario, the total cost of the subgraph is 15.6% higher than in the strong correlation scenario, i.e. 226.5. As we were expecting, the overall rate in the weak correlation case is also higher, requiring two more bits, i.e. 18 bits in total. This means that there was no bit saving due to the correlation between the sources, since we have 6 sources with quantization resolution of 3 bits. It is important to point out that the decoder works in both cases with good performance, in particular for the sequential and the joint source-network coding schemes.

From this analysis, we can conclude that our joint source-network coding approach is feasible and works for arbitrary scenarios and arbitrary correlation settings. For all the considered scenarios we observe good system performance.

We also showed that the decoder can be implemented with feasible complexity for large network sizes (in this case $N = 30$), and confirmed that our model is able to decode in situations where a separable approach cannot. In addition, our approach usually leads to rate savings compared to the separable solution according to the experiments carried out by [24].

Chapter 6

Conclusions and Future Work

This work is motivated by the challenges of efficiently transmitting data in large wireless sensor network scenarios, where source nodes (sensors) gather information and need to transmit it through intermediate nodes (sensors/transmitters) to a number of destinations (sink nodes). Due to restrictions, e.g. on bandwidth, power consumption of the sensors, throughput, etc., it is important to consider scenarios where source and network coding are employed. Optimal solutions call for joint source-network coding which come at the cost of high complexity decoders.

Key aspects to be mentioned in this context are: (i) due to the large size of real world sensor networks, low-complexity solutions need to be developed; (ii) joint source-network coding needs to be used to achieve optimal data transmission exploiting the correlation between the sources [6]; (iii) source-network coding calls for: (a) appropriate encoding concepts that exploit correlation between source data (e.g. source clustering, quantization, and mapping); (b) appropriate network coding techniques; (c) strategies for combining source and network coding; (d) low-complexity approaches to make decoder implementation feasible; (iv) the network topology and dependencies in sensor data need to be adequately represented, in order to make the decoder implementation feasible.

In this context, we established as our main goal to design and implement a decoding model inspired in the approach proposed by Maierbacher et al. [8]. The methodology we adopted uses a graphical model based on factor graphs, where we run the sum-product algorithm [18], to represent complex scenarios, allowing for the complexity to grow linearly with N , as we confirmed in our complexity analysis. This makes it possible to construct feasible decoders for large-scale networks. That is an advance compared to the standard implementation of the optimal MMSE decoder, which is unfeasible for large-scale sensor networks due to its complexity growing exponentially with the size of the network [21].

The decoding model we propose has the following distinguishing characteristics: (i) we combine a source-optimized clustering method with a subgraph construction method for two correlated sources, extending the last to an arbitrary number of correlated sources, in order to obtain adequate

flow rates for multicasting information in the network; (ii) we construct a feasible decoder factor graph combining our source and network models, allowing for a joint decoding of source and network coding operations. The source model uses a source-optimized clustering method to represent the correlation between the source nodes. The network model is constructed based on the results of the subgraph construction method, representing the coding operation performed in the network nodes; (iii) we propose three different coding strategies, including a systematic coding scheme for joint source-network decoding.

We have implemented a working prototype using Matlab. Experiments that have been carried out, led to the following conclusions: (i) the use of our subgraph construction method extension to a higher number of correlated sources is viable. We are able to calculate adequate flow rates that meet the Slepian-Wolf conditions and create simpler subgraphs of our original network specific for each sink; (ii) the proposed systematic coding scheme for joint source-network decoding performs better than a random mapping function.

Further work is needed to better understand the behavior of our proposed model. Concerning future developments of this research, we consider the following problems: (i) devise a subgraph construction scheme for more than two correlated sources at a time, so that we can increase the maximum cluster size of our proposed method and consequently take more correlation into account; (ii) test our proposed systematic coding scheme for joint source-network decoding in different scenarios to understand the condition under which it performs better than other solutions and eventually modify it to increase its performance.

Appendix A

Prototype Implementation of Subgraph Construction Method

In Chapter 3, we have described the linear optimization problem which we will use to construct the directed subgraph, obtaining the optimal flow rates for each edge of our network. In this appendix we will explain how we implemented that subgraph construction method.

To implement our scenario in this prototype, we had to find a way to solve the linear optimization problem stated before. We decided to use the Matlab function *linprog*:

$$x = \text{linprog}(f, A, b, A_{eq}, b_{eq}, lb, ub, x_0, options)$$

This function finds the minimum of a problem specified by

$$\min_{\mathbf{x}} \mathbf{f}^T \mathbf{x} \text{ such that } \begin{cases} \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \\ \mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}, \\ \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub}. \end{cases}$$

where \mathbf{f} , \mathbf{x} , \mathbf{b} , \mathbf{b}_{eq} , \mathbf{lb} , and \mathbf{ub} are vectors, and \mathbf{A} and \mathbf{A}_{eq} are matrices.

We define n_{edges} as the total number of edges, i.e. including both direction of a link that connects two nodes and including the virtual edges added. We also define n_{sinks} as the number of sinks and n_{nodes} as the total number of nodes, including all the virtual nodes added, i.e. the cluster nodes and the virtual source S^* . We describe next how we defined our variables.

- The unknown vector \mathbf{x} is our problem solution. We set its length to $n_{edges} \times (n_{sinks} + 1)$ because it represents the edges' flow rates and we need $n_{sinks} + 1$ flow rates for each edge: one for the global flow rates (z_{ij}) and n_{sinks} for the flow rates considering each sink independently (the various x_{ij}). Our variable \mathbf{x} is the concatenation of the subgraph construction algorithm variables z_{ij} and $x_{ij}^{(t)}, t \in T$.
- We set \mathbf{f} , the linear objective function vector, to the edges weights for the first n_{edges} values, which are the values that correspond to z_{ij} , and all the other values to zero as they are not to be considered in the minimization function.

- We set \mathbf{lb} , the vector of lower bounds, to the zero vector, because all the flow rates must be non-negative.
- We set \mathbf{ub} , the vector of upper bounds, to the capacities of the edges because the flow rates cannot exceed the capacities of the corresponding edges.
- \mathbf{A} and \mathbf{b} are our matrix and vector for linear inequality constraints, respectively. We set the length of \mathbf{b} to $n_{edges} \times n_{sinks}$. We set \mathbf{A} 's dimensions to $(n_{edges} \times n_{sinks}, n_{edges} \times (n_{sinks} + 1))$ which are the lengths of \mathbf{b} and \mathbf{x} respectively, because we want to represent the inequality $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$. We will use this inequality to implement the restriction about z_{ij} being the maximum value of all the x_{ij} . The inequality $z_{ij} \geq x_{ij}^{(t)}$ can be rewritten as $-z_{ij} + x_{ij}^{(t)} \leq 0$ so we implement it by setting \mathbf{b} to the zero vector and representing in \mathbf{A} , for each sink, the inequality $-z_{ij} + x_{ij}$ as shown in Fig. A.1.
- \mathbf{A}_{eq} and \mathbf{b}_{eq} are our matrix and vector for linear equality constraints, respectively. We set the length of \mathbf{b}_{eq} to $n_{nodes} \times n_{sinks}$. We set \mathbf{A}_{eq} 's dimensions to $(n_{nodes} \times n_{sinks}, n_{edges} \times (n_{sinks} + 1))$ which are the lengths of \mathbf{b}_{eq} and \mathbf{x} respectively, because we want to represent the equality $\mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}$. We will use this equality to implement the restriction about the difference in the amount of data that reaches and leaves a node. Each value of \mathbf{b}_{eq} represents a node, as well as the lines in \mathbf{A}_{eq} . The columns in \mathbf{A}_{eq} represent the edges, so if an edge leaves a node we set the corresponding value to -1, if an edge arrives at a node we set the corresponding value to 1, and set all the other values to 0. In the vector \mathbf{b}_{eq} we set the node that corresponds to the virtual source S^* to the overall rate $-R$, because that is the difference between the amount of data that arrives at that node and that leaves that node. We set the node that corresponds to the current sink to R and all the other nodes to 0, as the amount of data that reaches these nodes is the same that leaves. There is an illustration of this description in Fig. A.2.

The result \mathbf{x} gives us the optimal flow rates x_{ij} for each directional edge (i, j) , considering one sink at a time, and also its global flow rate z_{ij} , which is just the maximum of all of the corresponding previous mentioned ones.

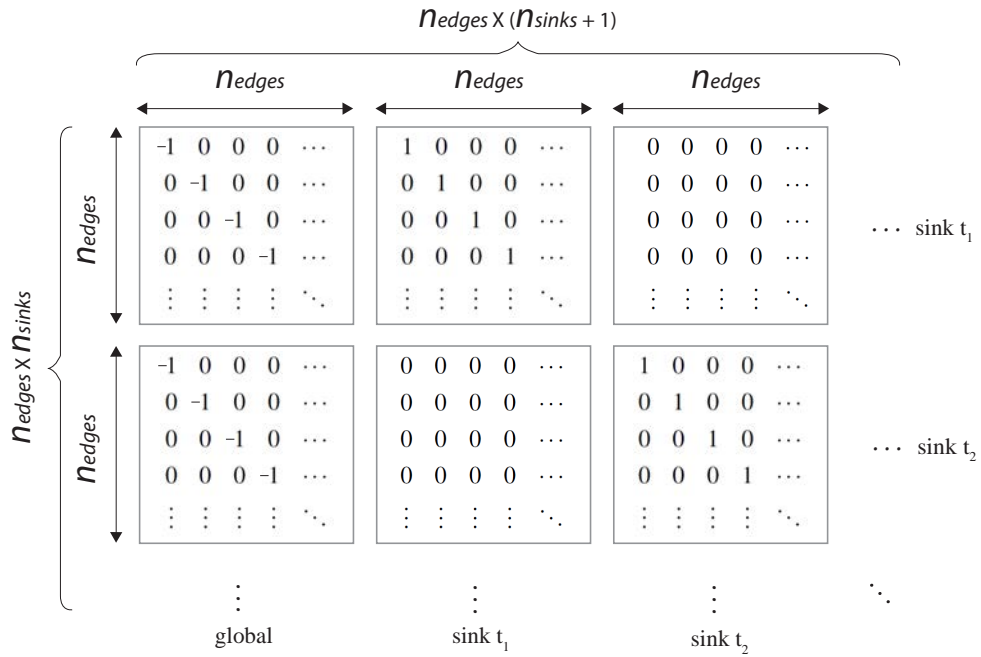


Figure A.1: Content of matrix A representing the inequality constraints

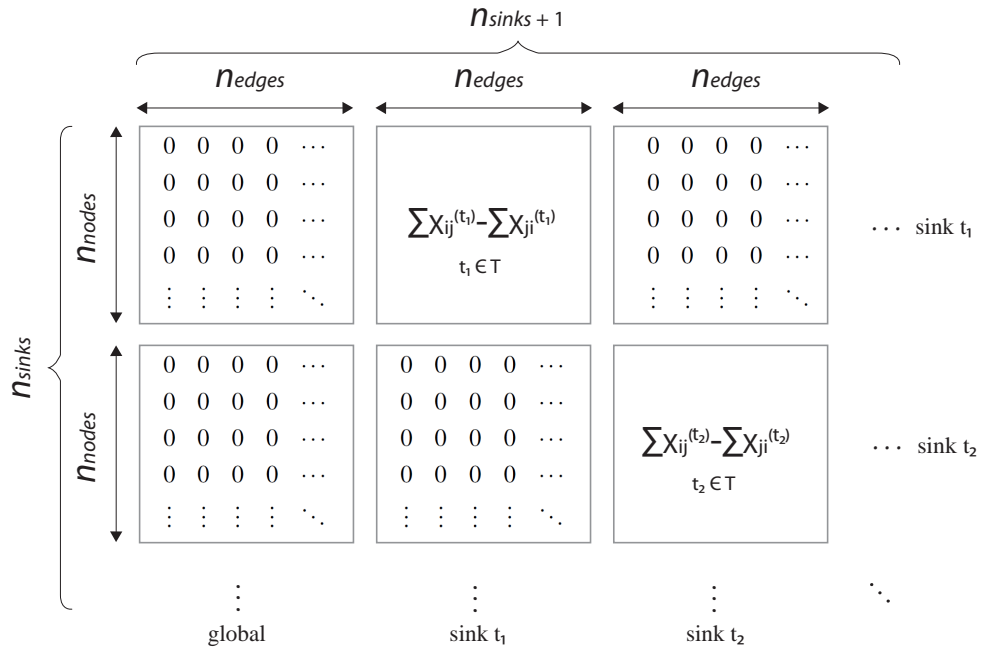


Figure A.2: Content of matrix A_{eq} representing the equality constraints

References

- [1] D. Slepian and J. K. Wolf. Noiseless coding of correlated information sources. *IEEE Trans. Inform. Theory*, IT-19(4):471–480, July 1973.
- [2] C. Chong and S. P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proc. IEEE*, 91(8):1247–1256, August 2003.
- [3] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.
- [4] 10 emerging technologies that will change the world. *Technology Review*, 106(1):33–49, February 2003.
- [5] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, July 2000.
- [6] A. Ramamoorthy, K. Jain, P.A. Chou, and M. Effros. Separating distributed source coding from network coding. *IEEE Trans. Inform. Theory*, 52(6):2785–2795, June 2006.
- [7] T. P. Coleman, M. Médard, and M. Effros. Towards practical minimum-entropy universal decoding. *Proceedings of the Data Compression Conference (DCC 2005)*, pages 33–42, March 2005.
- [8] G. Maierbacher, J. Barros, and M. Médard. Practical source-network decoding. In *IEEE International Symposium on Wireless Communication Systems (ISWCS'09)*, Siena, Italy, September 2009.
- [9] T. Cover. A proof of the data compression theorem of Slepian and Wolf for ergodic sources. *IEEE Trans. Inform. Theory*, March 1975.
- [10] T. J. Flynn and R. M. Gray. Encoding of correlated observations. *IEEE Trans. Inform. Theory*, IT-33(6):773–787, November 1987.
- [11] Z. Xiong, A. D. Liveris, and S. Cheng. Distributed source coding for sensor networks. *Signal Processing Magazine*, 21(5):80–94, September 2004.
- [12] Ralf Koetter and Muriel Médard. Breaking network logjams. *Scientific American Magazine*, June 2007.
- [13] P. Elias, A. Feinstein, and C. E. Shannon. Note on maximum flow through a network. *IRE Transactions on Information Theory*, 27:117–119, 1956.
- [14] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

- [15] Professor Muriel Médard Publications. Available at <http://www.mit.edu/~medard/pubs.html>, last visited in 8 February 2010.
- [16] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Netw.*, 11(5):782–795, October 2003.
- [17] C. Fragouli, J.-Y. LeBoudec, and J. Widmer. Network coding: An instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68, January 2006.
- [18] F. R. Kschischang, B. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *Trans. Inf. Theory*, 47(2):498–519, February 2001.
- [19] T. Ho, M. Médard, M. Effros, and R. Koetter. Network coding for correlated sources. In *Proc. Conf. Information Science and Systems*, Princeton, March 2004.
- [20] M. Effros, M. Médard, T. Ho, S. Ray, D. Karger, and R. Koetter. Linear network codes: A unified framework for source, channel and network coding. In *Proc. DIMACS Workshop Network Information Theory*, Piscataway, NJ, 2003.
- [21] J. Barros and M. Tüchler. Scalable decoding on factor trees: a practical solution for wireless sensor networks. *IEEE Transactions on Communications*, 54(2):284–294, February 2006.
- [22] G. Maierbacher and J. Barros. Diophantine index assignments for distributed source coding. In *Proceedings of the 2007 IEEE Information Theory Workshop (ITW 2007) - Frontiers in Coding*, Lake Tahoe, California, USA, September 2007.
- [23] G. Maierbacher and J. Barros. Low-complexity coding and source-optimized clustering for large-scale sensor networks. *ACM Trans. Sen. Netw.*, 5(3), May 2009.
- [24] A. Lee, M. Médard, K. Z. Haigh, S. Gowan, and P. Rubel. Minimum-cost subgraphs for joint distributed source and network coding. In *Proc. of Workshop on Network Coding, Theory and Applications*, January 2007.
- [25] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.
- [26] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.
- [27] L. Song and R. W. Yeung. Network information flow - multiple sources. In *Proc. IEEE International Symposium on Information Theory*, Washington, DC, June 2001.
- [28] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- [29] A. V. Aho and J. D. Ullman. *Foundations of Computer Science, C Edition*. W. H. Freeman, 1994.