UNIVERSITÀ DEGLI STUDI DI BRESCIA

DEPARTMENT OF INFORMATION ENGINEERING

Master of Science in
Communication Technologies and Multimedia

MASTER THESIS

Constraints in Source Coding

Supervisor:
Prof. Marco Dalai

Candidate:
Stefano Della Fiore

Candidate number:
704779

Academic Year 2017/2018

UNIVERSITÀ DEGLI STUDI DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in *Communication Technologies and Multimedia*

TESI MAGISTRALE

Vincoli nella Codifica di Sorgente

Relatore:
Prof. Marco Dalai

Candidato:
Stefano Della Fiore

Matricola:
704779

Anno Accademico 2017/2018

Abstract

In this thesis, we review and investigate on three different topics that are related to Information theory, discrete mathematics and a little bit of graph theory. First topic is about unique decodability of a code, how the definitions known in literature adapt when we are dealing with constrained sources; referring to Dalai's [3] work, we see that, redefining the definitions given by Cover and Gallager about unique decodability, the Kraft inequality which was always verified for the previous definitions of uniquely decodable codes (McMillan theorem) is no longer verified for constrained sources. This implies that the expected length of a code can be less than the joint entropy of the symbols for which the code is constructed (proof given by Dalai using a first-order Markovian source). We give a proof of the equivalence (in terms of eigenvalues) between Mealy and Moore representation of the same Markovian source which can be useful to represent a source with fewer number of states. The transformation process to derive the Mealy form of a Markov chain given the Moore representation is implemented in Matlab and some useful examples are given. We show a Sardinas-Patterson Matlab code for testing unique decodability of codes and trying to adapt the codewords accordingly to the feedback that the test provides (like colliding codewords). Second topic is about fix-free codes, how they are defined and how can be constructed. We review the work of Ahlswede [7] and Yekhanin [8] whom theorems are the basis to understand correctly this topic. A modified Kraft inequality for this types of codes and its upper-bound conjectured by Ahlswede are seen. We review the complete proof of Yekhanin theorem for the construction of a fix-free codes under the 5/8-constraint related to the Kraft sum and an implementation of the Yekhanin's construction process of fix-free codes is given and some tests are carried out. The last topic is about Graph entropy, how it is defined and how it is related to the classical entropy function defined by Shannon. Graph entropy was introduced by Korner [11] in 1973 for a source coding problem about distinguishability of symbols. We review the work of Simonyi[12] about the proof of Korner theorem in terms of rate distortion theory and graph theory.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dalai for the continuous support of my thesis work, for his patience, motivation, and immense knowledge.

I would like to thank my parents for supporting me spiritually throughout writing this thesis and my life in general. Thank you both for giving me strength to reach my dreams.

A special thank to Veronica, my partner, for the patience and support that she always gave to me. For encouraging me before every exam and for having celebrated every success with me.

Last but not least a thank goes to all my family that has always supported me and encouraged me in difficult moments.

Contents

Introduction

Source coding is a technique used in communication systems to describe information sources with probabilistic descriptions. The simplest class of source models are the discrete memoryless sources in which each symbol produced by the source does not depend on previous symbols. Each symbol is taken from a source alphabet with a certain probability. The goal of Source coding is to assign to each sequence of symbols a codeword that will be sent to the Channel encoder which adds some redundancy in order to make possible the reconstruction of the sequence of codewords at the receiver also if there are some errors due to noisy channels.

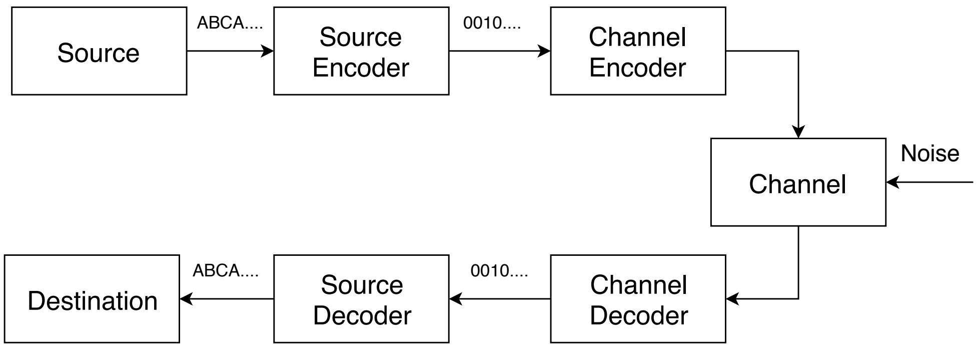

Figure 1: *Communication scheme. Separate Source and Channel encoders.*

The first topic in our work is about constrained sources (sources with memory) where some sequences of symbols cannot be constructed due to the probabilistic description of the source under consideration. We have dealt with first-order Markovian sources in order to bring out some properties that are not easily derivable from the definitions that can be found in literature. Unique decodability is a fundamental property that a code must have in order to allow the decoder to correctly decode a message. We will see different definitions about unique decodability in a classic sense (Cover, McMillan) and in an extended sense (Dalai [3]). It is well known that the uniquely decodable codes par excellence are the prefix-free codes. In this thesis we study the work of Ahlswede and Yekhanin

about fix-free codes which are codes that can be instantaneously decoded in both directions and can improve the channel noise resistance and make faster the decoding process.
In a bigger picture, if we consider a memoryless and stationary information source where the emitted symbols are not all distinguishable these types of sources may be defined as constrained sources which are described by graphs where each edge determines if two symbols are distinguishable. Korner provides an information measure to define the minimum number of codewords for representing these types of information sources.

The structure of the thesis is as follows:

In Chapter 1 the main topic is the unique decodability of constrained sources, the review of all the definitions and theorems about unique decodability allows us to focus on the Mealy and Moore representation of first-order Markovian sources proving the equivalence between two forms. Dalai [3] showed that codes created for constrained sources do not respect anymore the Kraft inequality; we will see in details the proof and how this affects the lower bound on the expected length of the code. Sardinas-Patterson procedure is used to determine if a code is uniquely decodable, it is adapted and implemented in Matlab to carry out some tests on different Markovian sources.

In Chapter 2 the main topic is Yekhanin proof of the improved Kraft-like bound for the existence of fix-free codes with given codewords lengths. We review the fundamental theorems about fix-free codes known in the literature and we give an implementation of the construction process defined by Yekhanin in order to produce some tests with different sets of codeword lengths.

In Chapter 3 Korner entropy is defined and different definitions are stated (Simonyi [12]). The proof of Korner's theorem in terms of rate distortion theory is reviewed and some direct properties of the graph entropy are described. The work of Simonyi has highlighted correlation between graph theory and information theory. This leads to different formulations of graph entropy and also to different proofs of its properties.

Chapter 1

Constrained Sequences

1.1 Basics

We start with some useful definitions (given by Cover [1]) needed in the following sections. Basic knowledge of probability theory is taken for granted.

Definition 1. *The entropy of a discrete random variable X with an alphabet $\mathcal{X}$ and probability mass function $p(x) = P(X = x)$ is defined as:*

$$\begin{aligned} H(X) &= -\sum_{x\in\mathcal{X}} p(x) \cdot \log_2 p(x). \qquad (1.1) \\ &= E[-\log_2 p(x)]. \end{aligned}$$

where $E[\cdot]$ is the Expected value.

$H(X)$ gives a measure on the uncertainty of the random variable X; if X is deterministic $H(X) = 0$ instead if we have a uniform probability mass function so that each symbol has the same probability of each other $H(X) = \log_2(|\mathcal{X}|)$. $H(X)$ is a strictly concave function with a global maximum when $p(x) = \mathbf{u}$ (uniform distribution).

Definition 2. *A stochastic process $\{X_i\}$ is an indexed sequence of random variables. The process is characterized by the joint probability mass function:*

$$P\{(X_1, X_2, ..., X_n) = (x_1, x_2, ..., x_n)\} = p(x_1, x_2, ..., x_n) \qquad (1.2)$$

$$\textit{with } (x_1, x_2, ..., x_n) \in \mathcal{X}^n$$

for all $n = 1, 2, ...$

Definition 3. *A discrete stochastic process $X_1, X_2, ...$ is said to be a Markov chain or a Markov process if for $n = 1, 2, ...$:*

$$P(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) = \tag{1.3}$$

$$P(X_{n+1} = x_{n+1} | X_n = x_n)$$

for all $n = 1, 2, ...$
The joint probability mass function can be written as:

$$p(x_1, x_2, ..., x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2) \cdots p(x_n|x_{n-1}). \tag{1.4}$$

The expression 1.4 can be easily obtained by expressing the joint probability mass function with the chain-rule and then applying the Markov property on each term.

Definition 4. *The Markov chain is said to be **homogeneous** if the conditional probability $p(x_{n+1}|x_n)$ does not depend on n for $n = 1, 2, ...$:*

$$Pr\{X_{n+1} = b | X_n = a\} = Pr\{X_2 = b | X_1 = a\} \ \forall a, b \in \mathcal{X}. \tag{1.5}$$

Definition 5. *A markov chain is said to be **irreducible** if it is possible to go with positive probability from any state of the Markov chain to any other state in a finite number of steps.*

This means that if we have a transition probability matrix $\mathbf{P}$ then there exists k_{min} (minimum number of steps) such that $(\mathbf{P}^k)_{ij} > 0$ for all the indices i, j, for $k \geq k_{min}$.

Definition 6. *Given a Markov chain a **cycle** is a sequence of state transitions that starts from a specific state and ends in the same state.*

Definition 7. *A Markov chain is said **aperiodic** when the GCD(Greatest common divisor) of the lengths of all cycles is 1.*

Definition 8. *π is called a stationary distribution for a homogeneous Markov chain if:*

$$\pi(x_n) = \pi(x_{n+1}) = \sum_{x_n} \pi(x_n) P_{x_n x_{n+1}}, \tag{1.6}$$

where P_{ij} is the transition probability from state i to j.
If the finite-state homogeneous Markov chain is irreducible and aperiodic, the stationary distribution is unique, and from any starting distribution, the distribution of X_n tends to the stationary distribution π as $n \to \infty$.

If the probability distribution of the first symbol $P(X_1)$ is equal to the stationary distribution π the Markov source is **stationary**.

Definition 9. *The entropy (entropy rate) of a stochastic process $\{X_i\}$ is defined by:*

$$H(\mathcal{X}) = lim_{n\to\infty}\frac{1}{n}H(X_1, X_2, ..., X_n) \tag{1.7}$$

if the limit exists. For stationary processes the limit always exists.

Invoking the weak law of large numbers (see [1]), it can be proved by means of the Asymptotic Equipartition property (AEP) that:

$$-\frac{1}{n}\log\ P(X_1, X_2, ..., X_n) \to H(\mathcal{X}) \quad \text{in probability.} \tag{1.8}$$

Theorem 1. *For a stationary first-order Markov chain, the entropy rate is given by:*

$$H(\mathcal{X}) = H(X_2|X_1) = \sum_i \pi_i \sum_j P_{ij}\log\frac{1}{P_{ij}} \tag{1.9}$$

where π is the stationary distribution and P is the transition probability matrix of the Markov chain associated.

1.2 Unique decodability

In this chapter we consider the problem of encoding a source with codes that are not *decodable* in the classic sense, but are decodable for some sources with memory with different constraints on the possible generated sequences. As previously defined we consider first order Markov sources, the output alphabet is binary (bits) and codes are fixed-to-variable which means mapping a fixed number of source symbols (for simplicity we always consider one symbol) into a variable number of output symbols as shown in Figure 1.1.

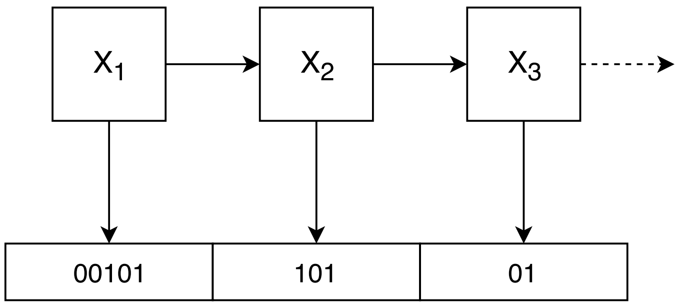

Figure 1.1: *First-order Markovian source encoding.*

From now on, we always consider lossless encoding of discrete sources. The model defined in Figure 1.1 leads to a problem in the definition of

"*uniquely decodable*" and expected length of codes. Shannon showed that, asymptotically, it is not possible to encode a source at rates below the entropy. Nothing was said about the minimum rate required for the representation of a finite number of symbols.
McMillan [6] showed that a *uniquely decodable* code must satisfy the Kraft inequality, and so the minimum expected length of a lossless code is always greater or equal to the entropy of the source (see Theorem 2.1 and the following section). We will see an example [3] of a code associated to a Markovian source in which this assumption is not satisfied due to a non-complete definition of uniquely decodable codes.

Let us see some definitions and theorems on uniquely decodable codes that can be found in literature [1] and then alternative definitions based on Dalai's [3] work.

Definition 10. *An information source $\boldsymbol{X}$ is an infinite sequence of random variables $X_1, X_2, X_3, \ldots$ taking values in a finite alphabet $\mathcal{X}$. The source $\boldsymbol{X}$ is said to be memoryless if $X_1, X_2, \ldots$ are independent and identically distributed (i.i.d.). So we say that the source has memory if the random variables $X_1, \ldots, X_n$ are not independent.*

Definition 11. *A variable-length code for a random variable X is a map from the alphabet $\mathcal{X}$ to $\mathcal{D}^*$, where $\mathcal{D}^*$ is the set of finite length sequences of symbols from a D-ary alphabet. For each possible symbol $x \in \mathcal{X}$ the codeword associated to x is $C(x)$ and with $l(x)$ we identify its length.*

Definition 12. *A code is said to be non-singular if:*

$$\forall x_i, x_j \in \mathcal{X},\ \ x_i \neq x_j \textit{ implies } C(x_i) \neq C(x_j). \tag{1.10}$$

Definition 13. *The extension C^* of a code C is the mapping from finite length sequences of $\mathcal{X}$ to finite length sequences of $\mathcal{D}$ defined by:*

$$C(x_1 x_2 \ldots x_n) = C(x_1)C(x_2)\ldots C(x_n) \tag{1.11}$$

where the strings represent the concatenation of different source symbols and coding symbols.

Definition 14. *In the classic sense a code C is said uniquely decodable if its extension is non-singular.*

Definition (14) makes the assumption that all the combinations of symbols can be produced by the source with positive probability. It means that if we represent the source with a first-order Markov chain the transition probability matrix P has all its entries positive meaning

that all the transitions between different states are always possible. When we are in this configuration (when all combinations of symbols are possible) then we call these sources: ***unconstrained sources***; at the other end if some combinations are impossible to obtain we call these sources: ***constrained sources***.
Gallager [2] gives his own definition for *uniquely decodable* codes that is not based on the extension of the code but he talks about "*sequence of code letters*" so it is a different definition with respect to the one given by Cover (14), but implicit in the definition memoryless sources are considered.

Definition 15 (Dalai [5]). *Let $\boldsymbol{X}$ be a discrete information source on the alphabet $\mathcal{X}$. We say that $\boldsymbol{X}$ is a constrained source if for at least one finite n there exists an element of $\mathcal{X}^n$ that cannot be obtained as outcome of the first n symbols of the source. Otherwise we say that the source is unconstrained.*

Definition 16 (Dalai [5]). *Generalization of uniquely decodable code that works also with constrained sources. Let $\boldsymbol{X}$ be an information source with alphabet $\mathcal{X}$. A code C is said to be uniquely decodable for the source $\boldsymbol{X}$ if no two different finite sequences of source symbols producible by $\boldsymbol{X}$ have the same codeword.*

With these definitions (15, 16) we have a complete coverage of all the possible sequences of symbols outcome from a source. All the uniquely decodable sources in the "*classic sense*" and also codes for constrained sources are incorporated in the new definitions.

Definition 17. *A code is called a prefix-code (prefix-free) if no codeword is a prefix of any other codeword.*

Under prefix-free conditions the code is said to be *instantaneous*: this means that the decoder can decode the received messages in linear time ($O(N)$ where N is the length of the received sequence and $O(\cdot)$ is the computational complexity). Given a sequence of code symbols the decoder can identify a unique sequence of the received symbols in order to reconstruct the message correctly.

Theorem 2 (Kraft Inequality). *Let $l_i, i = 1, ..., n$, be the lengths of the codewords of a prefix code and let $D = |\mathcal{D}|$ be the size of the code alphabet $\mathcal{D}$. It states that*

$$\sum_{i=1}^{n} D^{-l_i} \leq 1. \tag{1.12}$$

Conversely, if a set of integers $l_i, i = 1, ..., n$ satisfy the Kraft inequality then a prefix-free code can be constructed with those codeword lengths.

Corollary 2.1. *It follows from the Kraft inequality that if a prefix code is used for encoding a random variable X, then the expected length of the codeword generated is always greater or equal to the entropy of X.*

$$E[l(X)] \geq H_D(X). \tag{1.13}$$

Proof. If p_i are the probabilities of the symbols in $\mathcal{X}$ and l_i the lengths of the related codewords, we have that

$$\begin{aligned} H(D) - E[l(X)] &= \sum_{i=1}^{|\mathcal{X}|} -p_i \log_D p_i - \sum_{i=1}^{|\mathcal{X}|} p_i \cdot l_i \qquad (1.14) \\ &= \sum_{i=1}^{|\mathcal{X}|} p_i \log_D \frac{D^{-l_i}}{p_i} \\ &\leq \sum_{i=1}^{|\mathcal{X}|} p_i (\frac{D^{-l_I}}{p_i} - 1) \log_D e \\ &\leq 0. \end{aligned}$$

The upper bound of logarithm $\log_D(x) \leq (x-1)\log_D e$ and the Kraft inequality allow to prove the 1.13 inequality. □

Corollary 2.2. *For every prefix code the expected length of the code for n symbols of a source $\boldsymbol{X}$ satisfies:*

$$E[l(X_1, X_2, ..., X_n)] \geq H(X_1, X_2, ..., X_n) \tag{1.15}$$

This corollary on prefix-code is a strong result with respect to the asymptotic lower bound given by Shannon (infinite number of symbols) because it is applied to finite sequences of symbols.

In the previous theorem we consider prefix-codes or non-constrained uniquely decodable codes. For these codes McMillan [6] proved that both uniquely decodable and prefix-free codes satisfy the Kraft inequality, meaning that there is no advantage in using a uniquely decodable code instead of a prefix-free because both have the same lower bound on the expected length seen in Corollary 2.2. Using a uniquely decodable code only makes the decoder more complex. As we said before there are some sources that do not produce some of the possible sequences of symbols; in these cases the Kraft inequality does not hold anymore [3] and this brought to a new theorem (Theorem 3).

So the Kraft inequality is no longer a necessary condition for the code to be uniquely decodable (in the larger sense).

Theorem 3. *(Dalai [5]) There exists at least one source $\boldsymbol{X} = (X_1, X_2, ..., X_n)$ and a uniquely decodable code for $\boldsymbol{X}$ such that, for every $n \geq 1$:*

$$E[l(X_1, X_2, ..., X_n)] < H(X_1, X_2, ..., X_n) \tag{1.16}$$

Proof. Let us see an example with a first order Markovian source, considering a source $\mathbf{X}$ generating symbols $X_1, X_2, ...$ where each $X_i \in \mathcal{X} = \{A, B, C, D\}$. The sequence of generating symbols is based on the following transition probability graph:

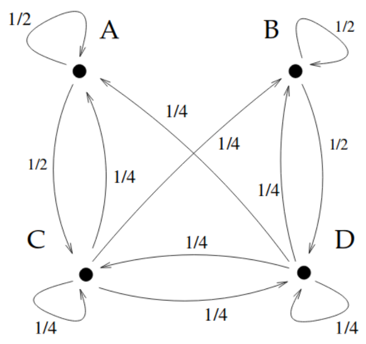

Figure 1.2: *Graph related to Markov source with some impossible transitions (Moore form), from [5].*

The associated transition probability matrix is

$$\mathbf{P} = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix}. \tag{1.17}$$

It can be easily shown that the unique stationary distribution is the uniform distribution over the set of source symbols, that is $\boldsymbol{\pi} = (1/4, 1/4, 1/4, 1/4)$.
If we set the distribution $P(X_1) = \boldsymbol{\pi}$ then the source is stationary. $\mathbf{P}$ is also irreducible and aperiodic and so the source is ergodic.

First we consider a classic encoding algorithm (Huffman) that generates the optimal code in the classic sense for a sequence of symbols. In this particular case all the entries of $\mathbf{P}$ are powers of the coding dictionary cardinality $|\mathcal{D}| = 2$ and this implies that the $H(X_1^n)$ can be reached with a prefix-free code. The Huffman code is constructed by encoding the first symbol independently and all the successive symbols are encoded using the transition probability matrix $\mathbf{P}$ creating different Huffman codes for all the $\mathbf{P}$-rows.

Example 1. *An Huffman code for the stationary Markov chain based on the transition probability matrix* $\boldsymbol{P}$ *(eq. 1.17) can be constructed as* *follows*

$$\begin{aligned} P(X_1) &= (1/4, 1/4, 1/4, 1/4), \\ \mathcal{C}_1 &= (A \to 00, B \to 01, C \to 10, D \to 11), \end{aligned}$$

where $\mathcal{C}_1$ *is the Huffman code for the first symbol. Then for all the possible state transitions an Huffman code is constructed:*

$$\begin{aligned} P(X_2|X_1 = A) &= (1/2, 0, 1/2, 0), \\ \mathcal{C}_A &= (A \to 0, -, C \to 1, -), \end{aligned}$$

where $\mathcal{C}_A$ *is the Huffman code related to transitions which start from symbol* A*. With the same procedure we carry out*

$$\begin{aligned} \mathcal{C}_B &= (-, B \to 0, -, D \to 1), \\ \mathcal{C}_C &= (A \to 00, B \to 01, C \to 10, D \to 11), \\ \mathcal{C}_D &= (A \to 00, B \to 01, C \to 10, D \to 11), \end{aligned}$$

which are the codes for the remaining state transitions.

The expected length of the Huffman code for the first n symbols reaches the entropy of the sequence which can be expressed as

$$H(X_1^n) = H(X_1) + \sum_{i=2}^{n} H(X_i|X_1^{i-1}) = H(X_1) + (n-1)H(X_2|X_1), \quad (1.18)$$

$$H(X_1^n) = 2 + \frac{3}{2}(n-1), \quad (1.19)$$

for all $n \geq 1$. This result is achieved using the chain rule of the entropy and the fact that the source is stationary.
As we said before since the entries of the $\mathbf{P}$ matrix are power of 2 and $P(X_1) = (1/4, 1/4, 1/4, 1/4)$, then the expected length of the codeword equals the entropy of the sequence:

$$E[l(X_1^n)] = H(X_1^n) = 2 + \frac{3}{2}(n-1). \quad (1.20)$$

In [3], a different code is proposed: a fixed map from $\mathcal{X} \to \mathcal{D}^*$ such that $A \to 0$, $B \to 1$, $C \to 01$, $D \to 10$. This code is not uniquely decodable in the classic sense because the concatenation BA produces 10 that is also produced by symbol D. The advantage of this coding scheme is that the impossible transitions are exploited. In fact, the transition $A \to B$ and vice versa are not allowed by the Markovian source, so this encoding scheme does not produce ambiguity and the source sequence can

be correctly reconstructed at the decoder (making it uniquely decodable code in this sense).
Evaluating the expected length of the code, we see that:

$$E[l(X_1^n)] = \sum_{i=1}^{n} E[l(X_i)] = \frac{3}{2}n. \tag{1.21}$$

□

We have now proved that the expected length of this code is strictly smaller than the entropy of the symbols.
The difference between the two expected code lengths can be calculated as

$$r = E[l(X_1^n)] - H(X_1^n) = -\frac{1}{2}, \tag{1.22}$$

where r is called the redundancy of the code and is usually a non-negative quantity.
Thinking about the asymptotic equipartition (AEP) for ergodic sources (McMillan) shown in Gallager's book [2], intuitively, the minimum expected length per symbol for both constrained and unconstrained Makovian ergodic sources is the entropy rate of the source.

$$E[l(X_1^n)] \geq nH(\mathcal{X})\, with\, n \geq 1. \tag{1.23}$$

With the fix-mapped code the gain obtained with respect to the Huffman code is only at the first symbol because $H(X_2|X_1) = 3/2$ and so, there is no gain when we have state transitions. The Huffman code is more expensive in terms of computational complexity because the decoder must have stored all the codes for each possible transition, while for the custom-code the decoder needs to know the impossible transitions but the matching between encoding bits and symbols is faster.

1.3 Karush's proof for constrained uniquely decodable codes

As we said in the previous section, McMillan proved that the uniquely decodable codes (classic sense) satisfy the Kraft inequality. When constrained sources are used this necessary condition is no more correct.
Karush's proof of McMillan theorem is useful to determine a new necessary condition for constrained sources. Considering the following quantity for $k > 0$

$$\left(\sum_{i=1}^{n} D^{-l_i}\right)^k \tag{1.24}$$

where n is the number of codewords and the number of terms produced is n^k (e.g. one term is $D^{-l_1}D^{-l_2}\cdots D^{-l_n}$). This expression produces all the possible product combinations of D^{-l_i}. Each term can be associated to a unique sequence of symbols by means of the length indices in the product term. Given r the total codeword length of k symbols, the associated term is D^{-r}. If the code must be uniquely decodable there are at most D^r sequence with a code of length r meaning that each sequence of k symbols contributes in the sum with a quantity that is at most 1.

$$\left(\sum_{i=1}^{n} D^{-l_i}\right)^k = \sum_{r=kl_{min}}^{kl_{max}} N(r)D^{-r} \leq k(l_{max} - l_{min} + 1) \tag{1.25}$$

where $l_{min} = \min_i(l_i)$, $l_{max} = \max_i(l_i)$ and $N(r)$ is the number of sequences of length r. This inequality must hold for every $k > 0$. But at the left-hand side of the inequality we have an exponential function in k while at the right-hand side we have a linear function in k. This implies that

$$\sum_{i=1}^{n} D^{-l_i} \leq 1. \tag{1.26}$$

In this proof we have implicitly supposed that $N(r) > 0$, $\forall r = kl_{min}, ...,$ kl_{max}: meaning that the code is uniquely decodable in classic sense. Hence we need an expression which incorporates the possibility that one sequence of symbols is forbidden. Consider the following matrix:

$$\mathbf{Q}(D) = \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} & 0 \\ 0 & D^{-l_2} & 0 & D^{-l_4} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} & D^{-l_4} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} & D^{-l_4} \end{pmatrix}, \tag{1.27}$$

and also the following row vector:

$$\mathbf{L} = (D^{-l_1}, D^{-l_2}, D^{-l_3}, D^{-l_4}), \tag{1.28}$$

where D $= |\mathcal{D}| = 2$ related to the Markov source in Figure 1.2 and l_1, l_2, l_3, l_4 are the lengths of the codewords associated to the symbols A, B, C, D.

The inequality 1.25 for constrained sources can be rewritten using $\mathbf{Q}$ and $\mathbf{L}$:

$$\mathbf{L} \cdot \mathbf{Q}(D)^{k-1} \cdot \mathbf{1}_4^T \leq k(l_{max} - l_{min} + 1)^1, \tag{1.29}$$

where $\mathbf{1}_4 = (1, 1, 1, 1)$. It is shown in [3] that a necessary condition for this inequality to be satisfied for every k is that the spectral radius $\rho(\mathbf{Q}) = \max_i |\lambda_i|$ (λ_i are the eigenvalues of the matrix $\mathbf{Q}(D)$) of $\mathbf{Q}$ must be less than or equal to 1.

$^1\mathbf{A}^T$ means the transpose of the matrix or vector $\mathbf{A}$.

Theorem 4 (Dalai [3])**.** *Let $\boldsymbol{P}$ be an irreducible $N \times N$ transition probability matrix of a Markov chain and $\boldsymbol{l} = (l_1, l_2, ..., l_N)$ a set of integers. Let the matrix $\boldsymbol{Q}$ be defined as*

$$\boldsymbol{Q}_{ij}(D) = \begin{cases} D^{-l_i} & \boldsymbol{P}_{ij} > 0 \\ 0 & \boldsymbol{P}_{ij} = 0 \end{cases}. \tag{1.30}$$

A necessary condition for a code with lengths l_i to be uniquely decodable (extended sense) is that $\rho(\boldsymbol{Q}) \leq 1$.

Proof. Starting from inequality 1.29 and knowing that $\mathbf{Q}$ is also a non-negative matrix by Perron-Frobenius theorem its spectral radius $\rho(\mathbf{Q})$ is also an eigenvalue with algebraic multiplicity 1 and with a positive associated eigenvector. If we call this eigenvector $\mathbf{w}^T$, we can define the vector $\mathbf{L} = \alpha\mathbf{w} + \mathbf{z}$ using the eigenvector $\mathbf{w}$ and a non-negative vector $\mathbf{z}$. Since $\mathbf{L}$ is a non-negative vector there exists such positive constant α.

$$\begin{aligned} \mathbf{L}\mathbf{Q}^{k-1}\mathbf{1}_n^T &= \alpha\mathbf{w}\mathbf{Q}^{k-1}\mathbf{1}_n^T + \mathbf{z}\mathbf{Q}^{k-1}\mathbf{1}_n^T \\ &= \alpha\rho(\mathbf{Q})^{k-1}\mathbf{w}\mathbf{1}_n^T + \mathbf{z}\mathbf{Q}^{k-1}\mathbf{1}_n^T. \end{aligned} \tag{1.31}$$

We can see that the coefficient of $\rho(\mathbf{Q})$ is non-negative and $\mathbf{z}\mathbf{Q}^{k-1}\mathbf{1}_4$ is also non-negative. If $\rho(\mathbf{Q}) > 1$ the left-hand side of inequality 1.29 grows exponentially with k while the right-hand side grows linearly in k meaning that $\rho(\mathbf{Q}) \leq 1$ for the inequality to be satisfied for every k. □

Some considerations can be carried out: if all the entries of the matrix $\mathbf{P}$ are positive then the matrix $\mathbf{Q}$ has all equal rows, in this case the spectral radius $\rho(\mathbf{Q}) = \sum_i D^{-l_i}$ applying the inequality stated before we obtain the ordinary Kraft inequality.
The limit case is when $\rho(\mathbf{Q}) = 1$. Suppose that we have constructed a uniquely decodable code with a set of lengths $\{l_i\}$ for which the spectral radius of the matrix $\mathbf{Q}$ is equal to 1. If we decrease only just one codeword length l_j this will because an increase in the spectral radius and so the code is not anymore a uniquely decodable code.

It is important to notice that this is a necessary condition for the code to be uniquely decodable and not a sufficient condition (contrarily to the classic Kraft inequality). So there exist codeword lengths which satisfy the spectral radius inequality but a uniquely decodable (UD) code with the mentioned lengths can not be constructed. The Sardinas-Patterson test [9] for UD-codes can be generalized to work also for constrained sources.

1.4 Generalized Sardinas-Patterson test

In the previous sections, based on [3], we have seen that the Kraft inequality is not a necessary and sufficient condition for the code to be uniquely decodable in the general sense and thanks to the Karush proof of the McMillan theorem Dalai was able to derive a new necessary condition (not sufficient): $\rho(\mathbf{Q}) \leq 1$.

For a better understanding of the Sardinas-Patterson test we will consider binary codes and transition probability matrices in Moore form where each vertex of the graph represents a source symbol.
In the following examples we encode each source symbol with its associated codeword (fix-map) without encoding state transitions.

Example 2. *Let us consider a ternary Markov source with transition graph shown in fig. 1.3.*

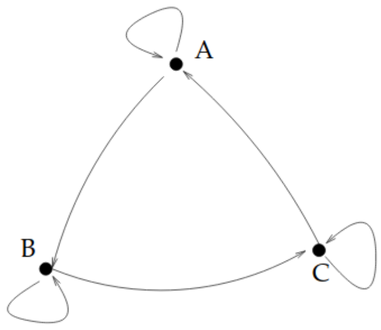

Figure 1.3: *Transition graph where $\rho(\boldsymbol{Q}) \leq 1$ is not a sufficient condition, from [5].*

If we set the vector of lengths $\boldsymbol{l} = (1, 1, 1)$ then $\rho(\boldsymbol{Q}) = 1$ but it is clearly impossible to decode if we assign only one bit to the three possible symbols. In general if we have more than 2^k codewords of length k the code is always a non decodable code. Here the transition probability matrix related to transition graph in Figure 1.3 with lengths $\boldsymbol{l}$ = (1, 1, 1) satisfies

$$\rho(\boldsymbol{Q}) = \rho \begin{pmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{pmatrix} = 1.$$

Example 3. *Considering a source shown in fig. 1.4 with three symbols A, B, C.*

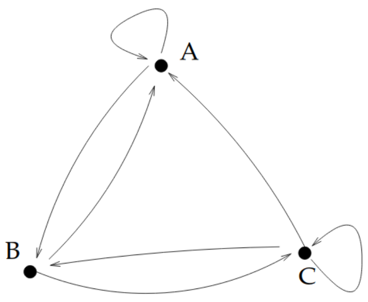

Figure 1.4: *Transition graph where $\rho(\boldsymbol{Q}) < 1$ it is not a sufficient condition, from [5].*

If we set the vector of lengths $\boldsymbol{l} = (1, 1, 2)$, we have two codewords of length 1 and one codeword of length 2, nevertheless a uniquely decodable code does not exist. Let us give an example: if we assign to symbol A codeword 1, for B codeword 0 then the only possible codeword for C is 00, but we can easily see that the sequences of symbols BC and CB produce the same code so it is not an uniquely decodable code. Here the transition probability matrix related to transition graph in Figure 1.4 with lengths $\boldsymbol{l} = (1, 1, 2)$ has spectral radius

$$\rho(\boldsymbol{Q}) = \rho \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/4 \\ 1/2 & 1/2 & 1/4 \end{pmatrix} < 1.$$

We have observed that the condition on the spectral radius $\rho(\mathbf{Q}) \leq 1$ is a necessary condition but not a sufficient condition. In order to check if a code is uniquely decodable (for both constrained and unconstrained sources) the Sardinas-Patterson [9] test can be generalized to consider also constrained sources. Given a set of codewords, the Sardinas-Patterson test allows us to determine in a finite number of steps if the code is uniquely decodable also in the general case. Let us see in details the Sardinas algorithm modified [5] to work with first-order Markovian sources.

Let $\mathcal{X} = \{1, 2, ..., n\}$ be our source alphabet, $\mathcal{C} = \{c_i\}_{i=1,...,n}$ our code under test with associated codewords c_i and $\mathbf{P}$ the transition probability matrix associated to the source, the following steps define the algorithm process:

1. $\forall i = 1, ..., n$ let $F_i = \{c_j \mid P_{ij} > 0, 1 \leq j \leq n\}$ be the subset of $\mathcal{C}$ containing all codewords that can be reached from state i (defined in the transition probability matrix);

2. we have to build a sequence of sets $S_1, S_2, ...$, so that at the first step ($n = 1$) for building S_1 we consider all pairs of codewords drawn form the set $\mathcal{C}$ this means that $(c_i, c_j) \in \binom{\mathcal{C}}{2}$; if the code c_i is a prefix of a code c_j meaning that $c_j = c_iA$ (with A non-empty word) we then put the suffix A into S_1 set.
   In order to consider only the possible transitions we have to save the indices related to the codewords that have generated all the suffixes, so, we have to tag the suffix A with two labels i, j. The representation of the suffix become ${}_iA_j$;

3. for $n > 1$ the set S_n is constructed in an iterative way comparing the elements of S_{n-1} with elements of $\mathcal{C}$. For each element ${}_lH_m \in S_{n-1}$ we consider $F_l \subseteq \mathcal{C}$:

   (a) if a codeword $c_i \in F_l$ is equal to ${}_lH_m$ the iterative process ends and the code is not uniquely decodable;

   (b) if ${}_lH_m$ is a prefix of a codeword $c_r = {}_lH_mB \in F_l$ we label suffix B (${}_mB_r$) and we insert it into S_n;

   (c) if a codeword $c_s \in F_l$ is a prefix of ${}_lH_m = c_sD$ we label suffix D (${}_sD_m$) and place it into S_n.

   (d) if $S_n = \emptyset$ or $S_n = S_r$ for some $r < n$ then the algorithm stops and the code is uniquely decodable.

The algorithm stops after a finite number of cycles (because there are a finite number of different sets S_i). The sequence $S_1, S_2, ...$ can be finite or periodic. A code is said to be uniquely decodable with *finite delay* if the sequence of S_i is finite (from a certain $k \geq 1$, S_k becomes equal to the empty set) while it is said to be uniquely decodable with *infinite delay* if the sequence of the S_i is periodic.
In case of periodicity the code is always decodable after sending the entire sequence of symbols but fixing the length of the sequence to n there are at least two or more sequences of symbols that share the same code and thus the delay for decoding cannot be bounded.

Example 4. *Let us consider an example of the test applied on the following constrained source:*

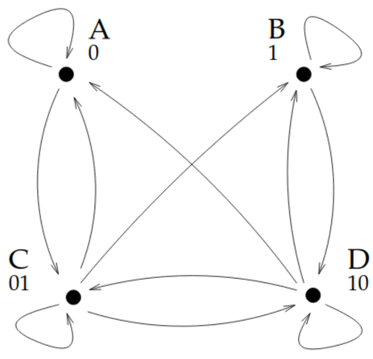

Figure 1.5: *Transition graph where $\rho(\boldsymbol{Q}) \leq 1$ is a sufficient condition, from [5].*

We can see from Figure 1.5 that the code associated is $\mathcal{C} = \{0, 1, 01, 10\}$. $A \to 0$, $B \to 1$, $C \to 01$ and $D \to 10$. Applying the Sardinas-Patterson procedure $S_1 = \{_A1_C,\ _H0_D\}$ and $S_2 = \emptyset$ meaning that the code $\mathcal{C}$ is a uniquely decodable code with finite delay and this delay (in bits) can be calculated knowing the first index i for which the set $S_i = \emptyset$.

Example 5. *Another similar example can be useful to understand the procedure:*

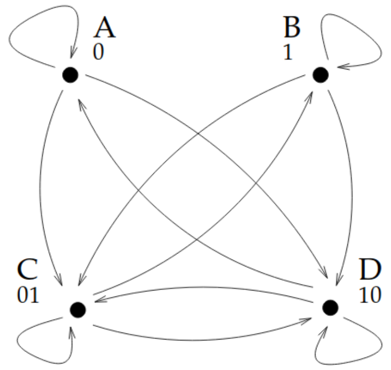

Figure 1.6: *Transition graph where $\rho(\boldsymbol{Q}) \leq 1$ is a sufficient condition, from [5].*

We can see from Figure 1.6 that the code is the same as before: $\mathcal{C} = \{0, 1, 01, 10\}$. The mapped codewords are: $A \to 0$, $B \to 1$, $C \to 01$ and $D \to 10$. The suffix sets are $S_1 = \{_A1_C,\ _B0_D\}$, $S_2 = \{_C0_D,\ _D1_C\}$ and from now on $S_i = S_{i-1}$ with $i > 2$ meaning that the code $\mathcal{C}$ is uniquely decodable with infinite delay (it is not possible to distinguish sequences BCCC... and DDD... until the end of the entire sequence).

Example 6. *Now we see an example of a code that is not easy to determine a-priori if it is a uniquely decodable code. Consider a stationary Markov source defined by its transition probability matrix* $\boldsymbol{P}$. *For the Sardinas-Patterson test the only important information is the adjacency matrix relative to the Markov chain so:*

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

There are five source symbols $\mathcal{X} = \{A, B, C, D, E\}$ *and the associated code that we want to test is* $\mathcal{C} = \{00, 001, 101, 010, 10101\}$.

The first iterations of the test are as follows:

1. $S_1 = \{{}_A1_B,\ {}_C01_E\}$;
2. $S_2 = \{{}_B01_C,\ {}_B0101_E,\ {}_E0_D\}$;
3. $S_3 = \{{}_C0_D,\ {}_D1_E,\ {}_D0_A,\ {}_D10_D\}$;
4. $S_4 = \{{}_D0_A,\ {}_D01_B,\ {}_D01_D,\ {}_A0_A,\ {}_A01_B,\ {}_A10_D\}$;
5. $S_5 = \{...\}$;
6. $S_6 = \{{}_A0_A,\ {}_A10_D,\ {}_B10_D,\ {}_D1_C,\ {}_D101_E,\ {}_D01_B,\ {}_D10_D\}$;
7. $S_7 = S_6$;
8. $S_i = S_{i-1}$ *for* $i > 7$.

So, we end up having set S_6 *which is equal to set* S_7, *meaning that code* $\mathcal{C}$ *is a uniquely decodable code with infinite delay. It is not possible to distinguish the sequences* $CDBBB...$ *and* $EADDD...$ *until the end of the entire sequence of symbols.*

If we change just only one value of the adjacency matrix defined in Example 6 the code might become non-uniquely decodable, we see this in detail in the following example.

Example 7. *As we said we change only one value of the adjacency matrix defined before:*

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

The code for this Markov chain is the same as before so $\mathcal{C} = \{00, 001, 101, 010, 10101\}$. *The iterations of the test follow:*

1. $S_1 = \{{}_A1_B,\ {}_C01_E\}$;
2. $S_2 = \{{}_B01_C,\ {}_B0101_E,\ {}_E0_D\}$;
3. $S_3 = \{{}_C0_D,\ {}_D1_E,\ {}_D0_A,\ {}_D10_D\}$;
4. $S_4 = \{{}_D0_A,\ {}_D01_B,\ {}_D10_D,\ {}_A0_A,\ {}_E01_C,\ {}_A01_B,\ {}_A10_D,\ {}_D1_C\}$;
5. $S_5 = \{{}_D0_A, {}_D01_B, {}_A0_A, {}_A01_B, {}_A10_D, {}_D1_C, {}_B0_D, {}_C0_D, {}_D\mathbf{101}_E, {}_C01_C\}$.

Set S_5 *contains the codeword related to the symbol* C*, so making possible the transition* $D \to C$ *the code becomes not uniquely decodable. Using a software implementing the Sardinas-Patterson test (as the one in the appendix) we can carry out the sequence of codewords that generate the collision, in this case we have:*

$$\mathbf{101010010101},$$

which can be split up in two different modes:

1. $\mathbf{101} - \mathbf{010} - \mathbf{010} - \mathbf{101} \to CDDC$;
2. $\mathbf{10101} - \mathbf{00} - \mathbf{10101} \to EAE$.

Allowing the transition from state $D \to C$ *this particular sequence of coded symbols is not uniquely identified.*

Changing the last time the matrix $\boldsymbol{P}$ *by avoiding the transition from* $C \to D$*, the code is still not uniquely decodable and the colliding sequence is:*

$$\mathbf{001010010101},$$

which can be obtained with two sequences of symbols:

1. $\mathbf{001} - \mathbf{010} - \mathbf{010} - \mathbf{101} \to BDDC$;
2. $\mathbf{00} - \mathbf{101} - \mathbf{00} - \mathbf{10101} \to ACAE$.

It is interesting to see that if we do not allow the transition from $C \to D$ *the code is not uniquely decodable while if we do not allow the transition from* $D \to C$ *the code becomes uniquely decodable.*

The Sardinas-Patterson test can be a useful tool in order to have a feedback on colliding sequences and to develop a new method for the construction of uniquely decodable code for constrained sources that exploits the impossible state transitions.

1.5 Mealy and Moore Markov chains

In Figure 1.2 the Markov chain is represented in the Moore form in which the outputs of the source happen inside each state. We can represent the same Markov chain in the Mealy form, which is usually more compact. To do this, states that have the same transition probability distribution are merged together defining unique states, where output symbols are determined by state transitions. This form reduces the dimensionality of the matrix $\mathbf{P}$ and also its associated matrix $\mathbf{Q}(D)$. In the example shown in Figure 1.2, we see that

$$P_{X_i|X_{i-1}}(x|C) = P_{X_j|X_{j-1}}(x|D) \ \forall i, j > 1, \tag{1.32}$$

thanks to the fact that the Markov chain is homogeneous (time-invariant). In Figure 1.7 state γ merges together states C and D of the Moore representation because the transition probability vectors, fixing the previous state to C or D, are equal.

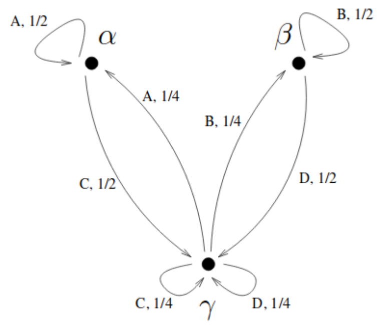

Figure 1.7: *Mealy form related to the Markov source seen in Figure 1.2, from [5].*

The output symbols are now associated to transitions. We have reduced the number of states from 4 to 3.

Theorem 4 can be adapted to the case of Markov sources in Mealy form.

Theorem 5. *(Dalai [3]) Given a Markov chain defined over a source alphabet $\mathcal{X}$ where $|\mathcal{X}| = n$, and let O_{ij} be a subset of $\mathcal{X}$ which contains all possible output symbols when a transition from state i to state j occurs. A necessary condition for the set of integers $l_1, l_2, ..., l_n$ to be lengths of a uniquely decodable code for the source is that $\rho(\mathbf{Q}) \leq 1$, where $\mathbf{Q}$ is defined by*

$$\mathbf{Q}_{ij}(D) = \begin{cases} \sum_{v \in O_{ij}} D^{-l_v} & P_{ij} > 0 \\ 0 & P_{ij} = 0 \end{cases}. \tag{1.33}$$

The proof can be carried out following the same procedure described for the Moore form (see Theorem 4).

We can define for the Mealy Markov chain, shown in Figure 1.7, its associated matrix $\mathbf{Q}$:

$$\mathbf{Q}(D) = \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} \\ 0 & D^{-l_2} & D^{-l_4} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} + D^{-l_4} \end{pmatrix}, \tag{1.34}$$

It can be easily checked that this current matrix has the same spectrum (eigenvalues) as the Moore matrix but with a lower multiplicity related to eigenvalue $\lambda = 0$.

1.5.1 Mealy-Moore equivalence

Theorem 6. *Let $\boldsymbol{Q}$ be the associated matrix to an irreducible Markov chain with transition probability matrix $\boldsymbol{P}$ and a set of N integers $\boldsymbol{l} = (l_1, l_2, ..., l_N)$ related to codeword lengths. The matrix $\boldsymbol{Q}$ expressed in both Moore and Mealy form share the same spectrum with the only difference that the eigenvalue 0 has lower or equal multiplicity in Mealy form than in Moore form.*

Proof. Let $\mathbf{Q}_{moore}$ be the matrix associated to the transition probability matrix $\mathbf{P}$ in Moore form and $\mathbf{Q}_{mealy}$ be the matrix associated with the same matrix $\mathbf{P}$ in Mealy form. The dimension of the square matrix $\mathbf{Q}_{moore}$ is N. The conversion between Moore and Mealy form can be described with the following procedure and operators:

1. define a set of indices $I = \{(k,l) \in \{1,2,...,N\}^2 : \mathbf{r}(k) = \mathbf{r}(l), l > k\}$ which contains all the indices pairs (k,l) for which the row $\mathbf{r}(k)$[2] is equal to another row $\mathbf{r}(l)$ in matrix $\mathbf{Q}_{moore}$;

2. filtering phase: let $I_\pi = \{(k,l) \in I \mid \nexists l' : (l,l') \in I\}$ (removing all the pairs that are redundant inside I);

3. define the matrix $\mathbf{T}_{ij} = \begin{cases} 1 & i = j \\ -1 & (i,j) \in I_\pi \\ 0 & otherwise \end{cases}$;

4. $\hat{\mathbf{Q}}_{mealy} = \mathbf{T} \cdot \mathbf{Q}_{moore} \cdot \mathbf{T}^{-1}$, matrix $\mathbf{T}$ represents a linear and invertible operator on $\mathbb{R}^N$;

[2] $\mathbf{r}(i)$ is the i-esim row of the associated $\mathbf{Q}$ matrix

5. let $\mathcal{R} = \{1, 2, ..., N\} \setminus \{k : (k, l) \in I_\pi\}$ be the set containing all the row-indices for which the corresponding row has to be preserved. So the reduction matrix can be defined as:

$$\mathbf{Z}_{ij} = \begin{cases} 1 & j = m(\mathcal{R}, i) \\ 0 & otherwise \end{cases},$$

with $i \in \{1, 2, ..., N - M\}$, $j \in \{1, 2, ..., N\}$ and $M = |I_\pi|$. Function $m(\mathcal{R}, i)$ returns the i-th minimum of the set $\mathcal{R}$. Matrix $\mathbf{Z}$ represents a linear non-invertible operator;

6. $\mathbf{Q}_{mealy} = \mathbf{Z} \cdot \hat{\mathbf{Q}}_{mealy} \cdot \mathbf{Z}^T$.

Lemma 1. *By construction the operator matrix $\boldsymbol{T}$ (upper triangular matrix) related to matrix $\boldsymbol{Q}_{moore}$ is always invertible and $det(\mathbf{T}) = 1$ for all the transition probability matrices and set of codeword lengths that can be associated to matrix $\boldsymbol{Q}_{moore}$.*

The matrix operator $\mathbf{Z}$ is a *reduction* operator which reduces the dimensionality of the matrix to which it is applied by removing a row and the associated column (i.e. row 3 and column 3).

The matrix $\hat{\mathbf{Q}}_{mealy}$ is a similar matrix to $\mathbf{Q}_{moore}$ thanks to the construction process. This means that these two matrices share the same eigenvalues (and also other properties). $\hat{\mathbf{Q}}_{mealy}$ has the same dimensionality of $\mathbf{Q}_{moore}$ but a certain number M of rows ($M \geq 0$) have all zero entries; this is due to the operator $\mathbf{T}$ that subtracts equal rows producing "null" rows and the operator $\mathbf{T}^{-1}$ that sums/converges probabilities into different states.
The last operation is done to obtain a matrix with lower dimensionality removing the i-th row and the i-th column (with $i : (i, l) \in I_\pi$) of $\hat{\mathbf{Q}}_{mealy}$ where the i-esim row of $\hat{\mathbf{Q}}_{mealy}$ is equal to $\mathbf{0}$. This M-reduction brings the initial dimensionality from N to $N - M$.
Let $\mathbf{Q}_{mealy}$ be matrix the result of this reduction; $\mu_1 = det(\lambda\mathbf{I} - \mathbf{Q}_{moore})$ and $\mu_2 = det(\lambda\mathbf{I} - \mathbf{Q}_{mealy})$ then the ratio of μ_1 to μ_2 is:

$$\frac{\mu_1}{\mu_2} = \lambda^M, \tag{1.35}$$

where μ_1 and μ_2 are the characteristic polynomials of Moore and Mealy matrices.

□

At the end we can see that we just need $N - M$ states to describe the Markov source instead of N states, and we have preserved the structure of the problem in the conversion from Moore to Mealy.

1.5.2 Examples of the Moore-Mealy equivalence

Let us see some applications of the Mealy-Moore equivalence with different Markovian stationary sources.

Example 8. *The first example is a trivial example where all transitions are possible. We set the source alphabet $\mathcal{X} = \{A, B, C\}$, so there are three symbols with associated codeword lengths l_1, l_2, l_3 and the Markov source is described by a transition probability matrix with all positive entries (all transitions are possible). The matrix $\boldsymbol{Q}_{moore}(D)$ in Moore form associated to this Markov source can be stated:*

$$\boldsymbol{Q}_{moore}(D) = \begin{pmatrix} D^{-l_1} & D^{-l_2} & D^{-l_3} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} \end{pmatrix},$$

where D is the cardinality of the code alphabet. In this case all rows are the same (unconstrained source), so the set I_π contains 2 pairs of indices: $I_\pi = \{(1,3),(2,3)\}$. The operator $\boldsymbol{T}$ can be derived:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}.$$

The inverse of matrix $\boldsymbol{T}$ can be easily calculated:

$$\mathbf{T}^{-1} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

The matrix $\hat{\boldsymbol{Q}}_{mealy}$ can be derived:

$$\hat{\boldsymbol{Q}}_{mealy} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} D^{-l_1} & D^{-l_2} & D^{-l_3} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\hat{\boldsymbol{Q}}_{mealy} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D^{-l_1} & D^{-l_2} & D^{-l_3} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\hat{\boldsymbol{Q}}_{mealy} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D^{-l_1} & D^{-l_2} & D^{-l_1} + D^{-l_2} + D^{-l_3} \end{pmatrix}.$$

Now that we have built the matrix $\hat{\boldsymbol{Q}}_{mealy}$ which is similar by construction to matrix $\boldsymbol{Q}_{moore}$, we can reduce the dimensionality of the Mealy *matrix removing all the rows with zero entries and their relative columns in order to maintain the same spectrum excepts for the zero eigenvalue multiplicity.*

$$\boldsymbol{Q}_{mealy} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D^{-l_1} & D^{-l_2} & D^{-l_1} + D^{-l_2} + D^{-l_3} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$

$$\boldsymbol{Q}_{mealy} = D^{-l_1} + D^{-l_2} + D^{-l_3}.$$

As we can see the reduced matrix $\boldsymbol{Q}_{mealy}$ is a scalar which contains the sum of D^{-l_i} for $i = 1, 2, 3$. This scalar is the spectral radius of the initial matrix $\boldsymbol{Q}_{moore}$. After the reduction we have lost eigenvalue 0 which in the Moore matrix had multiplicity 2.

Example 9. *Let us consider an example using the Markov chain in Moore form stated before (Figure 1.2):*

$$\boldsymbol{Q}_{moore}(D) = \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} & 0 \\ 0 & D^{-l_2} & 0 & D^{-l_4} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} & D^{-l_4} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} & D^{-l_4} \end{pmatrix}.$$

Here rows 3 and 4 are equal.
The set I_π will contain only one pair of indices $\{(3,4)\}$. The operator matrix $\boldsymbol{T}$ can be constructed:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and easily the inverse of $\boldsymbol{T}$ can be calculated:

$$\mathbf{T}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The Mealy matrix (not reduced) can be carried out as

$$\hat{\boldsymbol{Q}}_{mealy} = \mathbf{T} \cdot \boldsymbol{Q}_{moore} \cdot \mathbf{T}^{-1}.$$

The matrix $\boldsymbol{T}$ operates only on rows of the matrix $\boldsymbol{Q}_{moore}$ while the $\mathbf{T}^{-1}$ operates only on columns. Let us see the values inside the $\hat{\boldsymbol{Q}}_{mealy}$ matrix:

$$\hat{\boldsymbol{Q}}_{mealy} = \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} & D^{-l_3} \\ 0 & D^{-l_2} & 0 & D^{-l_4} \\ 0 & 0 & 0 & 0 \\ D^{-l_1} & D^{-l_2} & D^{-l_4} & D^{-l_3} + D^{-l_4} \end{pmatrix},$$

the last step consists in removing the third row and the third column, obtaining the correct Mealy matrix seen before in eq. 1.7:

$$\boldsymbol{Q}_{mealy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} & D^{-l_3} \\ 0 & D^{-l_2} & 0 & D^{-l_4} \\ 0 & 0 & 0 & 0 \\ D^{-l_1} & D^{-l_2} & D^{-l_4} & D^{-l_3} + D^{-l_4} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{Q}_{mealy} = \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} \\ 0 & D^{-l_2} & D^{-l_4} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} + D^{-l_4} \end{pmatrix}.$$

The two spectrums relative to Moore and Mealy can be calculated and they only differ by a different multiplicity of eigenvalue 0. This multiplicity differs exactly by $d = dim(\boldsymbol{Q}_{moore}) - dim(\boldsymbol{Q}_{mealy}) = 1$.

Example 10. *Last example about the equivalence between the two forms takes the following Moore matrix:*

$$\boldsymbol{Q}_{moore}(D) = \begin{pmatrix} D^{-l_1} & 0 & D^{-l_3} & D^{-l_4} & 0 \\ 0 & D^{-l_2} & D^{-l_3} & D^{-l_4} & D^{-l_5} \\ D^{-l_1} & D^{-l_2} & D^{-l_3} & 0 & 0 \\ D^{-l_1} & D^{-l_2} & D^{-l_3} & 0 & 0 \\ D^{-l_1} & 0 & D^{-l_3} & D^{-l_4} & 0 \end{pmatrix}. \tag{1.36}$$

We can see that pairs of rows: $(1,5)$ and $(3,4)$ are equal, meaning that set $I_\pi = \{(1,5),(3,4)\}$.
The Moore graph associated to $\mathbf{Q}_{moore}$ matrix follows

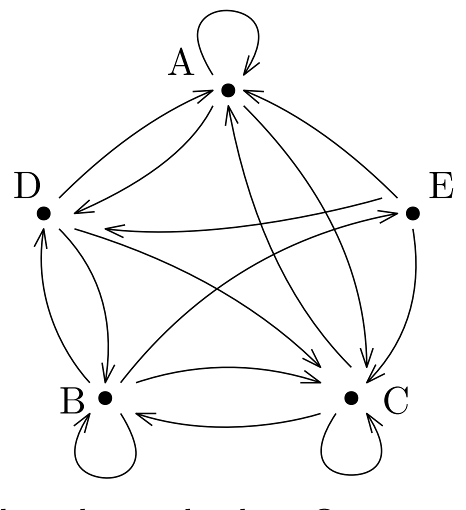

Figure 1.8: *Markov chain related to $\mathbf{Q}_{moore}$ matrix defined in eq. 1.36, where codeword lengths l_1, l_2, l_3, l_4, l_5 are associated to symbols A, B, C, D, E.*

The "transformation" matrix $\mathbf{T}$ *can be stated:*

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Inverting the -1 *values to* $+1$ *values it is a method to calculate the inverse of* $\mathbf{T}$*:*

$$\mathbf{T}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The Mealy form of this matrix can be generated thanks to the following operations:

$$\mathbf{Q}_{mealy} = \mathbf{Z} \cdot \mathbf{T} \cdot \mathbf{Q}_{moore} \cdot \mathbf{T}^{-1} \cdot \mathbf{Z}^T,$$

$$\mathbf{Q}_{mealy} = \begin{pmatrix} D^{-l_2} & D^{-l_3} + D^{-l_4} & D^{-l_5} \\ D^{-l_2} & D^{-l_3} & D^{-l_1} \\ 0 & D^{-l_3} + D^{-l_4} & D^{-l_1} \end{pmatrix}, \tag{1.37}$$

where the matrix $\mathbf{Z}$ *is constructed in order to remove rows* $1, 3$ *and columns* $1, 3$*. The new dimensionality related to Mealy form is* 3 *respect to the initial one that is* 5 *meaning that we reduce the multiplicity of the eigenvalue* 0 *by* 2*.*

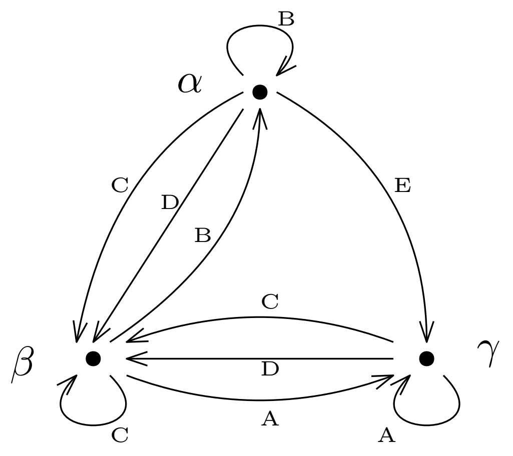

Figure 1.9: *Markov chain related to* $\mathbf{Q}_{mealy}$ *matrix defined in eq. 1.37.*

We can see from Figure 1.9 that states A *and* E *are merged into state* γ*, states* C *and* D *are merged into state* β *while state* B *remains a single state which is called* α*.*

1.6 Future works

Many open questions remain to be clarified. When we are dealing with *one-to-one* codes, it is assumed that only one symbol must be coded, codes are maps from symbols to binary strings without any requirement on concatenation of codewords. It is well known that for *one-to-one* codes the average codeword length can always be made lower than the entropy (see Wyner [13]). In this type of codes a discrete random variable X that takes values over an alphabet $\mathcal{X}$ has the following inequality always satisfied: $E[l(X)] \leq H(X)$; where $E[l(X)]$ is the expected length of the *one-to-one* code. So, there is no need to have unique decodability because there is no concatenation of codewords; it is a one to one mapping between symbols and codewords. The question that arises spontaneously: "is there another block-code, also, among variable-to-fixed codes that is better than the one we have given before for the Markov chain in Figure 1.2?". The answer is not trivial because if we can easily see that for fixed-to-variable block-codes the AEP tells us that we cannot do better than the entropy rate of the source, for variable-to-fixed codes this is not easy to show. A future work consists in proving that the code we have constructed is the best code between all the possible block-codes. This means that if we have constructed a new code which for a particular length n is better than the code seen before, there must exists a length $m \neq n$ for which the expected length of the new code is greater than the old one. In this sense, our code could be claimed to be pareto-optimal.

About the Sardinas-Patterson, further work could be done in order to use feedback information given by the test to adapt our initial non-uniquely decodable code trying to make it decodable. So, pseudo-randomic techniques can be applied for the construction of a uniquely decodable code with constrained sources. Some important information can be carried out from the sparsity of the transition probability matrix related to the Markov chain under consideration and this information can be used to describe a construction process of uniquely decodable codes (in the large sense). Methods that exploit the sparsity of the state transition matrix are not known in literature.

Appendix

1.A Moore to Mealy transformation in Matlab

We give an implementation of the Moore-Mealy transformation which follows step by step the construction process defined in Theorem 6. Let us divide the construction process in several steps:

1. given a set of codeword lengths $\mathbf{l} = \{l_1, l_2, ..., l_N\}$ with N the number of codewords, D the alphabet cardinality of our code and the adjacency matrix $\mathbf{P}^0$ related to the Markovian source that it is taking into account, we can define matrix $\mathbf{Q}_{moore} = \mathbf{P}^0 \cdot \begin{pmatrix} l_1 & 0 & \cdots & 0 \\ 0 & l_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_N \end{pmatrix}$;

2. construction of set $I = \{(k, l) : \mathbf{r}(k) = \mathbf{r}(l), l > k\}$ which contains all pairs of row-indices of matrix $\mathbf{Q}_{moore}$ that are equal;

3. construction of set $I_\pi = \{(k, l) \in I \mid \nexists l' : (l, l') \in I\}$ in which we preserve only informative pairs;

4. definition of the transformation matrix $\mathbf{T}$ which is an invertible linear operator;

5. creation of reduction matrix $\mathbf{Z}$ that removes rows and columns from Moore matrix in order to lower the initial dimensionality loosing only multiplicity of eigenvalue 0;

6. last operation is the calculation of the Mealy matrix:

$$\mathbf{Q}_{mealy} = \mathbf{Z} \cdot \mathbf{T} \cdot \mathbf{Q}_{moore} \cdot \mathbf{T}^{-1} \cdot \mathbf{Z}^T. \tag{1.38}$$

The first code snippet is about point (1) of the construction process defined before:

```
L = [2, 3, 3, 3, 5]; % lengths vector
D = 2; % cardinality of code dictionary

% Adjacency matrix
P0 = [1 0 1 1 0; 0 1 1 1 1; 1 1 1 0 0; ...
      1 1 1 0 0; 1 0 1 1 0];

% Qmoore computation
Q_moore =  P0 * diag(D.^(-L));
```

we first define an arbitrary set of codeword lengths **L** and the cardinality of our code alphabet then we can calculate the matrix $\mathbf{Q}_{moore}$ with the formula given before.

Second code snippet produces the set I (point 2 of the construction process):

```
I = {}; % initialization of set I to the empty set

%loop on all the row-pairs
for i=1:length(L)
    for j=i+1:length(L)
        if isequal(Q(i, :), Q(j, :)) == 1
            I = [I, {[i, j]}];
        end
    end
end
```

The for loop cycles on all the pairs (i, j) of rows, if one of these pairs of rows are equal then it is inserted into I.

Third code snippet is about the *filtering phase* meaning that we keep only the pairs of rows that are significant (point 3):

```
Ip = {}; % initialization of set Ip to the empty set

%loop on all the elements of I
for i=1:length(I)
    cur = I{i}; % get current pair
    b = cur(2); % take the second element of the pair
    ins = 1; % boolean value (1-insert, 0-not insert)
    for j=1:length(I)
        comp = I{j};
```

```
            a = comp(1);
            if a == b
                ins = 0;
                break;
            end
        end
        if ins == 1
            % adding an element to the set
            Ip = [Ip, {I{i}}];
        end
end
```

so, we cycle on all the elements of I and we insert the pair indexed by i into the set $I_p(I_\pi)$ if and only if does not exist another pair in I that has the first element equal to the second element of i-esim pair.

Fourth code snippet is about the construction process of the transformation matrix $\mathbf{T}$ (point 4):

```
T = zeros(size(P0)); % initialize to zero-matrix T

% loop on all the positions of T
for i=1:size(T,1)
    for j=1:size(T,2)
        if i == j  % main diagonal
            T(i,j) = 1;
        else
            for k=1:length(Ip)
                cur = Ip{k};
                % (i, j) belongs to Ip
                if i == cur(1) && j == cur(2)
                    T(i, j) = -1;
                    break;
                end
            end
        end
    end
end
```

first of all, we initialize the matrix $\mathbf{T}$ to a zero-matrix (all entries equal to 0) then we cycle on all the positions (i, j) and we change the value in the current position to 1 if the position is on the main diagonal while we change the value to -1 when $(i, j) \in I_\pi$.

Fifth code snippet produce a set which contains the indices of rows that must be preserved during the reduction process:

```
indices = []; % initialize the set to empty vector

% loop from 1 to the number of codewords
for i=1:length(L)
    found = 0;
    for j=1:length(Ip)
        cur = Ip{j};
        if i == cur(1)
            found = 1;
            break;
        end
    end
    if found == 0
        indices = [indices; i];
    end
end
```

this code simply implements a set difference between $\{1, 2, ..., N\}$ (set containing all integer numbers from 1 to the number of codewords N) and the set induced by I_π (taking only the first element from all the pairs). The set of indices is implemented as a vector so each inside element it's listed in ascending order.

Sixth code snippet (point 5) defines the construction of **Z** matrix:

```
% initialize matrix Z with correct dimension
Z = zeros(length(indices), size(T, 2));

% loop on all the positions of Z
for i=1:size(Z,1)
    for j=1:size(Z,2)
        % if j is equal to i-esim minimum of indices
        if j == indices(i)
            Z(i, j) = 1;
        end
    end
end
```

this matrix is first initialize to a zero-matrix, then we cycle on all its positions (i, j) and we set the value in that position to 1 if and only if j is equal to the i-esim minimum of the set of indices.

The last code snippet is related to the computation of the Mealy matrix:

```
Q_mealy = Z * T * Q_moore * inv(T) * Z'
```

we apply the formula given in eq. 1.38 using built-in function of matlab to compute the inverse of matrix **T**.

1.B Implementation of Sardinas-Patterson test

In this section we present a Matlab implementation of the extended Sardinas-Patterson test seen in previous sections. The algorithm follows exactly the same steps describe in the mentioned section. Let us show some code snippets from the matlab implementation.

First code snippet:

```
MAX = 1000; % max value that variable n can assume
C = {'0', '1', '01', '10'}; % initial code
% adjacency matrix example
A = [1 0 1 0; 0 1 0 1; 1 1 1 1; 1 1 1 1];
F = cell(1, length(C));
```

first of all we have to initialize the needed data structures. $\mathcal{C}$ is the code under test, **A** is the adjacency matrix where $\mathbf{A}_{ij} = 1$ if $\mathbf{P}_{ij} > 0$ and $\mathbf{A}_{ij} = 0$ otherwise. The variable $\mathcal{F}$ is a cell-array and contains different sets $\mathcal{F} = \{\mathcal{F}_i\}_{i=1,\dots,n}$ where $n = \text{length}(\mathcal{C})$ is the number of codewords in $\mathcal{C}$.

Second code snippet:

```
for i=1:length(C)
    F{i} = {}; % initialize each set to the empty set
    for j=1:length(C)
        % check if the transition is possible
        if A(i, j) = 1
            add = cell(1, 2);
            add{1} = C{j};
            add{2} = j;
            % add a cell that contains c_j word
            F{i} = [F{i}, {add}];
        end
    end
end
```

this section of the matlab code is used for the construction of the sets $\mathcal{F}_i$. A generic set $\mathcal{F}_i$ contains all the codewords that can follow i-esim symbol:

$$\mathcal{F}_i = \{c_j \in \mathcal{C} \mid \mathbf{A}_{ij} = 1\}.$$

So the outer loop cycles on all the codeword indices $i = 1, ..., n$, the set $\mathcal{F}_i$ is initially set to the empty-set and then will be populated in the following loop. The inner loop cycles again on all the codeword indices $j = 1, ..., n$ and add to the set $\mathcal{F}_i$ the codeword c_j and the index j (will be useful later) if and only if the the state transition from i to j is possible.

Third code snippet:

```
% definition of the sequence of sets
St = cell(1, MAX);

% loop on all the codewords c_i
for i=1:length(C)
    for j=1:length(C) % loop on all the codewords c_j
        if length(C{j}) <= length(C{i})
            continue;
        end
        len = length(C{i});
        cmp = C{j};

        % check if c_i is a prefix of c_j
        if strcmp(C{i}, cmp(1:len)) == 1
            add = cell(1, 3);
            add{1} = i;
            add{3} = j;
            add{2} = cmp(len+1:end);

            % building the possible colliding sequence
            add{4} = strcat(C{i}, add{2});

            % add the residual suffix to S_1
            St{1} = [St{1}, {add}];
        end
    end
end
```

this matlab code is used to construct the set S_1 (seen in the related chapter) which consists cycling on all pairs of codewords $(c_i, c_j) \in \binom{\mathcal{C}}{2}$ and check if codeword c_i is a prefix of a codeword c_j meaning that $c_j = c_i A$ where A is the residual suffix of c_j; if this condition is met then the suffix A is inserted into S_1 set. Together with suffix A are saved in S_1 also the two indices i, j keeping trace of the codewords that generate such suffix. In all the matlab snippets we used cell-arrays instead of simple arrays because the non-homogeneity (in length and type) of the data values.

Fourth code snippet:

```
n = 2; % starting from 2
while n < MAX
    S_new = St{n}; % taking the S_n set
    S = St{n-1}; % taking the S_{n-1} set
    for i=1:length(S)
        from = S{i}{1};
        to = S{i}{3};
        B = S{i}{2};
        F_from = F{from};
        cod = S{i}{4};
```

in this code section we have an outer while loop that cycles on n, the number of building sets, which will be broken under some conditions (see in further code sections). The inner for loop cycles on each codeword ${}_lH_m \in S_{n-1}$ where l value is defined in variable *from* and m value is defined in variable *to*. Then we assign to variable *F_from* the set $\mathcal{F}_l$ which contains all the codewords c_j that can follow the codeword c_i (possible transitions $\mathbf{A}_{ij} = 1$). Variable *delay* is initially set to 1 (delay is the number of bits that the decoder needs in order to decode the correct symbol).

Fifth code snippet:

```
for k=1:length(F_from)
    if length(B) == length(F_from{k}{1}) && ...
        strcmp(B, F_from{k}{1}) == 1

        disp('Code␣is␣not␣uniquely␣decodable');

        cod
        getSequence(cod, C, X, '')
        return;
    end
end
```

this code section is about checking if the code $\mathcal{C}$ under test is not a uniquely decodable code. This phase consists in comparing all the codewords inside the set $\mathcal{F}_l$ with the codeword ${}_lH_m$, if the two codewords are equal then the algorithm stops and produce a video message saying: "*Code is not uniquely decodable*". The condition can be simply seen as:

$$c({}_lH_m, \mathcal{F}_l) = \begin{cases} 1 & H \in \mathcal{F}_l \\ 0 & otherwise \end{cases} .$$

If the code is not uniquely decodable a message containing the colliding sequence will appear on the screen and the sequences of symbols that generate such sequence will be calculated using function **getSequence** (dealt later).

Sixth code snippet:

```
for r=1:length(F_from)
      len = length(B);
      cmp = F_from{r}{1};
      if len < length(cmp) && ...
      strcmp(B, cmp(1:len)) == 1
            add = cell(1, 3);
            add{1} = to;
            add{3} = F_from{r}{2};
            add{2} = cmp(len+1:end);
            add{4} = strcat(cod, add{2});
            S_new = [S_new, {add}];
      end
end
```

this code section is about building the S_n set of codewords starting from the set S_{n-1} and the set $\mathcal{F}_l$. The for loop cycles on all the codewords $c_r \in \mathcal{F}_l$ and there is a condition that checks if these codewords are prefixed by ${}_lH_m$ meaning that $c_r = {}_lH_mC$, where C is the residual suffix generated by the previous operation. We then add the labelled suffix ${}_mC_r$ to the set S_n.

Seventh code snippet:

```
for s=1:length(F_from)
      len = length(F_from{s}{1});
      cmp = F_from{s}{1};
      if len <  length(B) && strcmp(B(1:len),cmp) == 1
            add = cell(1, 3);
            add{1} = F_from{s}{2};
            add{3} = to;
            add{2} = B(len+1:end);

            S_new = [S_new, {add}];
      end
end
```

this code section is about adding to the S_n set some codewords starting from the set S_{n-1} and the set $\mathcal{F}_l$. The for loop cycles on all the codewords $c_s \in \mathcal{F}_l$ and there is a condition that checks for each of these codewords if they are prefix of codeword ${}_lH_m$ meaning that ${}_lH_m = c_sD$ where D is the residual suffix generated by the previous operation. We then add the labelled suffix ${}_sD_m$ to S_n.

Eighth code snippet:

```
 St{n} = S_new;

 if isempty(S_new) == 1 || checkequal(St, n) == 1
         break;
 end

 n = n + 1;
```

we have the last condition where we check if set $S_n = \emptyset$ (*finite delay*) or if there is a previous set S_{n-k} with $0 < k < n$ that is equal to the current set S_n(*infinite delay*), the condition can be expressed mathematically in this way:

$$c(S_n, S_1^{n-1}) = \begin{cases} 1 & S_n = \emptyset \\ 1 & \exists k = 1, ..., n-1 : S_n = S_k \\ 0 & \textit{otherwise} \end{cases},$$

where S_1^{n-1} represents the vector of all the sets from 1 to $n-1$. The function implementation of **checkequal** follows:

```
function equal = checkequal(S, n)

      S2 = S{n}; % setting S_2 equal to S_1

      % loop on all S_k with k < n
      for k=(n-1):-1:1
            S1 = S{k};
            if length(S1) != length(S2)
                  continue;
            end
```

```
        bzero = ones(1, length(S1));
```

this function has two parameters the vector $\mathbf{S} = (S_1, S_2, ..., S_n)$ which contains all the sets S_i from 1 to n (n is the length of the vector). The for loop cycles on all the sets S_k with k that varies from $n-1$ to 1. The main goal of this function is to check if a set S_k is equal to the set S_n. First condition that we must verify is on the lengths of the sets S_n and S_k if these lengths differ we will check the next set S_{k-1} and we continue until we find a set that is equal to S_n or the value $k < 1$.

We initialize the vector *bzero* to a vector that contains in all positions 1 (this will be useful to check if the two sets are equal).

```
for i=1:length(S1)
    from_1 = S1{i}{1};
    to_1 = S1{i}{3};
    cmp_1 = S1{i}{2};
    for j=1:length(S2)
        from_2 = S2{j}{1};
        to_2 = S2{j}{3};
        cmp_2 = S2{j}{2};

        % checking if the two words are equal
        if bzero(j) == 1 && from_1 == from_2 && ...
        to_1 == to_2 && strcmp(cmp_1, cmp_2) == 1

            bzero(j) = 0;

        end
    end
end

if sum(bzero) == 0
    equal = 1;
    return;
end

equal = 0;
```

In order to check if the set S_n and S_k are equal we have to cycle on each i-esim codeword ${}_lH_m$ of S_k and on each j-esim codeword ${}_nI_o$ of S_n. We then check if $l = n$, $m = o$ and $H = I$, if so, we set the vector *bzero* in position j equal to 1. At the end of the outer loop we check if the vector *bzero* has zero entries in all its positions, in this case, there is a match between set S_n and a set S_k and the function returns a true boolean value, otherwise if we have scanned all the sets S_k the function returns a false boolean value.

We give now the implementation of the function **getSequence** seen before that produces the colliding sequences of symbols for a code that is not uniquely decodable.

```
function [] = getSequence( seq, code, X, build )

   % loop on all the codewords
   for j=1:length(code)
       len = length(code{j});

       % check if the codeword is inside the sequence
       if  len <= length(seq) && strcmp(code{j}, ...
           seq(1:len)) == 1

           if len == length(seq) && ...
              isempty(seq) == 0

               ret = strcat(build, X{j});
               ret
               return
           else
               getSequence(seq(len+1:end), ...
               code, X, strcat(build, X{j}))
           end
       end
   end
end
```

This is a recursive function whose parameters are: the ambiguous code sequence (codeword symbols), the code $\mathcal{C}$ used for the encoding, the source alphabet $\mathcal{X}$ and variable *build* that contains the different colliding sequences of source symbols. All the colliding sequences are printed on the screen because with Matlab it is difficult to pass variable by reference (or almost impossible). In our case it was only necessary to know the colliding sequences so that some retrieved feedback information might be useful to update current code set making it become a uniquely decodable code.

Chapter 2

Fix-Free Codes

In this chapter we talk about fix-free codes which are codes where no codeword is a prefix or suffix of any other codeword. Fix-free codes have the advantage to be instantaneously decodable from both sides. The Kraft inequality, unfortunately, is not a necessary and sufficient condition for building fix-free codes, so researchers have tried to find/prove different upper bounds for the Kraft sum in order to define a sufficient condition for the construction of fix-free codes with given codeword lengths.

2.1 Basics

The notation and the definitions are essentially the same stated by Ahlswelde [7]. Given a finite set $\mathcal{X}$, which is the source alphabet, let $\mathcal{X}^n$ be the set of words of length n with symbols in $\mathcal{X}$.

Definition 18. *Given $\mathcal{X}^* = \cup_{n=0}^{\infty}\mathcal{X}^n$, where $\mathcal{X}^0 = \{e\}$ that is the empty word.*

$\mathcal{X}^*$ has an associative operation, called concatenation:

$$(x_1, ..., x_n)(y_1, ..., y_m) = (x_1, ..., x_n, y_1, ..., y_m).$$

Definition 19. *Let $\mathcal{X}^+$ be the set of non-empty words:*

$$\mathcal{X}^+ = \mathcal{X}^* \setminus \{e\},$$

Definition 20. *A word $w \in \mathcal{X}^*$ is a **factor** of word $x \in \mathcal{X}^*$ iff $\exists u, v \in \mathcal{X}^*$ such that $x = uwv$.*

Definition 21. *A factor w of x is called **proper** iff $w \neq x$.*

This means that given def. 20, u or v is different from the empty word.

Definition 22. *A set of words $\mathcal{C} \subset \mathcal{X}^*$ is called a **code**.*

If we have two subsets $\mathcal{Y}, \mathcal{Z} \subset \mathcal{X}^*$ then:

$$\mathcal{Y}\mathcal{Z} = \{yz \in \mathcal{X}^* : y \in \mathcal{Y}, z \in \mathcal{Z}\}.$$

Definition 23. *A code that is simultaneously prefix-free and suffix-free is called fix-free or biprefix:*

$$\mathcal{C}\mathcal{X}^+ \cap \mathcal{C} = \emptyset \ \wedge \ \mathcal{X}^+\mathcal{C} \cap \mathcal{C} = \emptyset$$

Definition 24. *A code $\mathcal{C}$ with $|\mathcal{C}| = N$ over a binary alphabet $\mathcal{X}$ is said to be **complete** iff the Kraft inequality is satisfied with equality:*

$$\sum_{i=1}^{N} 2^{-l_i} = 1$$

Definition 25. *A fix-free code $\mathcal{C}$ is said to be **saturated** iff it is impossible to find a fix-free code $\mathcal{C}'$ for which $\mathcal{C} \subset \mathcal{C}'$ (strictly contains).*

Definition 26. *The **shadow** of a word $w \in \mathcal{X}^*$ in **level** l is defined:*

$$\delta_l(w) = \{x^l \in \mathcal{X}^l : w \text{ is prefix or suffix of } x^l\}.$$

2.2 Modified Kraft inequality upper bound

Theorem 7 (Ahlswelede [7]). *The condition*

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1/2, \tag{2.1}$$

implies there exists a fix-free code $\mathcal{C}$ over the binary alphabet $\mathcal{X}$ with ordered codeword lengths $l_1 \leq l_2 \leq ... \leq l_N$.

Proof. Proceeding by induction in the number of codewords, for $N = 1$ is trivial, so we then assume that we found a fix-free code for $N - 1$ codewords. Like the proof for the Kraft inequality with prefix-free codes, we build a binary tree (for binary alphabet) in which each word is a vertex in the tree; a word of length l will be placed at l-th level in the tree.
The idea now is to count all the leaves at l_N-th level, which have one of the codewords as a prefix or a suffix (shadow). For each codeword c_i of length l_i we have $2^{l_N - l_i}$ leaves that have c_i has a prefix and the same amount that have it as a suffix.

The worst case is when the shadow of prefixed leaves and the shadow of suffixed leaves are disjoint but the total number of leaves is always less or equal to $2\sum_{i=1}^{N-1} 2^{-l_i}$. This quantity is smaller than 2^{l_N} by assumption and so there is a leaf on the l_N-th level which is not counted, and so can be used for our N-th codeword.

□

Theorem 8. *Ahlswede [7] et al. conjecture that the largest constant γ for which the fix-free sufficient condition that implies the existence of a fix-free code with lengths $l_1, l_2, ..., l_N$ in the Kraft sum is $\gamma \leq 3/4$.*

Proof by Ahlswede. The proof of the non-existence of a fix-free code for $\gamma > \frac{3}{4}$ can be easily shown: Let $\gamma = \frac{3}{4} + \epsilon$ with $\epsilon > 0$, choose k such that $2^{-k} < \epsilon$ (we can make ϵ as small as we want). Setting vector $(l_1, l_2, l_3, ..., l_N) = (1, k, k, ..., k)$ with $N = 2^{k-2} + 2$ then:

$$\sum_{i=1}^{N} 2^{-l_i} = \frac{1}{2} + 2^{-k} \cdot (2^{k-2} + 1) = \frac{3}{4} + 2^{-k} < \frac{3}{4} + \epsilon$$

But we can build only 2^{k-2} words of length k with fix-free property and, since are required $1 + 2^{k-2} < N$ words of length k then the theorem is proved.

□

Using some constraints on the lengths of the codewords Ahlswede proved in Theorem 9 the existence of a fix-free code with a Kraft sum less or equal to 3/4.

Theorem 9. *Ahlswede [7] proved that under the conditions*

$$l_i = l_{i+1} \vee 2l_i \leq l_{i+1} \; \forall i = 1, 2, ..., N.$$

the existence of a fix-free code with codeword lengths $l_1, l_2, ..., l_N$ is implied by the Kraft-like inequality $\sum_{i=1}^{N} 2^{-l_i} \leq \frac{3}{4}$.

Proof. The proof is again done using induction. Sort the codeword lengths from the smaller one to the bigger one $l_1 \leq l_2 \leq ... \leq l_N$.
For $N = 1$ the construction a fix-free code is obvious. Suppose that we construct a fix-free code with $N - 1$ different codeword lengths; we have to prove that we can add a new codeword without violating the fix-free property.
Call M the largest index i for which $l_i < l_N$. In this way we can exclude all the codewords lengths where $l_i = l_N$. By construction we know that $\sum_{i=1}^{M} 2^{-l_i} \leq \frac{3}{4}$ and by induction we have a fix-free code $\mathcal{C}'$ with lengths

$l_1 \leq l_2 \leq ... \leq l_M$. Thanks to the constraint on the lengths we can produce an exact formula for the cardinality of the shadow $|\delta_{l_N}(\mathcal{C}')|$:

$$|\delta_{l_N}(\mathcal{C}')| = 2\sum_{i=1}^{M} 2^{l_N - l_i} - 2^{l_N}(\sum_{i=1}^{M} 2^{-2l_i} + 2\sum_{1\leq i<j\leq M} 2^{-l_i - l_j}). \tag{2.2}$$

This formulation can be derived knowing that the maximum cardinality of the shadow $\delta_{l_N}(\mathcal{C}')$ is $2\sum_{i=1}^{M} 2^{l_N - l_i}$ (when the set of prefixes and suffixes are disjoint sets). Then we have to remove all the leaves counted twice, removing first all the leaves corresponding to codewords in which the prefix and suffix are equal to a certain codeword i, the second step is removing all the leaves corresponding to codewords in which the prefix is equal to codeword i and suffix is equal to codeword j and vice versa, this can be done without problems because the codeword lengths satisfy the initial conditions, meaning that if we sum two different codewords lengths l_i, l_j, their sum does not exceed l_N. The code can be constructed if:

$$|\delta_{l_N}(\mathcal{C}')| \leq 2^{l_N} - (N - M).$$

The quantity $N - M$ identifies the number of the maximum codeword lengths ($l_i = l_N$). Setting $K = N - M$ and $\alpha = \sum_{i=1}^{M} 2^{-l_i}$ we get:

$$2\alpha - \alpha^2 \leq 1 - \delta,$$

with $\delta = \frac{K}{2^{l_N}}$, setting $\beta = \sum_{i=1}^{N} 2^{-l_i} = \alpha + \delta$ we get:

$$(\alpha - 1)^2 \geq \delta,$$

we kept only the upper-bound of this second-grade inequality. Substitute $\alpha = \beta - \delta$ we obtain:

$$\beta \leq 1 + \delta - \sqrt{\delta}.$$

So we have a function $f(\delta) = 1 + \delta - \sqrt{\delta}$ on the right hand-side of the inequality; to find the minimum upper-bound we have to calculate the derivative of $f(\delta)$ and imposing it equal to 0:

$$f'(\delta) = 1 - \frac{1}{2\sqrt{\delta}} = 0.$$

We found $\delta = \frac{1}{4}$ and inserting it in the β inequality we found that:

$$\beta \leq \frac{3}{4} \implies \sum_{i=1}^{N} 2^{-l_i} \leq \frac{3}{4}.$$

□

This means that we can always construct a fix-free code if the Kraft sum is less or equal to 3/4 when we have the set of codeword lengths that respect the condition seen in Theorem 9. In this condition the selection of different words used to construct the code does not affect the success of the construction of the code. As we will see later, Yekhanin drops this condition on the lengths of the codewords in order to prove a generalize method of code costruction (with a different upper bound), the right selection of codewords will influence the construction process.

In Coding Theory, it is useful to know the minimum expected length of the codewords. If we are talking about Huffman codes it is well known that for a probability distribution $P(X = x)$ where X is a random variable over an alphabet $\mathcal{X}$:

$$H(X) \leq E[l(X)] \leq H(X) + 1.$$

Instead if we are talking about fix-free codes these inequalities change a little bit.

Theorem 10. *Let $P(X = x)$ be a probability distribution of a random variable X over an alphabet $\mathcal{X} = \{1, 2, ..., N\}$, there exists a binary fix-free code $\mathcal{C}$ satisfying*

$$H(X) \leq E[l(X)] < H(X) + 2$$

Proof. The left inequality is immediately proved because a fix-free code is also a prefix-free code.
The right inequality can be proved defining $l_i = \lceil -\log_2(P(X = i)) \rceil + 1$:

$$\sum_{i=1}^{N} 2^{-l_i} \leq \frac{1}{2} \sum_{i=1}^{N} 2^{\log_2(P(X=i))} = \frac{1}{2} \sum_{i=1}^{N} P(X = i) = \frac{1}{2}.$$

By Theorem 7 there always exists a fix free code $\mathcal{C}$ with codewords lengths $l_1, l_2, ..., l_N$.
The expected length of this code is:

$$E[l(X)] = \sum_{i=1}^{N} P(X = i) \cdot l_i$$
$$<$$
$$-\sum_{i=1}^{N} P(X = i)(\log_2(P(X = i)) + 2) = H(X) + 2$$

$\square$

This means that, given a probability mass function related to the symbols of an alphabet, we can create a fix-free code in which the expected length of the codeword is always less than the entropy over the probability distribution under consideration plus 2.

2.3 Improved upper bound of Kraft sum

In this section we describe the work of Yekhanin [8] on the improvement of the upper bound related to the Kraft sum respects to the 1/2 upper bound for unconstrained codeword lengths. We have proved in the previous section that a fix-free code can always be constructed when the Kraft sum is less than or equal to 1/2 (Theorem 7). Ahlswede proved under some constraints on codeword lengths that the Kraft sum less than or equal to 3/4 is a sufficient condition for the construction of a fix-free code. Yekhanin proved without any constraints on codeword lengths that when the Kraft sum is less or equal to 5/8 then a fix-free code can be constructed. It is important to notice that we are talking about sufficient conditions, so if the set of codeword lengths satisfy the Kraft sum upper bound, then a code can be constructed but the converse part does not hold with respect to the prefix-free codes.
Ahlswede and Yekhanin use slightly different notations when they define codes. The notation used by Yekhanin is more efficient and facilitates his proof so we follow his one, which is described below.

Definition 27. *Let $\mathcal{C}(\boldsymbol{v}_n)$ be a binary variable-length code, where $\boldsymbol{v}_n = (k_1, ..., k_n)$ is a vector of non-negative integers and each k_i defines the number of codewords of length i for all $i = 1, ..., n$.*

Definition 28. *The Kraft sum associated to vector of integers $\boldsymbol{v}_n$ is defined as*

$$S(\boldsymbol{v}_n) = \sum_{i=1}^{n} k_i 2^{-i}.$$

Definition 29. *Let $\boldsymbol{w}$ be a binary vector of length n. We call p-prefix of $\boldsymbol{w}$ the first p-symbols of $\boldsymbol{w}$ ans we denote by ${}^{p}\boldsymbol{w}$. Conversely we called p-suffix of $\boldsymbol{w}$ the last p-symbols of $\boldsymbol{w}$ and we denote by $\boldsymbol{w}^{p}$.*

Definition 30. *Fixing a binary fix-free code $\mathcal{C}(\boldsymbol{v}_n)$, we define four different sets:*

$${}^{0}\overrightarrow{F}(\mathcal{C}) = \{\boldsymbol{w} : \boldsymbol{w} \textit{ is prefix-free over } \mathcal{C} \textit{ and } {}^{1}\mathbf{w} = 0\};$$

$${}^{1}\overrightarrow{F}(\mathcal{C}) = \{\boldsymbol{w} : \boldsymbol{w} \textit{ is prefix-free over } \mathcal{C} \textit{ and } {}^{1}\mathbf{w} = 1\};$$

These two sets represent the words that are prefix-free over the code C, differentiating by the first bit of the words. The words that start with bit '0' are collocated in ${}^{0}\overrightarrow{F}(\mathcal{C})$ while words starting with bit '1' are collocated in ${}^{1}\overrightarrow{F}(\mathcal{C})$.

The union of such sets is called $\overrightarrow{F}(\mathcal{C}) = {}^0\overrightarrow{F}(\mathcal{C}) \cup {}^1\overrightarrow{F}(\mathcal{C})$*. The converse part for suffix-free words follows:*

$$\overleftarrow{F}^0(\mathcal{C}) = \{\boldsymbol{w} : \boldsymbol{w} \textit{ is suffix-free over } \mathcal{C} \textit{ and } \mathbf{w}^1 = 0\};$$
$$\overleftarrow{F}^1(\mathcal{C}) = \{\boldsymbol{w} : \boldsymbol{w} \textit{ is suffix-free over } \mathcal{C} \textit{ and } \mathbf{w}^1 = 1\};$$

The union of such sets is called $\overleftarrow{F}(\mathcal{C}) = \overleftarrow{F}^0(\mathcal{C}) \cup \overleftarrow{F}^1(\mathcal{C})$*.*

Definition 31. *Let P be a subset of $\{0,1\}^n$. P is said* ***right regular*** *iff:*

$$\forall c_1, c_2 \in P,\ c_1 \neq c_2 \implies c_1^{n-1} \neq c_2^{n-1}$$

Definition 32. *Let P be a subset of $\{0,1\}^n$. P is said* ***left regular*** *iff:*

$$\forall c_1, c_2 \in P,\ c_1 \neq c_2 \implies {}^{n-1}c_1 \neq {}^{n-1}c_2$$

It can be easily seen that ${}^0\overrightarrow{F}(\mathcal{C})$ and ${}^1\overrightarrow{F}(\mathcal{C})$ are **right regular** sets while $\overleftarrow{F}^0(\mathcal{C})$ and $\overleftarrow{F}^1(\mathcal{C})$ are **left regular** sets.

Definition 33. *Let $\otimes : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+1}$ be an operator defined by*

$$P_1 \otimes P_2 = \{\boldsymbol{w} \in \{0,1\}^{n+1} : {}^n\boldsymbol{w} \in P_1 \wedge \boldsymbol{w}^n \in P_2\}, \quad P_1, P_2 \in \{0,1\}^n.$$

This operator builds a new set of codewords starting from two sets in which words overlap in $(n-2)$ positions (excluding only the first bit and the last bit), this will be useful when we have to construct a code at a certain step t from a fix-free code already built at step $t-1$.

Theorem 11. *Fixing a $\mathcal{C}(\boldsymbol{v}_n)$ code, then $\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}(\mathcal{C})$ is the set of all words that can be added to $\mathcal{C}(\boldsymbol{v}_n)$ without violating the fix-free property.*

Theorem 12. *Supposing that $P_1 \subseteq \{0,1\}^n$ is a right-regular set and $P_2 \subseteq \{0,1\}^n$ is a left-regular set then:*

$$|P_1 \otimes P_2| \geq |P_1| + |P_2| - 2^{n-1}$$

Proof. We know from basic algebra that $|P_1 \cup P_2| = |P_1| + |P_2| - |P_1 \cap P_2|$. Knowing that P_1 is a right-regular set then $|P_1| = |P_1^{(n-1)}|$ (see Definition 31), and knowing that P_2 is a left-regular set then $|P_2| = |{}^{(n-1)}P_2|$.
From this assumption we can define an upper bound on the union of these two sets:

$$|P_1^{(n-1)} \cup {}^{(n-1)}P_2| \leq 2^{n-1}.$$

The last inequality can be used to find a lower bound of the intersection between these sets:

$$|P_1^{(n-1)} \cap {}^{(n-1)}P_2| \geq |P_1| + |P_2| - 2^{n-1}.$$

It can be easily seen that $P_1 \cap P_2 \subseteq P_1 \otimes P_2$.

□

The main theorem stated by Yekhanin is the following one:

Theorem 13 (Yekhanin [8]). *If $S(\boldsymbol{v}_n) \leq \frac{5}{8}$, then there exists a fix-free code $\mathcal{C}(\boldsymbol{v}_n)$.*

Proof. The proof is basically split in 3 parts:

1. $k_1 = 1$, meaning that we have a $\mathcal{C}(\mathbf{v}_n)$ with one codeword of length 1, and all the other k_i can assume arbitrarily values;
2. $k_1 = 0, k_2 = 2$, meaning 0 codewords of length 1 and 2 codewords of length 2;
3. $k_1 = 0, k_2 \leq 1$, meaning 0 codewords of length 1 and 0 or 1 codeword of length 2.

These 3 cases cover the entire space of k_i's. Proving that $S(\mathbf{v}_n) \leq \frac{5}{8}$ holds for each of these cases, it means we have proved Theorem 13.
The method for the proof is always by induction (like we have seen with Ahlswede). So we have to construct a code $\mathcal{C}(\mathbf{v}_n)$ in n steps. At step t we have to add k_t codewords of length t without violating the fix-free property. So we have to construct a code at step t knowing a fix-free code at step $t-1$.

Proof of case 1. In this particular case the proof can be done considering not the target inequality of this proof $S(\mathbf{v}_n) \leq \frac{5}{8}$ but directly the conjectured one $S(\mathbf{v}_n) \leq \frac{3}{4}$.
Setting the initial fix-free code $\mathcal{C}(\mathbf{v}_1) = \{0\}$ and supposing that we found a fix-free code $\mathcal{C} = \mathcal{C}(\mathbf{v}_{t-1})$ at step $t-1$, we have to prove that at step t adding k_t codewords of length t we do not violate the fix-free constraint. It is sufficient to prove that $|\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}(\mathcal{C})| \geq k_t$ for each $t \leq n$.
We define the partial sum at step t as $\delta = S(\mathbf{v}_{t-1})$, so at step t the inequality becomes:

$$\delta + k_t \cdot 2^{-t} \leq \frac{3}{4}.$$

Expressing the last inequality in terms of an upper bound for k_t we find that:

$$k_t \leq 3 \cdot 2^{t-2} - 2^t \cdot \delta.$$

The fundamental key of this proof is that $0 \in \mathcal{C}(\mathbf{v}_{t-1})$, which implies that

$${}^0\overrightarrow{F}(\mathcal{C}) = \overleftarrow{F}^0(\mathcal{C}) = \emptyset.$$

A consequence of the last statement is that $\overrightarrow{F}(\mathcal{C})$ is a right regular set and $\overleftarrow{F}(\mathcal{C})$ is a left regular set. So the cardinality of $|\overrightarrow{F}(\mathcal{C})| = |\overleftarrow{F}(\mathcal{C})| =$

$2^{t-1} \cdot (1-\delta)$. This is obtained considering δ as a capacity coefficient (ex. $\delta = 1$ means that the code is saturated), so at step $t-1$ the total possible codewords are 2^{t-1} but only the remaining available codewords can be selected $(1-\delta)$. Using the Definition 33 of the $\otimes$ operator we derive that:

$$\begin{aligned}|\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}(\mathcal{C})| &\geq |\overrightarrow{F}(\mathcal{C})| + |\overleftarrow{F}(\mathcal{C})| - 2^{t-2}\\ &= 2 \cdot 2^{t-1}(1-\delta) - 2^{t-2}\\ &\geq 3 \cdot 2^{t-2} - 2^{t} \cdot \delta.\end{aligned}$$

That we can rephrase as

$$|\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}(\mathcal{C})| \geq k_t.$$

□

Proof of case 2. Also for this case we can use the conjectured constraint for fix-free codes $S(\mathbf{v}_n) \leq \frac{3}{4}$.
This time we have 2 codewords of length 2, so our initial code $\mathcal{C}(\mathbf{v}_2) = \{00, 11\}$. Operating by induction we suppose that a fix-free code $\mathcal{C} = \mathcal{C}(\mathbf{v}_{t-1})$ at step $t-1$ is constructed. We have always to prove that $|\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}(\mathcal{C})| \geq k_t$, meaning that there are enough available codewords to assign at step t.
Like the previous proof we have to show that $\overrightarrow{F}(\mathcal{C})$ is a right-regular set and $\overleftarrow{F}(\mathcal{C})$ is a left-regular set (thanks to the two initial words 00 and 11 that are "orthogonal" meaning that '00' $\oplus$ '11' = 11 where $\oplus$ is the binary sum). Note that for this proof we could use also another set of initial words $\{01, 10\}$.
A more formal proof can be derive using indirect proof, so suppose that $\overrightarrow{F}(\mathcal{C})$ is a non right-regular set. This means that exists a vector $\mathbf{b} \in \{0,1\}^{t-2}$ such that both words '0$\mathbf{b}$' and '1$\mathbf{b}$' are prefix free over $\mathcal{C}(\mathbf{v}_{t-1})$. If ${}^{0}\mathbf{b} = 0$ then 0$\mathbf{b}$ is prefixed by codeword 00 while if ${}^{1}\mathbf{b} = 1$ means that 1$\mathbf{b}$ is prefixed by codeword 11; this leads to a contradiction and thus $\overrightarrow{F}(\mathcal{C})$ is a right-regular set. This procedure can be carried out also for the $\overleftarrow{F}(\mathcal{C})$ set proving that it is a left-regular set.
Under these considerations the exact same inequalities seen in the previous proof can be carried out and so the theorem is proved.

□

Proof of case 3. This is the main part of the proof where the upper bound of the Kraft sum is set to 5/8 ($S(\mathbf{v}_n) \leq \frac{5}{8}$). Since $k_1 = 0$ and $k_2 \leq 1$, Yekhanin made a clever splitting of the vector $\mathbf{v}_n$ into four vectors

$\mathbf{v}_n^1, \mathbf{v}_n^2, \mathbf{v}_n^3, \mathbf{v}_n^4$ such that:

$$\mathbf{v}_n = \sum_{i=1}^{4} \mathbf{v}_n^i.$$

We split the Kraft sum $\frac{5}{8}$ over the four groups in the following way

$$S(\mathbf{v}_n^1) = \frac{1}{4}, S(\mathbf{v}_n^2) = \frac{1}{8}, S(\mathbf{v}_n^3) = \frac{1}{8}, S(\mathbf{v}_n^4) = \frac{1}{8},$$

this leads to the target sum:

$$S(\mathbf{v}_n) = \sum_{i=1}^{4} S(\mathbf{v}_n^i) = \frac{5}{8}.$$

Such as we have split vector $\mathbf{v}_n$, we also split the code $\mathcal{C}(\mathbf{v}_n)$ into four different subcodes:

$$\mathcal{C}(\mathbf{v}_n) = \mathcal{C}^{00}(\mathbf{v}_n^1) \cup \mathcal{C}^{01}(\mathbf{v}_n^2) \cup \mathcal{C}^{10}(\mathbf{v}_n^3) \cup \mathcal{C}^{11}(\mathbf{v}_n^3),$$

where each code $\mathcal{C}^{ij}(\mathbf{v}_n^k)$ contains words with i as prefix and j as suffix (ex. $\mathcal{C}^{00}(\mathbf{v}_n^1)$ contains only words which start with a 0 and end with a 0). The most important part of the proof is how the four $\mathbf{v}_n^i$ vectors are structured. The idea is to arrange these vectors in an orthogonal-like way (with only one possible overlap). We start populating the subcodes from group 1 adding a certain quantity of codewords until the Kraft sum is reached ($S(\mathbf{v}_n^1) = \frac{1}{4}$), this is done for all the four subcodes. In a more concise way the coefficients k_i are arranged in this way:

$$\text{if } k_t^i \neq 0, \forall i' > i, t' < t \implies k_{t'}^{i'} = 0.$$

The initial code $\mathcal{C}(\mathbf{v}_1) = \emptyset$ is an empty code. Suppose we have constructed fix-free code $\mathcal{C} = \mathcal{C}(\mathbf{v}_{t-1})$ a step $t-1$; we have to prove that:

$$|{}^0\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^0(\mathcal{C})| \geq k_t^1,$$

$$|{}^0\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| \geq k_t^2,$$

$$|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^0(\mathcal{C})| \geq k_t^3,$$

$$|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| \geq k_t^4.$$

This is essentially the same procedure used before with the two previous proofs. We have to introduce for symmetry four partial sums $\delta_i = S(\mathbf{v}_{t-1}^i)$ for $i = 1, 2, 3, 4$. The clever split of the number of codewords in different vectors $\mathbf{v}_n$ leads to an important consequence:

$$\text{if } \delta_i = 0 \vee \delta_i < S(\mathbf{v}_n^i) \implies \delta_{i+1} = 0.$$

So we can discriminate four possible cases:

1. $\delta_1 < \frac{1}{4}$ and $\delta_2 = \delta_3 = \delta_4 = 0$ (case 1);
2. $\delta_1 = \frac{1}{4}$, $\delta_2 < \frac{1}{8}$ and $\delta_3 = \delta_4 = 0$ (case 2);
3. $\delta_1 = \frac{1}{4}$, $\delta_2 = \frac{1}{8}$, $\delta_3 < \frac{1}{8}$ and $\delta_4 = 0$ (case 3);
4. $\delta_1 = \frac{1}{4}$, $\delta_2 = \frac{1}{8}$, $\delta_3 = \frac{1}{8}$ and $\delta_4 < \frac{1}{8}$ (case 4).

In all these cases we must have the available codewords greater than the requested ones, this means:

$$k_t^i \leq 2^t \cdot (S(\mathbf{v}_n^i) - \delta_i). \tag{2.3}$$

Proof of case 3.1. Using the inequality 2.3

$$\begin{aligned} k_t^1 &\leq 2^{t-2} - \delta_1 \cdot 2^t, \qquad (2.4)\\ k_t^2 &\leq 2^{t-3},\\ k_t^3 &\leq 2^{t-3},\\ k_t^4 &\leq 2^{t-3}. \end{aligned}$$

The cardinalities of the regular sets are $|{}^0\overrightarrow{F}(\mathcal{C})| = |\overleftarrow{F}^0(\mathcal{C})| = 2^{t-2} - \delta_1 \cdot 2^{t-1}$ and $|{}^1\overrightarrow{F}(\mathcal{C})| = |\overleftarrow{F}^1(\mathcal{C})| = 2^{t-2}$. These cardinalities can be easily carried out thinking about the four different right-regular and left-regular sets. For example if we take ${}^0\overrightarrow{F}(\mathcal{C})$ set at step $t-1$ the total number of possible codewords that we can select, remembering that in the current case we are populating the code $\mathcal{C}^{00}(\mathbf{v}_t^1)$ since $\delta_1 < \frac{1}{4}$ so it is not saturated, are 2^{t-2} because two bits are fixed (the first 0 and the last 0). Hence, we show that the fix-free property is satisfied:

$$\begin{aligned} |{}^0\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^0(\mathcal{C})| &\geq 2^{t-2} - \delta_1 \cdot 2^t \geq k_t^1,\\ |{}^0\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| &\geq 2^{t-2} - \delta_1 \cdot 2^{t-1} > 2^{t-3} \geq k_t^2,\\ |{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^0(\mathcal{C})| &\geq 2^{t-2} - \delta_1 \cdot 2^{t-1} > 2^{t-3} \geq k_t^3,\\ |{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| &\geq 2^{t-2} > k_t^4. \end{aligned}$$

□

Proof of case 3.2. The procedure is the same as the one used in the previous proof.
Using the inequality 2.3

$$\begin{aligned} k_t^1 &= 0,\\ k_t^2 &\leq 2^{t-3} - \delta_2 \cdot 2^t,\\ k_t^3 &\leq 2^{t-3},\\ k_t^4 &\leq 2^{t-3}. \end{aligned}$$

The cardinalities become

$$\begin{aligned}
|{}^0\overrightarrow{F}(\mathcal{C})| &= 2^{t-2} - (\frac{1}{4} + \delta_2) \cdot 2^{t-1}, \\
|\overleftarrow{F}^0(\mathcal{C})| &= 2^{t-3}, \\
|{}^1\overrightarrow{F}(\mathcal{C})| &= 2^{t-2}, \\
|\overleftarrow{F}^1(\mathcal{C})| &= 2^{t-2} - \delta_2 \cdot 2^{t-1}.
\end{aligned}$$

In this case we have $\delta_1 = \frac{1}{4}$ and $\delta_2 < \frac{1}{8}$ so we must be careful in counting the correct number of available codewords that satisfy the fix-free constraint. For example take the set ${}^0\overrightarrow{F}(\mathcal{C})$, the code $\mathcal{C}^{00}(\mathbf{v}_t^1)$ is saturated meaning that $S(\mathbf{v}_t^1) = \frac{1}{4}$, so $\frac{1}{4}$ of the total amount of codewords are unavailable plus the amount defined by the coefficient δ_2. We can show that the fix-free property is satisfied:

$$\begin{aligned}
|{}^0\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| &\geq 2^{t-3} - \delta_2 \cdot 2^t \geq k_t^2, \\
|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^0(\mathcal{C})| &\geq 2^{t-3} \geq k_t^3, \\
|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| &\geq 2^{t-2} - \delta_2 \cdot 2^{t-1} > 2^{t-3} \geq k_t^4.
\end{aligned}$$

□

Proof of case 3.3. In a systematical way we continue the known procedure. Using the inequality 2.3

$$\begin{aligned}
k_t^1 &= 0, \\
k_t^2 &= 0, \\
k_t^3 &\leq 2^{t-3} - \delta_3 \cdot 2^t, \\
k_t^4 &\leq 2^{t-3}.
\end{aligned}$$

The cardinalities become

$$\begin{aligned}
|\overleftarrow{F}^0(\mathcal{C})| &= 2^{t-3} - \delta_3 \cdot 2^{t-1}, \\
|{}^1\overrightarrow{F}(\mathcal{C})| &= 2^{t-2} - \delta_3 \cdot 2^{t-1}, \\
|\overleftarrow{F}^1(\mathcal{C})| &= 3 \cdot 2^{t-4}.
\end{aligned}$$

We can see that $\mathcal{C}^{00}(\mathbf{v}_t^1)$ and $\mathcal{C}^{01}(\mathbf{v}_t^1)$ are full (saturated), so the set ${}^0\overrightarrow{F}(\mathcal{C})$ is no longer taken under consideration because the words starting with 0 can not be inserted in the code anymore. Thus the fix-free property is satisfied:

$$\begin{aligned}
|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^0(\mathcal{C})| &\geq 2^{t-3} - \delta_3 \cdot 2^t \geq k_t^3, \\
|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| &\geq 3 \cdot 2^{t-4} - \delta_3 \cdot 2^{t-1} > 2^{t-3} \geq k_t^4.
\end{aligned}$$

□

Proof of case 3.4. Using the inequality 2.3

$$\begin{aligned} k_t^1 &= 0, \\ k_t^2 &= 0, \\ k_t^3 &= 0, \\ k_t^4 &\leq 2^{t-3} - \delta_4 \cdot 2^t. \end{aligned}$$

The cardinalities become

$$|{}^1\overrightarrow{F}(\mathcal{C})| = |\overleftarrow{F}^1(\mathcal{C})| = 2^{t-2} - (\frac{1}{8} + \delta_4) \cdot 2^{t-1}.$$

Thus we can verify the inequality:

$$|{}^1\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^1(\mathcal{C})| \geq 2^{t-3} - \delta_4 \cdot 2^t \geq k_t^4.$$

□

Finally we complete the entire proof.

□

□

Theorem 14 (Yekhanin [8]). *Using the previous proof a new upper bound on the expected length of the code can be stated. Let $P(X = x)$ be a probability distribution for a random variable X over an alphabet $\mathcal{X}$ with $|\mathcal{X}| = n$ there exists a fix-free code $\mathcal{C}$ for which:*

$$H(X) \leq E[l(X)] \leq H(X) + 4 - \log_2 5.$$

Proof. Setting the lengths of the code $l_i = \lceil -\log_2 p(i) + 3 - \log_2 5 \rceil$ it follows:

$$\sum_{i=1}^{n} 2^{-l_i} \leq \sum_{i=1}^{n} 2^{\log_2 P(X=i) - 3 + \log_2 5} = \frac{5}{8} \sum_{i=1}^{n} P(X = i) = \frac{5}{8}.$$

By Theorem 13 the expected length can be calculated fixing $p_i = P(X = i)$:

$$\begin{aligned} E[l(X)] &= \sum_{i=1}^{n} p_i \cdot l_i \\ &< \sum_{i=1}^{n} p_i(-\log_2 p_i + 4 - \log_2 5) \\ &= H(X) + (4 - \log_2 5) \sum_{i=1}^{n} p_i \\ &= H(X) + 4 - \log_2 5. \end{aligned}$$

□

2.4 Results

In the appendix, Matlab code related to the implementation of the Yekhanin algorithm can be found. The program allowed us to carry out some tests in order to produce some fix-free codes for different input vectors. We will start with simple codes and then we move on to codes that are more difficult to find. We use Yekhanin's notation that is more user friendly than Matlab notation.

Example 1. *The initial vector* $\boldsymbol{v}_2 = \{1, 1\}$ *has* $n = 2$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_2) = \frac{3}{4}$.
The resulting code is $\mathcal{C}(\boldsymbol{v}_2) = \{0, 11\}$. *This code can be easily calculated by hand.*

Example 2. *The initial vector* $\boldsymbol{v}_9 = \{1, 0, 0, 0, 1, 0, 1, 0, 1\}$ *has* $n = 9$ *different codeword lengths witha Kraft sum* $S(\boldsymbol{v}_9) = \frac{277}{512}$.
The resulting code is $\mathcal{C}(\boldsymbol{v}_9) = \{0, 10001, 1000001, 100000001\}$.

These 2 examples are solved using the fact that the first position of the vector $\mathbf{v}_n$ is equal to 1.

Example 3. *The initial vector* $\boldsymbol{v}_6 = \{0, 2, 1, 0, 0, 3\}$ *has* $n = 6$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_6) = \frac{43}{64}$.
The resulting code is $\mathcal{C}(\boldsymbol{v}_6) = \{00, 11, 010, 011001, 011101, 011110\}$.

Example 4. *The initial vector* $\boldsymbol{v}_8 = \{0, 2, 0, 0, 0, 0, 11, 2\}$ *has* $n = 8$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_8) = \frac{19}{32}$.
The resulting code follows

$$\begin{aligned}\mathcal{C}(\boldsymbol{v}_8) = \{&00, 11, 0100001, 0100010, 0100101, 0100110,\\ &0101001, 0101010, 0101101, 0101110, 0110001,\\ &0110010, 0110101, 01000001, 01000110\}.\end{aligned}$$

These 2 examples are solved using the fact that the first position of the vector $\mathbf{v}_n$ is equal to 0 and second position is equal to 2.

Example 5. *The initial vector* $\boldsymbol{v}_8 = \{0, 1, 0, 5, 0, 1, 0, 1\}$ *has* $n = 8$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_8) = \frac{149}{256}$.
The resulting code is $\mathcal{C}(\boldsymbol{v}_8) = \{00, 0101, 0111, 1010, 1110, 1001, 100001, 10000001\}$.

Example 6. *The initial vector* $\boldsymbol{v}_8 = \{0, 0, 1, 0, 7, 13, 0, 1\}$ *has* $n = 8$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_8) = \frac{141}{256}$.
The resulting code follows

$$\begin{aligned}\mathcal{C}(\boldsymbol{v}_8) = \{&000, 00100, 00110, 01010, 01100, 00101, 00111, 01001, 010001,\\ &010111, 100010, 101110, 110010, 110100, 110110, 111010,\\ &111100, 111110, 100001, 100011, 101011, 10000001\}.\end{aligned}$$

These 2 examples satisfy the Kraft sum upper bound defined by Yekhanin (5/8) using the fact that the first position of the vector $\mathbf{v}_n$ is equal to 0 and the second position is less or equal to 1.

In all these examples we always consider at inductive step t an overlap between pairs of codewords of $t-1$ positions fixing the first bit of the first codeword and the last bit of the second codeword. If we also consider overlaps of $t-2$ positions, fixing the first 2 bits and the last 2 bits, we can increase the number of available codewords with fix-free property at step t in order to be able to create fix-free codes under the 11/16-constraint.

Let us now consider some examples of codes that do not satisfy the 5/8 upper bound for the Kraft sum but satisfy the 11/16 upper bound. What we are trying to show is that tuning the different Kraft sums of the split vectors $\mathbf{v}_n^i$ the conjecture stated by Ahlswede is always satisfied.

Example 7. *The initial vector* $\boldsymbol{v}_6 = \{0,0,3,0,3,11\}$ *has* $n = 6$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_6) = \frac{41}{64}$.
The resulting code follows

$$\begin{aligned}\mathcal{C}(\boldsymbol{v}_6) = \{&000, 010, 001, 10100, 10110, 11100,\\ &100100, 100110, 100011, 100101, 100111,\\ &101011, 101111, 110011, 110101, 110111, 111111\}.\end{aligned}$$

Example 8. *The initial vector* $\boldsymbol{v}_8 = \{0,0,0,7,0,6,11,5\}$ *has* $n = 8$ *different codewords with a Kraft sum* $S(\boldsymbol{v}_8) = \frac{163}{256}$.
The resulting code follows

$$\begin{aligned}\mathcal{C}(\boldsymbol{v}_8) = \{&0000, 0010, 0100, 0110, 0001, 0011, 1000,\\ &101010, 101100, 101110, 111010, 100101, 100111, 1001001,\\ &1001101, 1010111, 1011011, 1011111, 1101001, 1101011,\\ &1101101, 1101111, 1110111, 1111001, 10011001, 10101101\\ &11111110, 11111111\}.\end{aligned}$$

Example 9. *The initial vector* $\boldsymbol{v}_8 = \{0,0,0,0,14,10,6,1\}$ *has* $n = 8$ *different codeword lengths with a Kraft sum* $S(\boldsymbol{v}_8) = \frac{165}{256}$.
The resulting code follows

$$\begin{aligned}\mathcal{C}(\boldsymbol{v}_8) = \{&00000, 00010, 00100, 00110, 01000, 01010,\\ &01100, 01110, 00001, 00011, 00101, 00111,\\ &10000, 10010, 110100, 110110, 111000, 111010,\\ &101001, 101011, 101101, 101111, 110001, 110011,\\ &1001101, 1001111, 1010001, 1010101, 1111110,\\ &1111101, 11111111\}.\end{aligned}$$

2.5 Future work

In this chapter we have seen how fix-free codes are constructed using Yekhanin procedure when the Kraft sum is less or equal than 5/8. We also know that Ahlswede conjectured that the tightest possible upper bound is 3/4, so some of the future works about this topic are related to increase the 5/8 upper bound and asymptotically reach the conjectured one 3/4. The following step after 5/8 is 11/16; this upper-bound is supposed to be achievable allowing codewords to overlap with each other not only over $t-1$ positions (where t is the inductive step) but also over $t-2$ positions (this means that for the construction we go back of 2 inductive steps $t-1$ and $t-2$).

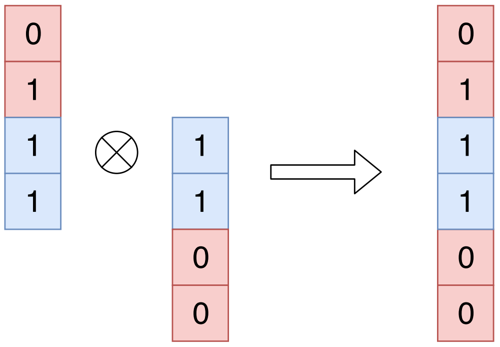

Figure 2.1: *Construction of a fix-free word fixing four bits.*

The extension of the Yekhanin proof to 11/16-constraint brings some difficulties about the exponentially increasing number of different cases that must be proved in order to demonstrate that this new upper-bound is achievable. There could be two different ways to prove the theorem with 11/16-constraint: the first one is doing it by hand, this means that all the inequalities and different cases must be carried out and proved individually; the second one is about trying to setup a computer program, with all the inequalities coded, that choose pseudo-randomly the Kraft sums associated to each disjoint subset of the original code set and verify that these inequalities are all satisfied.

Appendix

2.A Fix-free codes construction implemented in Matalb

In this section we will see a Matlab implementation of the construction process of a fix-free code based on the Yekhanin proof.
In this implementation the code is split into 4 subcodes fixing the first and the last bit of the codewords, but thanks to the generic functions which we created, it can be extended to work also with a generic number (power of 2) of subcodes. The implementation follow step-by-step all the procedure defined by Yekhanin in order to build a fix-free code starting from a vector of codeword lengths. So the input of the matlab script is a vector of lengths $\mathbf{v}_n$. This vector is defined as:

$$\mathbf{v}_n = \{k_1, k_2, ..., k_n\},$$

where n is the maximum codeword length (l_{max}) so there is a constraint on k_n which must be a positive number.
The matlab code is divided into four scripts:

1. kraftsum.m calculates the Kraft sum $S(\mathbf{v}_n)$;

2. createregular.m is the script that returns the sets left and right regulars ${}^{\alpha}\overrightarrow{F}(\mathcal{C})$ or $\overleftarrow{F}^{\beta}(\mathcal{C})$ where α, $\beta \in \{0, 1\}$. The variables α and β can be defined in different numeral systems such as $\alpha_{\mathcal{D}}$, $\beta_{\mathcal{D}} \in \{0, 1, ..., \mathcal{D} - 1\}$ where $\mathcal{D}$ defines the numeral system (classic case $\mathcal{D} = 2$), so that the construction process that split the initial code $\mathcal{C}(\mathbf{v}_n)$ in different disjoint sets (Yakhanin split the code into four codes freezing the first bit and last bit of words that belongs to a certain code) is totally tunable by the user. Changing the numeral system defined by $\mathcal{D}$ allows to construct different subcodes where the number of fixed bits at the head of the codeword and at the tail of the codeword can be easily modified.

3. mregular.m performs the operation: ${}^{\alpha}\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^{\beta}(\mathcal{C})$, so it produces a list of all the possible codewords that satisfy the fix-free property and can be added to the code $\mathcal{C}^{\alpha\beta}(\mathbf{v}_n)$;

4. checkps.m checks at the end of the construction process that the code $\mathcal{C}(\mathbf{v}_n)$ is a fix-free code;

5. fixfree.m is the core script that control and manage all the operations in order to achieve the target function which is the construction of a fix-free code starting from a vector $\mathbf{v}_n$.

2.A.1 Kraft sum

This is the first script, it's a function script that receive in input the vector $\mathbf{v}_n = (k_1, k_2, ..., k_n)$ and compute the Kraft sum as defined in previous sections:

$$S = \sum_{i=1}^{n} k_i \cdot 2^{-i} = \sum_{i=1}^{n} \mathbf{v}(i) \cdot 2^{-i}.$$

Here, a snippet of the matlab code:

```
function S = kraftsum(v)
    % initialize the sum to 0
    S = 0;
    for i=1:length(v)
        % sum up all the contributions
        S = S + v(i)*2^(-i);
    end

end
```

where variable v is the input vector, variable S is the return parameter that is the Kraft sum.

2.A.2 Check correctness

This function checks if the input code $\mathcal{C}(\mathbf{v}_n)$ is a correct fix-free code, this means that it does not contain any codewords which are prefix or suffix of other codewords. For the correct definition of fix-free codes see def. 23.
Here, a snippet of the matlab code:

```
function ok = checkps(code)
```

```
    ok = 1; % boolean set initially to 1
    for i=1:length(code)
        for j=i+1:length(code)
            a = code{i};
            b = code{j};
            if strcmp(a, b(1:length(a))) == 1
                ok = 0; % a prefix of b
                break;
            end
            if strcmp(a, ...
               b(end-length(a)+1:end)) == 1

                ok = 0; % a suffix of b
                break;
            end
        end
        if ok == 0
            break;
        end
    end

end
```

it is an exhaustive search in order to check if no codeword is prefix or suffix of another codeword. The variable *code* is represented with a cell-vector (matlab structure to handle multidimensional arrays each one with different length).

Function **strcmp** is a built-in matlab function which checks if two strings are equal and return constant value 1 if the condition is met. We exploit the fact that by construction the vector *code* contains words which are sorted by their length from the shortest to the longest.

The function returns 1 if the fix-free condition is met.

2.A.3 Create regular

This function creates the lists associated to **left regular** sets and **right regular** sets, this means in the Yekhanin case to create ${}^{0}\overrightarrow{F}(\mathcal{C})$, $\overleftarrow{F}^{0}(\mathcal{C})$, ${}^{1}\overrightarrow{F}(\mathcal{C})$ and $\overleftarrow{F}^{1}(\mathcal{C})$ sets. The code also handles multiple left and right regular sets, in the classic case we have 4 different sets because we fix the first bit and the last bit of the codewords, but if we fix more than one bit (F is the number of fixed bits, for simplicity we assume that F is an

even number) the number of different left and right regular sets increase exponentially in the fixed number of bits (2^F different sets).
We split the code in order to comment each section in a complete way.

```
function [left, right] =
createregular(code, step, opt)
    left = cell(2^opt, 1);
    right = cell(2^opt, 1);
```

In this first snippet we can see the prototype of the function *createregular* which has 3 parameters: the code represents with variabile *code*, the induction step contains in variable *step* and an option parameter associates with variable *opt*. The variable *code* represents the fix-free code at a certain induction step which can be found in the variable *step*.
The output variables *left* and *right* are 2 column cell-vectors. The dimension of these vectors is defined by the parameter *opt* (default *opt* = 1 which means 4 different sets).

```
for i=0:(2^step-1)
    num = dec2bin(i, step); % changing numeral system
    flag_l = 1; % prefix-free boolean value
    flag_r = 1; % suffix-free boolean value
    for j=1:length(code)
        if strcmp(num(1:length(code{j})), ...
        code{j}) == 1
            flag_l = 0; % num is not prefix-free
        end
        if strcmp(num(end-length(code{j})+1:end), ...
        code{j}) == 1
            flag_r = 0; % num is not suffix-free
    end
end
```

Function **dec2bin** transform a decimal number into a binary one specifying the number of bits for the representation. Variables *flag_l* and *flag_r* can assume two logic states (0 and 1) where state 0 means false and state 1 means true (classical boolean variable). The outer for loop cycles on all the possible binary representations of length *step* (2^{t-1} is the number different binary representations when t indicates the induction step). The inner for loop cycles on all the current codewords inside the code set $\mathcal{C}$. If we found a codeword which is a prefix or a suffix of a codeword inside the current code then the corresponding variable *flag_r* or *flag_l* is set to 0.

```
if flag_l == 1
    if opt == 0 % only one left regular set
```

```
            left{1} = [left{1}; num];
        else
            % creating different left regular sets
            for l=0:(2^opt-1)
                n = dec2bin(l, opt);
                if num(1:opt) == n
                    left{l+1} = [left{l+1}; num];
                end
            end
        end
    end
    if flag_r == 1
        if opt == 0 % only one right regular set
            right{1} = [right{1}; num];
        else
            % creating different right regular sets
            for l=0:(2^opt-1)
                n = dec2bin(l, opt);
                if num(end-opt+1:end) ==  n
                    right{l+1} = [right{l+1}; num];
                end
            end
        end
    end
```

If the variable *flag_l*(*flag_r*) is equal to 1 means that the current number contained in the variable *num* can be inserted into the *left*(*right*) vector. If the variable *opt* is 0 then the *left*(*right*) cell-vector is simply a string array containing all the prefix(suffix)-free codewords.
The for loop that cycles over the variable l is needed to match the correct *left* or *right* set where the selected codeword will be inserted into.

2.A.4 Mregular

This function computes the operation ${}^{\alpha_\mathcal{D}}\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^{\beta_\mathcal{D}}(\mathcal{C})$, where $\alpha_\mathcal{D}$ and $\beta_\mathcal{D}$ are integer numbers in base $\mathcal{D}$. This is useful when we increase the number of disjoint code sets (for an easier implementation this number is always a power of 2).

```
function union = mregular(left, right, opt)
    union = {}; % set of fix-free codewords
    c = 1; % number of elements
    for i=1:size(left, 1)
        l = left(i, :);
```

```
            for j=1:size(right, 1)
                r = right(j, :);
                % checking overlapping codewords
                if strcmp(l(opt+1:end), r(1:end-opt))
                    == 1
                 b = strcat(l(1:opt), l(opt+1:end), ...
                 r(end-opt+1:end));
                 % adding the codeword to the set
                 union{c} = b;
                 c = c+1;
                end
            end
        end
end
```

The parameters of this functions are *left*, *right* and *opt*. Variables *left* and *right* represent a single **right regular**(*left*) set and a single **left regular**(*right*) set. The $\otimes$ operation is done in order to build a set of codewords that satisfy the fix-free property and can be added to code $\mathcal{C}$ without any problems.
Let '01011' $\in \overrightarrow{F}(\mathcal{C})$ be a codeword that is prefix-free over the code $\mathcal{C}$ and let '10110' $\in \overleftarrow{F}(\mathcal{C})$ be another codeword that is suffix-free over the code $\mathcal{C}$, the $\otimes$ operation can be easily represented with the following figure:

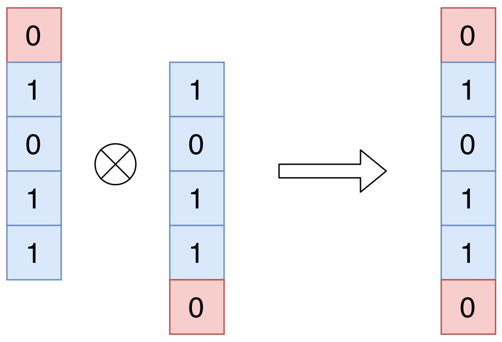

Figure 2.2: *Construction of a word that satisfy fix-free property.*

The bits of the two initial sequences with a red background represent the fixed part of the final codeword, while the bits with blue background represent the two overlapping sequence of bits; so the fixed parts plus the overlapping part will build the final codeword that has fix-free property over the code $\mathcal{C}$. If we check the correspondance between left regular set and right regular set we can build a new set that contains codewords with lengths increased by the value inside the variable *opt* (default is 1).

2.A.5 Fix-free construction

Now we see the control-logic unit of the current implementation for the construction of fix-free codes. This script recounts all the steps used by Yekhanin to proof the sufficient condition of 5/8 for the existence of a fix-free code.
The first snippet of the matlab code is about the initialization phase:

```
v = [0 1 0 0 2 3]; % example of lengths vector
len = length(v); % length of the vector
summ = kraftsum(v); % kraft sum of v
code = {}; % initialize the code to the empty set
```

vector v is the equivalent of the vector $\mathbf{v}_n$ in Yekhanin proof; so it contains k_i non negative integer values where k_i is the number of codewords we need of length i. Variable *len* contains the number of v elements, variable *summ* contains the Kraft sum calculated over the vector v and the variable *code* is initialized to the empty set.
The following snippet is about the construction of the code $\mathcal{C}$ when the vector v has fixed 1 in the first position (meaning that one codeword of length one is required) and the Kraft sum is less or equal to 3/4 (which is the upper bound conjectured by Ahlswede).

```
if v(1) == 1 && summ <= 3/4
    code = {'0'}; % initialize code to '0' codeword
    lc = 2; % index of the next codeword in code

    % loop on all the elements of v
    for i=2:len
        if v(i) > 0
            % creation of left-right regular sets
            [left, right] = createregular(code, ...
            i-1, 0);

            % new set of fix-free codes
            add = mregular(left{1}, right{1}, 1);
            for j=1:v(i)
                % adding new codewords to the code
                code{lc} = add{j};
                lc = lc+1;
            end
        end
    end
```

In a similar way as seen previously in the proof of Theorem 13, we set the initial code to a single codeword ($\mathcal{C}(\mathbf{v}_1) = \{0\}$), variable *lc* is the index for the next codeword inserted in the code. The for loop cycles on each position of vector v and call function **createregular** in order to build the left and right regular sets (containing codewords of length defined in the loop counter i). Then function **mregular** is called to obtain a vector of codewords that satisfy the fix-free property; this vector is assigned to variable *add*. The last operation is to assign the correct number of codewords needed selecting arbitrarily the codewords from the vector *add* (in a sequential manner).

The next snippet follows the second procedure for building a fix-free code when vector v has fixed 0 in the first position and 2 in the second position (meaning that we have for granted two codewords of length 2) and the Kraft sum upper bound is the maximum possible 3/4.

```
elseif v(1) == 0 && v(2) == 2 && summ <= 3/4
    % initialize the code with two codewords
    code = {'00', '11'};
    lc = 3;

    for i=3:len
        if v(i) > 0
            [left, right] = createregular(code, ...,
            i-1, 0);
            add = mregular(left{1}, right{1});
            for j=1:v(i)
                code{lc} = add{j};
                lc = lc+1;
            end
        end
    end
```

In this case the initial code $\mathcal{C} = \{00, 11\}$ has two "orthogonal" codewords (the xor bit a bit between codewords is always 1). Variable *lc* is set obviously to 3. The procedure is identical to the one described for the case stated before.

The following code snippet is related to the last case; the vector v has fixed 0 in the first position, and 0 or 1 in the second position. The Kraft sum upper bound in this case is lower and it is set to 5/8.

```
elseif v(1) == 0 && v(2) <= 1 && summ <= 5/8
    split = [];
    split(1) = 1/4; % first element set to 1/4
    split(2:4) = ones(1,3)*1/8; % others to 1/8
    % number of fixed bits
    lsplit = log2(length(split));
    % declaration of the split vectors
    ks = zeros(length(split), length(v));
    lc = 1;
```

This case is the most difficult one because we have to split the vector v into four different vectors with particular properties. Also the Kraft sum is split (natural consequence) for each of the new created vectors (in order to be able to build the fix-free code, the first upper bound for the Kraft sum is 1/4 and for the remaining three is 1/8). Vector *split* contains each Kraft's vector upper bound sum, variable *lsplit* represents the number of bits needed to map all the vectors (classical case *lsplit* = 2). Matrix *ks* contains the split vectors built from the initial vector *v*.

The following snippet is about the construction of the split vectors ($\mathbf{v}_n^i$ in Yekhanin proof). This can be done using the Matrix *ks* and starting to populate each split vector until its Kraft sum is reached.

```
for l=1:length(split)
    S = 0; % incremental sum initializes to 0
    for i=1:length(v)
        for j = 1:v(i)
            S = S + 2^(-i);

            % check the vector of Kraft sums
            if S <= split(l)
                ks(l, i) = ks(l, i)+1;
            end
            % check if the vector is full
            if S >= split(l)
                break;
            end
        end
        if S >= split(l)
            break;
        end
    end
    v = v - ks(l,:); % update the v vector
end
```

The variable S is the partial Kraft sum and for each cycle of the inner for loop it is updated until the Kraft sum of the current split vector is reached, if so, we terminate the two internal loops, then the partial sum is reset and the split vector index is incremented. At the end of all the cycles the vector v must be equal to the zero vector meaning that all the relative numbers of codewords have been inserted into the split vectors. The last part of the matlab code concerns the creation of the fix-free code computing the $\otimes$ operation for all the possible pairs made by one left regular set and one right regular set. This can be described by the following formula:

$$^{\alpha_{\mathcal{D}}}\overrightarrow{F}(\mathcal{C}) \otimes \overleftarrow{F}^{\beta_{\mathcal{D}}}(\mathcal{C}) \textit{ for all } \alpha_{\mathcal{D}}, \beta_{\mathcal{D}} \in \{0, 1, ..., \mathcal{D}\}$$

Here we can see the matlab snippet:

```
for i = 2:length(v)
    if sum(ks(:,i)) > 0
        [left, right] = createregular(code, ...,
        i-1, log2(lsplit));
    end
    for l=1:length(split)
        n = dec2base(l-1, lsplit, lsplit);
        if ks(l,i) > 0
            add = mregular(left{str2num(n(1))+1}, ...
            right{str2num(n(2))+1}, log2(lsplit));
            for j=1:ks(l,i)
                code{lc} = add{j};
                lc = lc + 1;
            end
        end
    end
end
```

variables *left* and *right* are the returned cell-vectors by the **createregular** function. In this particular case the *opt* parameter field is set to **log2**(*lsplit*)(which represent the number of fixed bits used for the construction of different left and right regular sets).
First operation needed is checking if the sum of matrix values inside *ks* that are related to *i*-split vector is greater than 0 (means that there are requested codewords). Then for each of the possible pair of left regular set and right regular set the function **mregular** is called in order to get back the possible codewords that can be added to the code $\mathcal{C}$ without violating the fix-free property. The *opt* parameter field, also here, is set to **log2**(*lsplit*) this defines the number of prefix(suffix) bits that are fixed so the overlap between two codewords (one in the left regular set and the other in the right regular set) will have shorter length.

The last snippet is the following:

```
if checkps(code) == 1
    code #shows on screen the code
else
    disp('Lengths␣do␣not␣respect␣constraints!');
end
```

This is only an ultimate check of the correct construction of the fix-free code; the function that fulfills the check is **checkps**.

In Figure 2.1 we have two codewords '0111' $\in \overrightarrow{F}(\mathcal{C})$ and '1100' $\in \overleftarrow{F}(\mathcal{C})$ which means that are prefix(suffix)-free over the code $\mathcal{C}$. The red boxes represent the fixed bits while the blue boxes represent the matching bits (whose bits that allow the overlap between two codewords). The codeword that is generated is '011100' which is for sure a codeword that is prefix and suffix free over the code $\mathcal{C}$.
So, reducing the overlap between codewords leads to a greater number of different code sets (in the case related to Figure 2.1 leads to 16 different sets). In Yekhanin proof we have only 4 code sets (only one fixed bit at the beginning and one fixed bit at the ending of the code):

$$^{0}\mathcal{C}^{0}(\mathbf{v}_n^1),\ ^{0}\mathcal{C}^{1}(\mathbf{v}_n^2),\ ^{1}\mathcal{C}^{0}(\mathbf{v}_n^3),\ ^{1}\mathcal{C}^{1}(\mathbf{v}_n^4),$$

allowing two fixed bits at the beginning and at the end of the code the number of code sets becomes 16:

$$^{00}\mathcal{C}^{00}(\mathbf{v}_n^1),\ ^{00}\mathcal{C}^{01}(\mathbf{v}_n^2),\ ^{00}\mathcal{C}^{10}(\mathbf{v}_n^3),\ ^{00}\mathcal{C}^{11}(\mathbf{v}_n^4),$$

$$^{01}\mathcal{C}^{00}(\mathbf{v}_n^5),\ ^{01}\mathcal{C}^{01}(\mathbf{v}_n^6),\ ^{01}\mathcal{C}^{10}(\mathbf{v}_n^7),\ ^{01}\mathcal{C}^{11}(\mathbf{v}_n^8),$$

$$^{10}\mathcal{C}^{00}(\mathbf{v}_n^9),\ ^{10}\mathcal{C}^{01}(\mathbf{v}_n^{10}),\ ^{10}\mathcal{C}^{10}(\mathbf{v}_n^{11}),\ ^{10}\mathcal{C}^{11}(\mathbf{v}_n^{12}),$$

$$^{11}\mathcal{C}^{00}(\mathbf{v}_n^{13}),\ ^{11}\mathcal{C}^{01}(\mathbf{v}_n^{14}),\ ^{11}\mathcal{C}^{10}(\mathbf{v}_n^{15}),\ ^{11}\mathcal{C}^{11}(\mathbf{v}_n^{16}).$$

So each code set contains codewords that are prefix-free and suffix-free over the code $\mathcal{C}$. The goal here is how to define the upper bounds on each Kraft sum related to the corresponding vector $\mathbf{v}_n^i$ with $i = 1, 2, .., 16$. This means finding the correct Kraft sum $S(\mathbf{v}_n^i)$ for each $\mathbf{v}_n^i$ where:

$$\sum_{i=1}^{16} S(\mathbf{v}_n^i) = \frac{11}{16}.$$

This leads to an exponential increase of the possible cases in order to prove that also with an upper bound of 11/16 can be always possible to produce a fix-free code over a vector $\mathbf{v}_n$. Proving all the cases by hand is hard (but there can be some strategies to simplify the procedure). The next step that can be done in this direction is to automatically check all the inequalities with a computer program and try to guess (deduce) the correct upper-bounds to assign at each Kraft sum related to its vector $\mathbf{v}_n^i$.

Chapter 3

Entropy of Graphs

In this chapter we study a different concept of "*constrained*" sources which concerns the lossless coding of ambigous sources. These sources are described by an undirected graph in which the edges distinguish the source symbols.
We talk about **graph complexity** and **graph entropy**. In order to better understand the importance and the limitations of the entropy approach we must first carachterize the complexity of graphs.

3.1 Taxonomy

Two different approaches to measure the complexity of graphs have been developed (Mowshowitz [14]). The first can be said *deterministic* and the second *probabilistic*; in this chapter we main focus on the *probabilistic* approach.
The deterministic category is composed by *encoding*, *substructure count* and *generative* approaches. Kolmogorov complexity represents the main enconding approach, substructure count includes measures which count the number of specified internal structures (like the number of induced subgraphs), generative approach represents the measures which are based on the operations needed to construct the graph. The Kolmogorov metrics uses as complexity measure the length of the word needed to encode an object (i.e. the number of symbols taken from the code alphabet). A way to describe graphs are with its adjacency matrix, so the Kolmogorov complexity can be viewed as the length of the word representing each cell of the adjacency matrix. An undirected graph with n vertices can be completely described by a binary sequence of length $\binom{n}{2}$ where each bit in the sequence states if a particular pair of vertices is adjacent. Hence the Kolmogorov complexity associated to an encoding scheme of the ad-

jacency matrix of a graph is at most $\binom{n}{2}$ (every addition information will reduce the length of the adjacency list).
The probabilistic category includes entropy measures which are based on probability distribution associated to graph elements. There are *intrinsic* and *extrinsic* measures. Intrinsic measures use internal features of a graph to determine a probability distribution over its elements (not an arbitrary distribution). Extrinsic measures use an arbitrary probability distribution to calculate the entropy. Shannon entropy is the most used for intrinsic measures but it is not suitable for extrinsic measures for which we see the Korner entropy of graphs.

3.2 Korner entropy

Let us introduce the concept of distinguishability of letter sequences which can be mathematically modeled by an undirected graph (see Definition 34).
We consider a discrete, memoryless and stationary information source $\mathbf{X} = (X_1, X_2, ...)$ where X_i are random variables defined over an alphabet $\mathcal{X}$, and we assume that the symbols emitted by $\mathbf{X}$ are not all distinguishable. Two sequences of source letters $(x_1, x_2, ..., x_n) \in \mathcal{X}^n$ and $(y_1, y_2, ..., y_n) \in \mathcal{X}^n$ are distinguishable if and only if $\exists i : x_i \neq y_i$. The main goal is to assign different codewords to groups of sequences which are not distinguishable with each other apart from a set of sequences with probability less than a coefficient ϵ $(0 < \epsilon < 1)$. We call this number of different codewords needed $N(n, \epsilon)$. Korner [11] proved that:

$$N(n, \epsilon) = 2^{nH(\mathbf{X})+o(n)}, \tag{3.1}$$

where $H(\mathbf{X})$ is a quantity independent of ϵ that depends only on the information source $\mathbf{X}$. This quantity is called the (Korner) entropy of the graph because we can immediately see the correlation with the Shannon entropy used in the Asymptotic equipartition property (AEP) where the number of elements in the typical set is 2^{nH} (in case of i.i.d. random variables).
The proof given by Korner [11] is done considering memoryless and stationary information sources of i.i.d. random variables over a finite set of symbols.

Definition 34. *An undirected graph G is defined by its vertices (symbols of alphabet $\mathcal{X}$) and its edges:*

$$G = (\mathcal{X}, E(\mathcal{X})), E(\mathcal{X}) \subseteq \binom{\mathcal{X}}{2}$$

where $E(\mathcal{X})$ is the set of edges of the graph G.

In our context edges represent distinguishability, that is, two symbols $a, b \in \mathcal{X}$ are connected in $\mathcal{G}$ if they are distinguishable.

Definition 35. *A probabilistic graphs $\mathcal{G}$ is defined by a graph G with a probability distribution $P(X = x)$ that associates a probability to each symbol $x \in \mathcal{X}$:*

$$\mathcal{G} = (G, P) = (\mathcal{X}, E(\mathcal{X}), P(X = x)).$$

We can extend these definitions for sequences of symbols $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathcal{X}^n$ as follows.

Definition 36. *The n-th power of a probabilistic graph is defined as:*

$$\mathcal{G}^n = (G^n, P^n) = (\mathcal{X}^n, E(\mathcal{X}^n), P^n(X_1^n)),$$

where $E(\mathcal{X}^n)$ is the set of edges of the power graph for which sequence $\boldsymbol{x} \in \mathcal{X}^n$ and sequence $\boldsymbol{y} \in \mathcal{X}^n$ are connected if and only if $\exists k, j \in \{1, 2, ..., n\} : (x_k, y_j) \in E(\mathcal{X})$.

Due to the source properties, the decoder, in order to obtain the initial sequence of symbols, must assign to sequences of symbols that are connected in graph $\mathcal{G}^n$ different codewords (without significant errors). Small probabilities of error are taken into account with variable ϵ imposing that sequences, with probability less than ϵ, are mapped into a single codeword. This structure leads to a known problem in graph theory that is the Graph coloring.

Definition 37. ***Graph coloring*** *or vertex coloring means that given a graph G, two connected vertices in G are assigned different colors. Graph coloring can be seen also as the partition of the set of vertices into edge-independent sets which means that elements in an edge-independent set are not connected.*

Definition 38. *The* ***chromatic number*** *of a graph G is the smallest number of colors needed to assign to vertices of G such that no adjacent vertices share the same color (the colors can be seen as the codewords needed); $\chi(G)$ is the notation for the chromatic number.*

The minimum number of codewords required to represent sequences of length n with probability greater or equal than $1 - \epsilon$ is defined as

$$\chi(n, \epsilon) = N(n, \epsilon) - 1 = \min_{\mathcal{S} \subset \mathcal{G}^n, P^n(\mathcal{S}) \geq 1-\epsilon} \chi(S),$$

where $N(n, \epsilon)$ is the minimum number of codewords required (one less than the chromatic number in order to map all the sequences with probability less than ϵ to only one codeword), $\mathcal{S}$ is a subset of the n-th cartesian product of the probabilistic graph $\mathcal{G}$. The probability $P^n(\mathcal{S}) = \sum_{\mathbf{x} \in \mathcal{S}} P^n(X_1^n = \mathbf{x})$.

Definition 39. *Korner [11] defines the entropy of a probabilistic graph $\mathcal{G}$ as:*

$$H(\mathcal{G}) = \lim_{n\to\infty} \frac{1}{n} \log_2 \chi(n, \epsilon) \tag{3.2}$$

where he proves that the limit is independent of the parameter ϵ.

The chromatic number $\chi(\mathcal{G})$ of a probabilistic graph $\mathcal{G}$ can be expressed in terms of minimum number of **kernels** needed to cover the entire graph, whose definition follows.

Definition 40. *A subset $K \subseteq \mathcal{X}$ where $\mathcal{X}$ is the alphabet of source symbols of a probabilistic graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P(X = x))$ is called **kernel** (maximal independent set) if:*

$$\forall x, y \in K \text{ with } x \neq y : (x, y) \notin E(\mathcal{X});$$

$$\forall x \in (\mathcal{X} \setminus K), \exists y \in K : (x, y) \in E(\mathcal{X}).$$

It can be easily proved that the minimum number of covering kernels is equivalent to the chromatic number. It is assumed that the probabilistic graphs under consideration have no loops (means that each vertex is not connected to itself) but we have to consider also the case when we have two sequences of symbols $\mathbf{x}$ and $\mathbf{y} \in \mathcal{X}^n$ which have $x_k = y_k$ for some $k < n$. In order to consider this situation Korner has defined a new graph operation.

Definition 41. *Given a probabilistic graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ we can define a new probabilistic graph $\mathcal{G}_\pi^n = (\mathcal{X}^n, E_\pi(\mathcal{X}^n), P^n(X_1^n))$ over the set of vertices $\mathcal{X}^n$ (sequences of length n). The sequences $\boldsymbol{x} = (x_1, x_2, ..., x_n) \in \mathcal{X}^n$ and $(y_1, y_2, ..., y_n) \in \mathcal{X}^n$ are connected in the new graph (meaning $(\boldsymbol{x}, \boldsymbol{y}) \in E_\pi$) if and only if:*

$$\forall k = 1, ..., n, (x_k, y_k) \in E(\mathcal{X}) \;\vee\; x_k = y_k. \tag{3.3}$$

So the number of codewords needed to color the graph can be expressed as

$$\chi_\pi(n, \epsilon) = \min_{\mathcal{S} \subset \mathcal{G}_\pi^n, P^n(\mathcal{S}) \geq 1-\epsilon} \chi(\mathcal{S}), \tag{3.4}$$

where $P^n(\mathcal{S}) = \sum_{\boldsymbol{x} \in \mathcal{S}} P^n(X_1^n = \boldsymbol{x})$.

If $x \equiv y$, where $(x, y) \in E(\mathcal{X})$ or $x = y$, is an equivalent[1] relation on the vertex set $\mathcal{X}$ then the graph is the union of pairwise disjoints complete subgraphs.

[1]Binary relation that is reflexive, symmetric and transitive

The main result proved by Korner [11] is that:

1. $\chi_\pi(n, \epsilon) \geq 2^{nH(X|e(X)) - K\sqrt{n}}$;
2. $\chi_\pi(n, \epsilon) \leq 2^{nH(X|e(X)) + K\sqrt{n}}$;

where X is a random variable over the alphabet $\mathcal{X}$ with discrete probability distribution $p(x)$ and $e(x)$ is the set which contains the union of all the elements that are equivalent ($\equiv$) to a particular symbol $x \in \mathcal{X}$. $K \in \mathbb{R}$ is a constant and it is independent of n and $p(x)$ but depends on the cardinality of the alphabet $|\mathcal{X}|$ and on the error coefficient ϵ. We can observe that $e : x \to e(x)$ is a surjective function, so:

$$p(x, e(x)) = p(x) = P(e(x)) \cdot p(x|e(x)),$$

knowing this, the formula for the conditional entropy $H(X|e(X))$ can be derived:

$$H(X|e(X)) = \sum_{x \in \mathcal{X}} p(x) \cdot \log_2 \frac{P(e(x))}{p(x)}.$$

Korner proved that the two inequalities seen before hold, and then asymptotically as $n \to \infty$:

$$\begin{aligned} H(\mathcal{G}) &= \lim_{n\to\infty} \frac{1}{n} \log_2 \chi(n, \epsilon) \\ &= \lim_{n\to\infty} \frac{1}{n} \log_2 \chi_\pi(n, \epsilon) \\ &= H(X|e(X)). \end{aligned}$$

We can easily see the similarity with the Entropy rate that is defined for stationary/ergodic sources (how many bits on average do we need to encode each symbol of the alphabet).
The concept of typical sequence can be carried out from the work of Korner (based on Wolfowitz's method).

Definition 42. *A sequence $\boldsymbol{x} \in \mathcal{X}^n$ of length n is called typical, over a probability distribution $P(X)$ where X is a random variable that takes values from the alphabet $\mathcal{X}$, if and only if for all $y \in \mathcal{X}$:*

$$|N(y|\boldsymbol{x}) - nP(X = y)| \leq K\sqrt{nP(X = y)},$$

where $N(y|\boldsymbol{x})$ is the number of times symbol y appears in the sequence $\boldsymbol{x}$.

The principal theorem that Korner [11] has stated follows.

Theorem 15. *Given the set of typical sequences of length n called $\mathcal{T}^n = \{\boldsymbol{x} \in \mathcal{X}^n\}$ over a probability distribution $P(X_1^n = x_1^n) = p(x)^n$ then $\forall \epsilon : 0 < \epsilon < 1$ there exists a constant K for which $P(\mathcal{T}^n) \geq 1 - \epsilon$. If $\boldsymbol{x}$ is a typical sequence then:*

$$2^{-nH(X)-C\sqrt{n}} \leq P(X_1^n = \boldsymbol{x}) \leq 2^{-nH(X)+C\sqrt{n}},$$

where $H(X)$ is the entropy of the random variable X defined over an alphabet $\mathcal{X}$ and over the probability mass function $P(X = x)$. The constant C is independent of n and $P(X = x)$, and only depends on the cardinality of the alphabet $|\mathcal{X}|$ and on the error coefficient ϵ.

3.3 Simonyi's proof of Korner Theorem

Korner [11] proved the direct part and the strong converse part of Theorem 15. It is interesting to see the proof of this theorem given by Simonyi using rate distortion theory obtaining a weak converse proof. Simonyi in his survey of graph entropy [12] gives different equivalent definitions of graph entropy.
The first definition that Simonyi gives is about the *vertex packing polytope* of a (probabilistic) graph $\mathcal{G}$ known as VP($\mathcal{G}$) which is the convex hull[2] of the indicator vectors related to the independent sets of $\mathcal{G}$. Let us clarify giving the definition of indicator vector:

Definition 43. *Let A be a subset of B where $B = \{b_1, b_2, ..., b_n\}$. The indicator vector of A is called $x_A = (x_1, x_2, ..., x_n)$ where $x_i = 1$ if $b_i \in A$ and $x_i = 0$ if $b_i \notin A$.*

The vertices of the *vertex packing polytope* are the indicator vectors of the independent sets of the graph $\mathcal{G}$. All the convex combinations of the polytope vertices generate the smallest convex set that contains all the indicator vectors.
After some notions we can see the first definition of graph entropy given by Simonyi [12]:

Theorem 16. *Given a probabilistic graph $\mathcal{G} = \{\mathcal{X}, E(\mathcal{X}), P\}$, the entropy of $\mathcal{G}$ can be equivalently defined as:*

$$H(\mathcal{G}) = \min_{\mathbf{c} \in VP(\mathcal{G}), \mathbf{c} > 0} \sum_{i=1}^{n} p_i \cdot \log_2(\frac{1}{c_i}). \tag{3.5}$$

[2]The convex hull of a set of points $\mathcal{P}$ in n dimensions is the intersection of all convex sets $\cap_{i=1}^{\infty} \mathcal{H}_i$ with $\mathcal{P} \subseteq \mathcal{H}_i$.

The equation 3.5 is a minimization over a function that is convex which implies that the minimum is finite and reachable ([12]).
Now after seeing the new definition about graph entropy, we want to see a new formulation of the theorem proved by Korner in terms of rate distortion theory. We consider a probabilistic graph $\mathcal{G}$ with edges that define distinguishability of symbols. As we have seen before the graph can be extended in order to considering sequences of length n by computing the n-th cartesian product of the graph: $\mathcal{G}^n$. We can assign the same codeword to sequences that are not connected in $\mathcal{G}$. An error probability coefficient ϵ is used to map all the sequences that have an associated probability which is less than ϵ to a single codeword. We always consider memoryless and stationary information sources.
The performance of the encoding is measured in terms of rate, so if we have sequences of length n the rate is $\frac{\log_2 |\mathcal{C}|}{n}$ where $|\mathcal{C}|$ is the cardinality of the set of codewords (smaller the rate better the performance). We want to find the minimum rate $R(\mathcal{G}^n, \epsilon)$ when $\epsilon \to 0$ and $n \to \infty$ between all the possible rates of encoding scheme that respect the constraint of distinguishability.
Let us see another definition of graph entropy that was derived by Korner [11] for which he proved that the Definition 3.2 is equivalent.

Theorem 17. *Given a probabilistic graph* $\mathcal{G} = \{\mathcal{X}, E(\mathcal{X}), P\}$*:*

$$H(\mathcal{G}) = \min_{(X,Y): X \in Y, Y \in S(\mathcal{G})} I(X;Y), \tag{3.6}$$

where $I(X;Y)$ *is the mutual information of the two discrete random variables* X, Y *(*$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$*). Random variable* X *takes values from the alphabet* $\mathcal{X}$ *(vertices of* $\mathcal{G}$*), random variable* Y *takes values from the maximal independent sets of* $\mathcal{G}$ *called* $S(\mathcal{G})$*. The minimization is over the pairs of random variables* (X, Y) *such that* $Pr\{X \in Y\} = 1$*.*

From Theorem 17 some considerations can be carried out:

1. If our probabilistic graph $\mathcal{G}$ has no edges, remembering the formula given in 3.6, $I(X;Y) = H(X) + H(Y) - H(X,Y) = H(Y) - H(Y|X)$, there is one possible configuration for Y which is the one that contains all the vertices of $\mathcal{G}$ implying that $I(X;Y) = 0$ and consequently $H(\mathcal{G}) = 0$.

2. If graph $\mathcal{G}$ is a complete graph (all the vertices are connected to each other) with n vertices, the possible configurations for Y are the sets containing only one symbol $x \in \mathcal{X}$, so the minimum mutual information becomes $I(X;Y) = H(X) - H(X|Y) = H(X) - 0 =$

$H(X)$. It emerges that with complete graphs the Korner entropy converges to Shannon entropy.

The proof reported in [12] brings out the equivalence between formulation 3.5 and formulation 3.6.

Proof. First of all it must be proved that:

$$\min_{(X,Y):X\in Y,Y\in S(\mathcal{G})} I(X;Y) \geq \min_{\mathbf{c}\in \mathrm{VP}(\mathcal{G}),\mathbf{c}>0} \sum_{i=1}^{n} p_i \cdot \log_2(\frac{1}{c_i}). \tag{3.7}$$

If the minimum for the formulation with the mutual information (eq. 3.6) is achieved by a pair (X_m, Y_m), then we can rewrite the mutual information as:

$$\begin{aligned} I(X_m;Y_m) &= \sum_{i=1}^{n} \sum_{i\in J,J\in S(\mathcal{G})} P(X_m = i, Y_m = J) \log_2 \frac{P(X_m = i, Y_m = J)}{P(X_m = i)\cdot P(Y_m = J)} \\ &= -\sum_{i=1}^{n} p_i \sum_{i\in J,J\in S(\mathcal{G})} P(Y_m = J|X_m = i) \log_2 \frac{P(Y_m = J)}{P(Y_m = J|X_m = i)}. \end{aligned}$$

From the concavity of the log function we have

$$-\sum_{i=1}^{n} p_i \sum_{i\in J,J\in S(\mathcal{G})} P(Y_m = J|X_m = i) \log_2 \frac{P(Y_m = J)}{P(Y_m = J|X_m = i)}$$

$$\geq$$

$$-\sum_{i=1}^{n} p_i \log_2 \sum_{i\in J,J\in S(\mathcal{G})} P(Y_m = J).$$

Since the vector $\mathbf{c} \in \mathrm{VP}(\mathcal{G})$ with components $c_i = \sum_{i\in J,J\in S(\mathcal{G})} P(Y_m = J)$ is in $\mathrm{VP}(\mathcal{G})$, inequality 3.7 is proved.

Now we have to prove the reverse inequality ([12]):

$$\min_{(X,Y):X\in Y,Y\in S(\mathcal{G})} I(X;Y) \leq \min_{\mathbf{c}\in \mathrm{VP}(\mathcal{G}),\mathbf{c}>0} \sum_{i=1}^{n} p_i \cdot \log_2(\frac{1}{c_i}). \tag{3.8}$$

If vector $\mathbf{c} \in VP(\mathcal{G})$ minimizes the formulation on the right hand-side of the inequality, each $c_i = \sum_{i\in J,J\in S(\mathcal{G})} P(Y'_s = J)$ where $\mathbf{c}$ can be seen as a probability distribution on $S(\mathcal{G})$. The conditional probability can be defined as $P(Y_s = J|X_s = i) = \begin{cases} P(Y'_s = J)/c_i & i \in J \\ 0 & otherwise \end{cases}$.

The probability distribution of Y_s can be derived:

$$P(Y_s = J) = \sum_{i=1}^{n} P(X_s = i) \cdot P(Y_s = J | X_s = i).$$

So, by definition

$$\min_{(X,Y):X\in Y, Y\in S(\mathcal{G})} I(X;Y)$$

$$\leq$$

$$-\sum_{i=1}^{n} p_i \sum_{i\in J, J\in S(\mathcal{G})} P(Y_s = J|X_s = i) \log_2 \frac{P(Y_s = J)}{P(Y_s = J|X_s = i)}$$

thanks to the concavity of the log function

$$\sum_{J\in S(\mathcal{G})} P(Y_s = J) \log_2 \frac{P(Y_s' = J)}{P(Y_s = J)} \leq 0,$$

thus we can derive the following inequality

$$\sum_{i=1}^{n} p_i \sum_{i\in J, J\in S(\mathcal{G})} P(Y_s = J|X_s = i) \log_2 \frac{P(Y_s = J)}{P(Y_s = J|X_s = i)}$$

$$\leq$$

$$-\sum_{i=1}^{n} p_i \sum_{i\in J, J\in S(\mathcal{G})} P(Y_s = J|X_s = i) \log_2 \frac{P(Y_s' = J)}{P(Y_s = J|X_s = i)}.$$

Hence the final inequality can be stated

$$\min_{(X,Y):X\in Y, Y\in S(\mathcal{G})} I(X;Y) \leq \sum_{i=1}^{n} p_i \cdot \log_2(\frac{1}{c_i}).$$

□

3.4 Graph entropy properties

In this section we see some properties of graph entropy such as: *monotonicity*, *sub-additivity*, *additivity by substitution* and *disjoint union.*

Theorem 18 (Sub-additivity). *Let $\mathcal{G}_1 = (\mathcal{X}, E_1(\mathcal{X}), P)$ and $\mathcal{G}_2 = (\mathcal{X}, E_2(\mathcal{X}), P)$ be two probabilistic graphs defined on the same vertex set $\mathcal{X}$ and with a fixed probability mass function P, then the union of the two graphs $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2 = (\mathcal{X}, E_1(\mathcal{X}) \cup E_2(\mathcal{X}), P)$ implies:*

$$H(\mathcal{G}) \leq H(\mathcal{G}_1) + H(\mathcal{G}_2).$$

Proof. Let $P(X, Y_1)$ and $P(X, Y_2)$ be two joint distributions that minimize the mutual information for $\mathcal{G}_1$ and $\mathcal{G}_2$, variables Y_1, Y_2 are independent conditioning on the fact that they have to contain X:

$$\begin{aligned}
H(\mathcal{G}) &\leq^a I(X; (Y_1 \cap Y_2)) \\
&\leq^b I(X; Y_1, Y_2) \\
&=^c H(Y_1, Y_2) - H(Y_1, Y_2|X) \\
&=^d H(Y_1, Y_2) - H(Y_1|X) - H(Y_2|X) \\
&\leq H(\mathcal{G}_1) + H(\mathcal{G}_2).
\end{aligned}$$

Inequality (a) holds because $Y_1 \cap Y_2$ contains X and it is an independent set in $\mathcal{G}$, inequality (b) holds because $I(X; f(Y_1, Y_2)) \leq I(X; Y_1, Y_2)$ where f is an arbitrary function and the last two inequalities (c) and (d) are carried out using the chain rule of the entropy and knowing that Y_1, Y_2 are independent conditioned on X. □

Theorem 19 (Monotonicity). *Given two probabilistic graphs $\mathcal{G}_1 = (\mathcal{X}, E_1(\mathcal{X}), P)$ and $\mathcal{G}_2 = (\mathcal{X}, E_2(\mathcal{X}), P)$ defined on the same vertex set $\mathcal{X}$ and with the same probability mass function P, under the condition $E_1(\mathcal{X}) \subset E_2(\mathcal{X})$ can be stated:*

$$H(\mathcal{G}_1) \leq H(\mathcal{G}_2)$$

Proof. Let (X, Y) be the pair that minimize the mutual information for graph $\mathcal{G}_2$, then Y is for sure an independent set also for $\mathcal{G}_1$, so $H(\mathcal{G}_1) \leq H(\mathcal{G}_2)$.

□

Definition 44 (Substitution of a vertex [12]). *Let $\mathcal{F} = (\mathcal{Y}, E_1(\mathcal{Y}), Q)$ and $\mathcal{G} = (\mathcal{X}, E_2(\mathcal{X}), P)$ be two probabilistic graphs where $\mathcal{X} \cap \mathcal{Y} = \emptyset$. If $v \in \mathcal{Y}$ we can define a new operation which produces a new graph $\mathcal{G}_{v \leftarrow \mathcal{F}} = (\mathcal{N} = (\mathcal{X} \setminus \{v\}) \cup \mathcal{Y}, E(\mathcal{N}), P_{v \leftarrow \mathcal{Q}})$; this operation substitutes vertex v with the entire graph $\mathcal{F}$ connecting each vertex of $\mathcal{F}$ to adjacent vertices of v in $\mathcal{G}$. The probability $P_{v \leftarrow \mathcal{Q}}$ is defined as:*

$$P_{v \leftarrow \mathcal{Q}}(X = x) = \begin{cases} P(X = x) & x \in \mathcal{X} \setminus \{v\} \\ P(X = v) \cdot Q(X = x) & x \in \mathcal{Y} \end{cases}.$$

Knowing the previous definition, the theorem on additivity by substitution can be stated.

Theorem 20 (Additivity by substitution [12]). *Given two probabilistic graphs $\mathcal{F} = (\mathcal{Y}, E_1(\mathcal{Y}), Q)$ and $\mathcal{G} = (\mathcal{X}, E_2(\mathcal{X}), P)$ where $\mathcal{X} \cap \mathcal{Y} = \emptyset$ and let v be a vertex of $\mathcal{X}$ then:*

$$H(\mathcal{G}_{v \leftarrow \mathcal{F}}) = H(\mathcal{G}) + P(X = v) \cdot H(\mathcal{F}).$$

Corollary 20.1 (K-partite graph). *Given a probabilistic graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ which is a complete K-partite graph with $m_1, m_2, ..., m_k$ the size of the maximal independent sets of $\mathcal{G}$. Given Q the probability distribution over $S(\mathcal{G})$ (set of the maximal independent sets) and let $Q(Y = J) = \sum_{v \in J} P(X = v)$ for all $J \in S(\mathcal{G})$ (Y is a random variable taking values over $S(\mathcal{G})$). The entropy of $\mathcal{G}$ follows:*

$$H(\mathcal{G}) = H(\mathcal{F}),$$

where $\mathcal{F}$ is a complete graph over k-symbols and over the probability distribution Q.

Corollary 20.2 (Disjoint union). *Given a probabilistic graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ which is composed by n disjoint graphs $\mathcal{G}_i = (\mathcal{X}_i, E(\mathcal{X}_i), P_i)$ (disjoint vertex sets), then the entropy of the graph is defined as:*

$$H(\mathcal{G}) = \sum_{i=1}^{n} P(V(\mathcal{G}_i)) \cdot H(\mathcal{G}_i),$$

where $\mathcal{X} = \cup_{i=1}^{n} \mathcal{X}_i$, $|\mathcal{X}| = \sum_{i=1}^{n} |\mathcal{X}_i|$, $P(V(\mathcal{G}_i)) = \sum_{x \in \mathcal{X}_i} P(X = x)$ and $P_i(X = x) = \frac{P(X=x)}{P(V(\mathcal{G}_i))}$ with $x \in \mathcal{X}_i$.

Theorems 18 and 19 can be also proved using the definition of graph entropy in terms of *vertex packing polytope* (Simonyi [12]).

Theorem 21 (Monotonicity, Simonyi [12]). *Given the statement of Theorem 19, it follows that*

$$H(\mathcal{G}_1) \leq H(\mathcal{G}_2).$$

Proof. If we have the relation defined above then working on *vertex packing polytopes* we have $\mathrm{VP}(\mathcal{G}_2) \subseteq \mathrm{VP}(\mathcal{G}_1)$. This means that if we have a vector $\mathbf{a}$ which minimizes the function given in eq. 3.5 for graph $\mathcal{G}_1$ this vector is also contained in $\mathrm{VP}(\mathcal{G}_2)$. $\square$

Theorem 22 (Sub-additivity, Simonyi [12]). *Given the statement of Theorem 18, it follows that*

$$H(\mathcal{G}) \leq H(\mathcal{G}_1) + H(\mathcal{G}_2).$$

Proof. Let call $\mathbf{a} \in \mathrm{VP}(\mathcal{G}_1)$ and $\mathbf{b} \in \mathrm{VP}(\mathcal{G}_2)$ the two vectors that define the entropies, knowing that the intersection of an independent set of $\mathcal{G}_1$ and an independent set of $\mathcal{G}_2$ is always an independent set of $\mathcal{G}$ meaning that the vector $(a_1b_1, a_2b_2, ..., a_nb_n) \in VP(\mathcal{G})$, then we can state:

$$\begin{aligned} H(\mathcal{G}_1) + H(\mathcal{G}_2) &= \sum_{i=1}^{n} p_i \log_2(\frac{1}{a_i}) + \sum_{i=1}^{n} p_i \log_2(\frac{1}{b_i}) \\ &= \sum_{i=1}^{n} p_i \log_2(\frac{1}{a_i \cdot b_i}) \\ &\geq H(\mathcal{G}). \end{aligned}$$

The graph entropy is said to be additive in the weak sense if there exists a probability distribution P (where $p_i > 0$ for all i) that implies:

$$H(\mathcal{G}) = H(\mathcal{G}_1) + H(\mathcal{G}_2).$$

Instead, if the graph satisfies the equality for all probability distributions on the vertex set: the graph is said to be additive in a strong sense.

□

3.5 Some examples of graph entropy

We give some examples of graphs where the entropy is easily computable.

Example 1. *In the first example we consider a complete graph in which every pair of distinct vertices is connected. The usual notation to identify complete graph is K_n where n defines the number of vertices. A probability distribution $p(x) = (1/4, 1/4, 1/8, 3/8)$ is given, assigning a probability to each symbol that lies on the graph. We call the following probabilistic graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ where $\mathcal{X} = \{A, B, C, D\}$ and $E(\mathcal{X}) = \{(x_i, x_j) \in \mathcal{X}^2 : i \neq j\}$.*

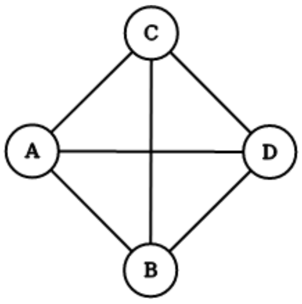

Figure 3.1: *K_4 complete graph with 4 vertices.*

In this case as we said before each symbol (A, B, C, D) has to be represented with different codewords cause there is an edge between every *pair of symbols. This means for this particular case that the minimum number of codewords needed to represent this graph is 4. The graph entropy defined by Korner becomes the Shannon entropy, so, it can be easily computed as:*

$$H(\mathcal{G}) = H(P) = \frac{1}{2}\log_2 4 + \frac{1}{8}\log_2 8 + \frac{3}{8}\log_2 \frac{8}{3} \approx 1.906.$$

Example 2. *A trivial example is when the graph has no edges. Let $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ be the probabilistic graph associated to the one in Figure 3.2 with $\mathcal{X} = \{A, B, C, D\}$, $E(\mathcal{X}) = \emptyset$ and $p(x)$ is an arbitrary distribution function.*

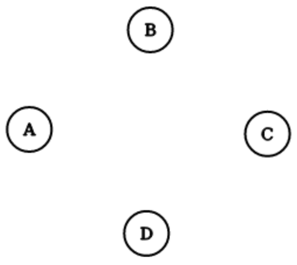

Figure 3.2: *Disconnected graph.*

The entropy of this graph can be easily calculated with the Definition 3.5 because the vector $\boldsymbol{a}$ *which minimizes that function is equal to* $(1, 1, 1, 1)$ *where all the vertices constitute an independent set. The entropy of the graph follows:*

$$H(\mathcal{G}) = \sum_{i=1}^{4} P(X = i) \cdot \log_2 1 = 0.$$

Example 3. *Let us consider an example of a graph constructed by substitution.*

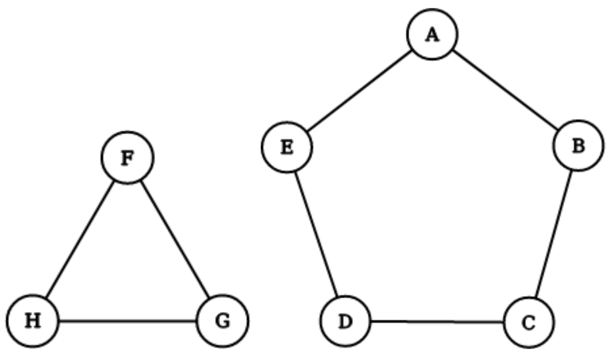

Figure 3.3: $\mathcal{F}$ *graph (left one) and* $\mathcal{G}$ *graph (right one).*

So, $\mathcal{F}$ is a probabilistic graph over the vertex set $\mathcal{Y} = \{F, G, H\}$ and over a probability distribution $Q = (1/6, 1/6, 2/3)$ while $\mathcal{G}$ is a probabilistic graph over $\mathcal{X} = \{A, B, C, D, E\}$ and over a probability distribution $P = (1/5, 1/5, 1/5, 1/5, 1/5)$. We can substitute the vertex (symbol) A of $\mathcal{G}$ with the graph $\mathcal{F}$ in the following way:

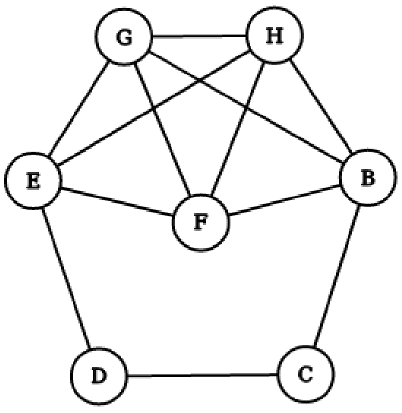

Figure 3.4: *Graph $\mathcal{G}_{A\leftarrow\mathcal{F}} = (\mathcal{Y} \cup \mathcal{X} \setminus \{A\}, E(\cdot), P_{A\leftarrow Q})$.*

Figure 3.4 represents the graph constructed by substitution of vertex A with the graph $\mathcal{F}$ by connecting all the vertices of $\mathcal{F}$ to adjacent vertices of symbol A in $\mathcal{G}$.
We know from the previous section that the entropy of the new graph can be derived from the entropies of the two building graphs $\mathcal{F}$ and $\mathcal{G}$.

$$H(\mathcal{G}_{A\leftarrow\mathcal{F}}) = H(\mathcal{G}) + P(X = A) \cdot H(\mathcal{F}).$$

Graph $\mathcal{G}$ is a 5-cycle graph and from [18] we know that $H(\mathcal{G}) = \log_2 \frac{5}{2}$ when the probability mass function related to source symbols is uniform. Graph $\mathcal{F}$ is a 3-complete graph and its entropy is

$$H(\mathcal{F}) = H(Q) \approx 1.26.$$

Hence the entropy of $\mathcal{G}_{A\leftarrow\mathcal{F}}$ follows

$$H(\mathcal{G}_{A\leftarrow\mathcal{F}}) \approx 1.32 + \frac{1}{5}1.26 \approx 1.57.$$

Example 4. *Given a probabilistic graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ with $\mathcal{X} = \{A, B, C, D, E, F, G, H, I\}$ and $P = (1/9, 1/9, ..., 1/9) = \boldsymbol{u}$ equals the uniform distribution. The edge-set $E(\mathcal{X})$ can be seen in the next figure:*

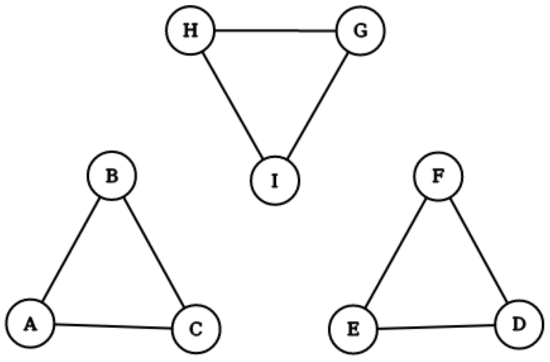

Figure 3.5: *Graph $\mathcal{G}$ with disjoint vertex sets.*

The graph $\mathcal{G}$ is composed by three connected-components (subgraphs), we call these subgraphs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$. As stated in Theorem 20.2 the entropy of the graph showed in Figure 3.5 can be computed:

$$\begin{aligned} H(\mathcal{X}) &= \frac{1}{3}H(\mathcal{G}_1) + \frac{1}{3}H(\mathcal{G}_2) + \frac{1}{3}H(\mathcal{G}_3) \\ &= 3 \cdot \frac{1}{3} \cdot H(1/3, 1/3, 1/3) \\ &= \log_2 3. \end{aligned}$$

Example 5. *Given a graph $\mathcal{G} = (\mathcal{X}, E(\mathcal{X}), P)$ that is a complete 3-partite graph over an alphabet $\mathcal{X} = \{A, B, C, D, E, F\}$. Let P be the uniform distribution over the symbols, from previous theorems we can easily compute its entropy.*

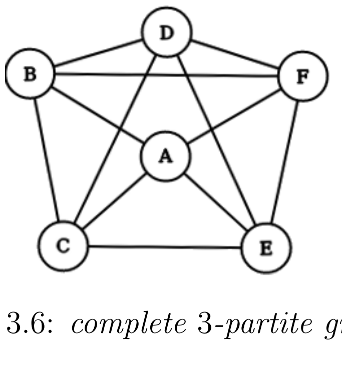

Figure 3.6: *complete 3-partite graph $\mathcal{G}$.*

In the graph in Figure 3.6 we can identify three maximal independent sets, so, $S(\mathcal{G}) = \{\{A, D\}, \{B, E\}, \{C, F\}\}$. The probability distribution

Q is the uniform distribution over $S(\mathcal{G})$ (due to the uniformity of P). The entropy of $\mathcal{G}$ follows:

$$H(\mathcal{G}) = H(Q) = H(1/3, 1/3, 1/3) = \log_2 3.$$

Example 6. *In the last example we compute the entropy of the Turán graph $T(5,4)$*[3]*. Consider a probabilistic graph $\mathcal{G} = (T(5,4), P)$ constructed by the Turán graph in Figure 3.7 and defined over a probability distribution $P = (1/2, 1/4, 1/8, 1/16, 1/16)$.*

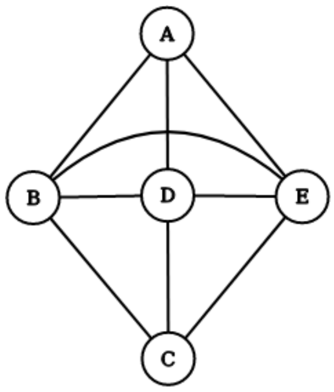

Figure 3.7: *Turán graph: complete 4-partite graph.*

The set of maximal independent sets is $S(\mathcal{G}) = \{\{A, C\}, \{B\}, \{D\}, \{E\}\}$. The probability distribution Q can be derived: $Q(Y = y) = (5/8, 1/4, 1/16, 1/16)$, where Y is a random variable that takes values over $S(\mathcal{G})$. Knowing that the Turán graph is a complete 4-partite graph then the entropy of these types of graphs is known and can be computed as:

$$H(\mathcal{G}) = H(\mathcal{F}),$$

where $\mathcal{F}$ is a probabilistic complete graph defined over a 4-symbols alphabet and over the probability distribution Q. This leads to:

$$H(\mathcal{G}) = H(Q) = \frac{5}{8}\log_2\frac{8}{5} + \frac{1}{2} + \frac{1}{2} = \frac{5}{8}\log_2\frac{8}{5} + 1 \approx 1.42.$$

[3]Complete 4-partite graph in which the maximal independent sets size are $2, 1, 1, 1$.

3.6 Future work

Korner [11] proved that Definition 3.2 is equivalent to Definition 3.6 with the assumption that the sequences of symbols are produced by memoryless sources. It can be useful to study possible extensions of Korner's theorem when the sources are Markovian, stationary and also ergodic. This means that the sequences which lie on the vertices of the probabilistic graph under consideration are described by a transition probability matrix that defines the change of state in terms of probabilities. The AEP (Asymptotic equipartition property) for ergodic sources can be used for the definition of typical sequences (procedure defined by Wolfowitz [15]).

Graph coloring and the enumeration of all maximal independent sets of a graph are NP-hard problems so polynomial time algorithms that solve this kind of problems are not known (P vs NP). The implementation of an algorithm that computes the entropy of a given graph can be written using exponential time algorithms that list all the maximal independent sets and doing an exhaustive search on all the pairs of alphabet symbols and maximal independent sets. During this coupling we can compute the mutual information for each pair in order to find the minimum one. For each alphabet symbol we have to select only the maximal independent sets that contain the symbol.
Bron–Kerbosch (BK) algorithm can be used to list all the maximal independent sets in an undirected graph, it is based on a recursive backtracking algorithm. Knowing that given a graph with n-vertex there are at most $3^{n/3}$ maximal independent sets [17], the worst case of the BK algorithm has computational complexity $O(3^{n/3})$. An implementation of graph entropy can be useful to bring out some peculiarities between different graphs.

Definitly a future work is to continue the study of graph entropy extending it to hypergraphs and convex corners (Simonyi [12]). The sub-additivity property of graph entropy is a fundamental property for perfect hashing problem which could be interesting to study.

Conclusions

We have dealt with different topics in this thesis trying to give a general overview on unique decodability of constrained sequences, fix-free codes and graph entropy.

Dalai [3] shows that it is necessary to consider a modified Kraft inequality when we are dealing with constrained sources (with memory) in order to give a general definition of unique decodability. This leads to a new necessary condition for the code to be uniquely decodable but unfortunately it is not a sufficient condition. A modified Sardinas-Patterson test [3] has been used to check if a code is uniquely decodable in general sense. This test was implemented in Matlab to carry out some results and to make clear the whole process. We gave a proof of equivalence between the Moore and Mealy representation of the same Markovian source. The implementation of this equivalence has been proposed in Matlab code, trying to clarify all the procedure steps. The Sardinas-Patterson might be useful to get some feedback information about a non-uniquely decodable code in order to find an optimal code for a given constrained source.

The work of Ahlswede [7] and Yekhanin [8] allowed us to produce a Matlab implementation of the construction process of a fix-free code given a set of integer codeword lengths when the Kraft sum is upper bounded. Some useful tests have been carried out to better understand the procedure defined by Yekhanin and in order to construct codes which do not respect the 5/8-constraint of the Kraft sum but they respect the 11/16-constraint. This can be useful to outline an automatized proof for the construction of fix-free codes under the 11/16-constraint.

Korner [11] provides a definition of graph entropy in terms of distinguishability of source symbols, this led to a formulation of the problem in terms of mutual information between source symbols and maximal independent sets. Korner proved the equivalence between the chromatic number formulation with the one about the mutual information. This allowed us to calculate by hand the entropy of simple graphs like complete-graphs and k-partite graphs. The main properties of graph entropy have been treated [12] in order to see the graph entropy behaviour.

References

[1] T. M. Cover and J.A. Thomas. *Elements of Information Theory. John Wiley, New York, 1990.*

[2] R. G. Gallager. *Information Theory and Reliable Communication. Wiley, New York, 1968.*

[3] M. Dalai and R. Leonardi. *On Unique Decodability, McMillan's Theorem and the Expected Length of Codes, Technical Report R.T., 2008.*

[4] M. Dalai and R. Leonardi *On Unique Decodability, IEEE Transactions on Information Theory, 2008.*

[5] M.Dalai. *New Techniques for Signal Representation and Coding, Ph.d thesis, 2006.*

[6] B. McMillan. *Two inequalities implied by unique decipherability. IEEE Trans. Inform.Theory, 1956.*

[7] R. Ahlswede, B. Balkenhol, L. Kharchatrian. *Some Properties of Fix-Free Codes. InProc. First INTAS International Seminaron Coding Theory and Combinatorics, Thazkhadzor, Armenia, 1996.*

[8] S. Yekhanin *Improved upper bound for the redundancy of fix-free codes, IEEE Transactions on Information Theory, 2004.*

[9] A. Sardinas and G.W. Patterson. *A necessary and sufficient condition for the unique decomposition of coded messages. In IRE Convention Record, 1953.*

[10] R.B. Ash. *Information Theory. Interscience, New York, 1965.*

[11] J. Korner. *Coding of an information source having ambiguous alphabet and the entropy of graphs, 1971.*

[12] G. Simonyi. *Graph entropy - A survey, in Combinatorial Optimization, DIMACS Series in Discrete Mathematics and Computer Science, 1991.*

[13] A.D. Wyner *An Upper Bound on the Entropy Series, 1972.*

[14] Abbe Mowshowitz and Matthias Dehmer *Entropy and the Complexity of Graphs Revisited, 2012.*

[15] J. Wolfowitz *The Coding Theorems of Information Theory. Springer, Berlin 1964.*

[16] C. E. Shannon *Coding Theorems for a Discrete Source with Fidelty Criterion, New York 1960.*

[17] J.W. Moon and L. Moser *On cliques in graphs, Israel J. Math., 1965*

[18] P. Koulgi, E. Tuncel, S. Regunathan, and K. Rose. *On zero-error source coding with decoder side information, IEEE Transactions on Information Theory, January, 2003.*