Communication over Channels with Symbol Synchronization Errors

by

Hugues Mercier

B.Sc., Université Laval, 1997
M.Sc., Université de Montréal, 2003

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University of British Columbia

March 2008

Abstract

Synchronization is a problem of fundamental importance for a wide range of practical communication systems including reading media, multi-user optical channels, synchronous digital communication systems, packet-switched communication networks, distributed computing systems, etc. In this thesis I study various aspects of communication over channels with symbol synchronization errors.

Symbol synchronization errors are harder to model than erasures or substitution errors caused by additive noise because they introduce uncertainties in timing. Consequently, the capacity of channels subjected to synchronization errors is a very challenging problem, even when considering the simplest channels for which only deletion errors occur. I improve on the best existing lower and upper bounds for the capacity of the deletion channel using convex and stochastic optimization techniques. I also show that simply finding closed-form expressions for the number of subsequences when deleting symbols from a string is computationally prohibitive.

Constructing efficient synchronization error-correcting codes is also a challenging task. The main result of the thesis is the design of a new family of codes able to correct several types of synchronization errors. The codes use trellis and modified versions of the Viterbi decoding algorithm, and therefore have very low encoding and decoding complexities. They also have high data rates and work for reasonably noisy channels, which makes them one of the first synchronization-correcting codes that have any chance of being used in practical systems.

In the last part of the thesis, I show that a synchronization approach can solve the opportunistic spectrum access problem in cognitive radio, where cognitive users want to communicate in presence of legacy users to whom the bandwidth has been licensed. I also consider the amount of communication required to solve a large class of distributed problems where synchronization errors can occur. More precisely, I study how allowing the parties to solve the problems incorrectly with small probability can reduce the total amount of communication or the number of messages that need to be exchanged.

Table of Contents

List of Tables

List of Figures

Acknowledgement

First, I would like to thank my advisor and mentor Vijay K. Bhargava. From the first day I joined his research group, his guidance, professional counsel, financial assistance, and unfaltering support have been paramount to my success and defy any attempt to be properly acknowledged. I especially express my gratitude for his patience, understanding, and compassion after the difficult birth of my second daughter.

I would also like to thank Brian Marcus for organizing coding seminars and for encouraging me to discuss research problems with his students. Special thanks to Lutz Lampe and Robert Schober from my supervisory committee for their comments and suggestions during my departmental examination, which greatly improved the presentation of my dissertation, and to William A. Aiello, Panos Nasiopoulos, and Vahid Tarokh for their presence on my examining committee.

I would like to acknowledge the financial support of the Fonds québecois de la recherche sur la nature et les technologies, without which completing my graduate studies would simply not have been possible. Many thanks to the University of British Columbia for offering me a Graduate Fellowship for the fourth year of my doctoral program.

Working in Vijay's research group, I had the privilege to be surrounded by many bright graduate students and researchers from all over the world. From discussing scientific problems to learning about their respective cultures and interests, I would like to thank them for making my stay at UBC so enjoyable. They include Gaurav Bansal, Zeljko Blazek, Daniela Djonin, Dejan Djonin, Serkan Dost, Olivier Gervais, Ziaul Hasan, Jahangir Hossain, Praveen Kaligineedi, Ashok Karmokar, Majid Khabbazian, Kin-Kwong Leung, Erez Louidor, Zhiwei Mao, Chris Nicola, Piplu Paul, Umesh Phuyal, Anjana Punchihewage, Mamunur Rashid, Poramate Tarasak, and Chandika Wavegedara. Special thanks to Majid for using his sharp mind on some research questions of mine.

Above all, I would like to thank my wife and daughters for their unconditional love and for making me a better man.

À Isabelle, Charlotte et Chloé

Chapter 1

Introduction

> "The synchronization problem is not a mere engineering detail, but a fundamental communications problem as basic as detection itself."
> −Solomon W. Golomb, 1961 [39].

Early communication systems designers were preoccupied with symbol and word synchronization even more than they were preoccupied with additive noise. Samuel Morse, when designing his famous eponymous code in the 1830s, has allowed three time units of silence for a letter space and six time units of silence for a word space, a very high price to pay to reduce the synchronization errors and facilitate the decoding of messages sent over telegraph lines.

Today, synchronization problems remain an integral part of technological systems operating in environments affected by uncertainties in timing, or time noise. These include hard drives, semi-conductors, integrated circuits, and synchronous digital communication networks. In reading media, for example hard disks and digital audio tapes, physical disturbances (vibrations, tape stretch, ...) may cause the position of the head relative to the media to digress from its expected location, causing timing errors [9, 87]. In modern semiconductor devices like integrated circuits, timing errors can be caused by the varying sampling rates of the system devices [25]. Time noise caused by inaccurate clocking devices can affect most of the synchronous digital communication systems [37], and synchronization errors can be caused by the propagation medium, like crosstalk on multi-user optical channels [91] and Doppler shifts for underwater and satellite communications [94].

Time noise introduces insertions and deletions of symbols for the recipient. As a result, systems corrupted by timing errors do not know their exact position when reading or decoding data. Moreover, as the performance of modern communication systems im-

proves, increasingly stringent synchronization constraints are required. Synchronization techniques are expensive. Some, like sending a pilot tone or periodic synchronization pulses, require additional power. Others, like adding a pseudo-noise sequence in the transmitted signal that can be replicated by the receiver, require increased bandwidth, and error-correcting codes robust against timing errors increase the overall latency and complexity of the systems, reduce their throughput, and are very difficult to construct.

Synchronization models can also be used for a large number of problems ranging from mobile computing to quantum cryptography. Deletion channels with large alphabets can model packet-switched communication networks like the Internet, where packets are either lost or dropped due to buffer overflow at finite-buffer queues [22, 23, 95]. In this specific scenario deletion errors are transformed into erasures: the packets are numbered so they can be retransmitted when they are deleted. Deletion errors can also be used to model traitor tracing, where attackers remove parts of the embedded fingerprint in a digital object in order to create untraceable pirate copies [88]. In cognitive radio systems, a discrepancy between primary user detection by a cognitive transmitter and a cognitive receiver can be viewed as a loss of synchronization [79]. Insertions and deletions can also occur for a large class of distributed problems involving reconciliation of correlated data such as reconciliation of nucleotide sequences in DNA molecules [30], remote file storage [15], synchronization of mobile data [1], and quantum key distribution [10].

Although great progress has been made in the last few years, we are still far from a unified theory of synchronization. Synchronization errors are poorly modeled by the usual channels such as the binary symmetric channel or the additive white Gaussian noise channel, and new techniques need to be designed.

1.1 Thesis Outline and Contributions

My thesis considers various problems related to communication over channels with symbol synchronization errors. The main body of the thesis is divided into three parts and comes after the description of a very general channel with synchronization errors in Chapter 2. The first part focuses on the understanding of deletion channels, which are the simplest nontrivial channels with synchronization errors. Improved bounds for the capacity of the discrete binary deletion channel are presented in Chapter 3, and closed-form expressions and an algorithm for calculating the number of subsequences when deleting symbols from a string in Chapter 4. In the second part of the thesis, I study binary error-correcting codes capable of correcting synchronization errors. Block codes are presented

in Chapter 5, and a new class of trellis codes is described in Chapter 6. In the third part of the thesis, I study two problems that can be modeled using a synchronization approach: opportunistic spectrum access in cognitive radio systems in Chapter 7, and worst-case randomized interactive communication in Chapter 8. Finally, in Chapter 9, conclusions and suggestions for future work are presented. A more detailed outline follows.

1.1.1 Synchronization Channel Models

In Chapter 2, I describe the mother communication channel considered throughout the thesis. The channel can be broken down into simpler channels closely related to various types of synchronization errors like symbol deletions, symbol insertions, duplications, and combinations of more than one of them with or without substitution errors.

1.1.2 Capacity of the Discrete Binary Deletion Channel [70]

Quantifying the reduction in capacity due the presence of synchronization errors is a very difficult problem. There is no known simple analytical closed-form for the capacity of channels admitting synchronization errors, and no numerical approximation either. In Chapter 3, I consider the deletion channel, where each transmitted bit is deleted with probability p_d, and present two computational methods to estimate its capacity. The first approach bounds the mutual information between the input and the output of the channel. It converges from above as the size of the input blocks approaches infinity and leads to the first nontrivial upper bound for the capacity. The second method uses stochastic optimization to maximize the information rate of the channel when the input source is modeled by a Markov process. It converges from below as the order of the optimal Markov chain increases, and as a result I derive an approximate lower bound improving on the existing capacity lower bounds.

1.1.3 Subsequences, Sequences, and Supersequences [71]

Having been unsuccessful in my attempt to improve significantly on the capacity bounds for channels admitting synchronization errors, I decided to gain more insight on the difficulties of these channels by studying the varying size of the spheres associated to synchronization codewords. In Chapter 4, I consider the problem of finding the number of subsequences when deleting d symbols from a string. I present a framework to find closed-form expressions for the number of subsequences, and use it to prove the first

explicit formulae for $d \in \{3, 4, 5\}$ and the first formulae for nonbinary alphabets. Since in general the formulae cannot be calculated in polynomial time for large values of d, I also present the first efficient algorithm to compute the number of subsequences for any string, number of deletions, and alphabet size. A closely related problem is to compute the multiplicity of a subsequence in a string. I provide a recursive definition and an efficient algorithm for this problem.

1.1.4 Synchronization Error-Correcting Block Codes [69]

In Chapter 5, I present a simple technique to increase the detection and correction capabilities of any block code robust against synchronization errors, and use it to extend a well-known family of single-synchronization error-correcting codes for multiple insertion and deletion errors. The extended codes, although far from optimal, outperform several known block codes for synchronization errors.

1.1.5 Nonlinear Synchronization-Correcting Trellis Codes [68]

Chapter 6 presents a comprehensive investigation of nonlinear trellis codes for synchronization errors. I show that the Viterbi decoding algorithm can be modified to correct most types of synchronization errors. It can be used for insertion, deletion, and duplication errors, for instance, and as well for more complex scenarios where insertion and deletion errors can occur together. In all cases, I prove that the algorithm is able to find the closest codeword from the received sequence. The computational complexity of the algorithms is comparable to the complexity of the original Viterbi algorithm for insertion, deletion, and duplication errors, but larger for more general synchronization errors.

I show that for trellis codes whose underlying graphs have a fixed number of states, there is a fundamental tradeoff between their capacity to recover synchronization and the number of synchronization errors they can correct. A corollary of this result is that the structure of convolutional codes is not appropriate to correct synchronization errors, which explains why none of the previous attempts to use them in this context has been very successful.

I present a class of rectangular graphs well suited to correct most types of synchronization errors when used as encoders. The graphs in conjunction with the adapted versions of the Viterbi decoding algorithm offer several advantages over most coding techniques proposed to protect communication systems against time noise. Encoding information is not difficult and the decoding complexity is reasonable. It is also possible to cor-

rect bursts of synchronization errors and to take advantage of soft-information from the channels.

1.1.6 Cognitive Radio and Synchronization Errors [79]

Cognitive radio is an innovative technology proposed to increase spectrum usage by allowing dynamic allocation of the unused spectrum in changing environments. Cognitive users monitor the spectrum and are allowed to use it as long as it does not interfere with the primary users to whom it has been licensed. In Chapter 7, I describe how some well established tools from the fields of error-correcting codes and information theory can be applied to the opportunistic spectrum access problem in cognitive radio. More precisely, I describe simple channel models, compute their capacity, and demonstrate that low-density parity-check codes and synchronization error-correcting codes can be used to solve this problem.

1.1.7 Interactive Communication and Synchronization Errors [73, 72]

Synchronization errors also appear in a wide range of distributed problems. Consider the following scenario: a user modifies a large file on his laptop by inserting, deleting, and substituting characters, and wants to update the centralized version of the file located on a remote server. If few changes are made, then the files are highly correlated. Although the simplest way to reconcile the files is to copy the entire file from the laptop to the server, more clever algorithms can significantly reduce the number of bits that need to be transmitted, especially if the user and the server are allowed to interact. Thus, an interesting problem is to study the minimum amount of communication required to reconcile the two versions of the files, with or without interaction, depending on the "distance" between them. This is a special case of "interactive communication".

In the interactive communication model, two distant parties, $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$, possess respective private but correlated inputs x and y, and $P_{\mathcal{Y}}$ wants to learn x from $P_{\mathcal{X}}$ while minimizing the communication required for the worst possible input pair (x, y). In chapter 8, I analyze four nonzero-error models in this correlated data setting, i.e., models for which $P_{\mathcal{Y}}$ is allowed to learn x incorrectly with small probability. In the private coin randomized model, both players are allowed to toss coins, and P_Y must learn x with high probability for every input pair. The public coin randomized model is

similar to this model, but instead of private coins, both players have access to a common source of randomness. The private coin randomized amortized model is also similar to the first model, but the players are allowed to solve several independent instances of the same problem simultaneously instead of sequentially. Finally, the distributional model is deterministic, but P_Y is allowed to answer incorrectly for a small fraction of the inputs weighted by their probability distribution.

I show that the public coin randomized, private coin randomized amortized, and distributional models are equivalent and can be more efficient than the original worst-case deterministic model. Moreover, when the players are not allowed to interact, the difference between the best deterministic and public coin randomized protocols can be arbitrarily large. I prove that one round of communication is nearly optimal for the private coin randomized model. I also show that the deterministic model and all the nonzero-error models are equivalent for a large class of symmetric problems arising from several practical applications, although nonzero-error and randomization allow efficient one-way protocols.

Chapter 2

Synchronization Channel Models

The *mother binary synchronization channel* described in Figure 2.1 includes all the types of errors encountered in this thesis. Although synchronization between the sender and the recipient is not guaranteed, it is a memoryless channel in the sense that the noise behaves independently on each input bit.

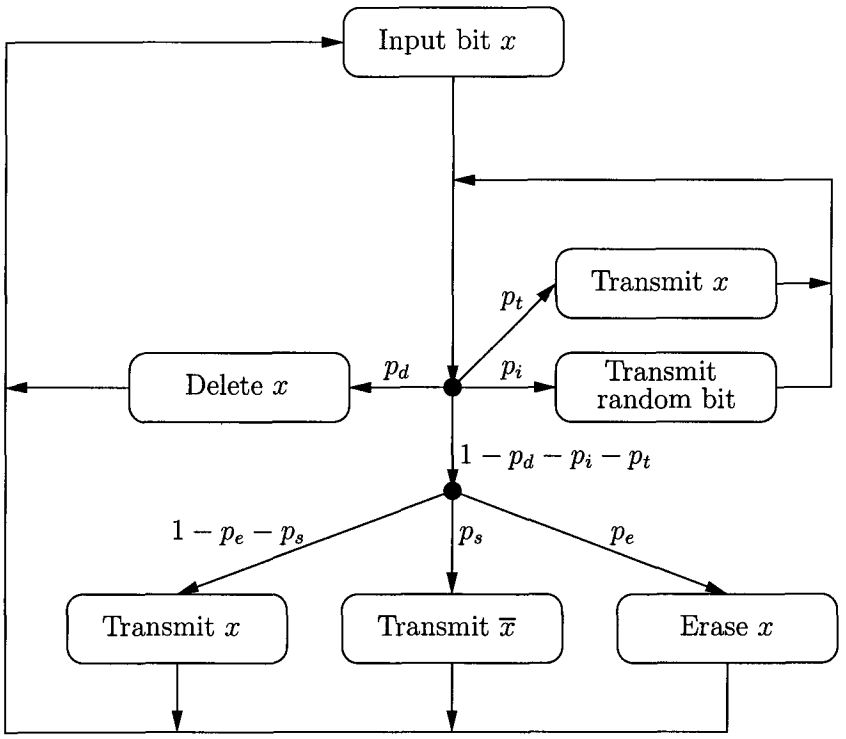

Figure 2.1: Flow chart for the mother binary synchronization channel.

There are three types of synchronization errors: *deletions*, *insertions*, and *duplications*. There are also two types of errors having no detrimental effect on synchronization: *substitutions* and *erasures*. The difference between deletions and erasures is that deletions leave no trace of their existence. For instance, deleting the middle bits from 0011 results in the output 01, whereas if the bits are erased by the channel the output is 0??1. A duplication transmits a copy of the input bit and is therefore a special case of insertion error. It is assumed that an arbitrarily large number of bits can be inserted between any two input bits.

When an input bit x is transmitted over this channel, it suffers from a synchronization error with probability $p_d + p_i + p_t$. The bit is deleted with probability p_d, a random bit is inserted before x with probability p_i, and a copy of x is inserted before x with probability p_t. Once the bit x is transmitted, it suffers from a substitution error with probability p_s and is erased with probability p_e.

All the channels used in this thesis, including the binary symmetric and binary erasure channels, can be derived from the mother synchronization channel. The *binary symmetric channel* transmits a bit correctly with probability $1 - p_s$ and transmits its complement with probability p_s, i.e., $p_d = p_t = p_i = p_e = 0$. In the binary *deletion channel*, each input bit is deleted with probability p_d and transmitted correctly with probability $1 - p_d$. The *erasure channel* has $p_e \neq 0$ and $p_d = p_t = p_i = p_s = 0$. The *duplication channel* and the *insertion channel* only cause duplication and insertion errors, respectively. The number of insertions or duplications between two input bits is geometrically distributed, although other distributions could be used. For instance, Gallager [37] considered a channel where a bit can only be replaced by two random bits, and Mitzenmacher [75] a channel where each bit is duplicated at most once; both channels have the advantage of being easier to analyze than their counterparts with more general distributions. In the literature, the *synchronization channel* and the *insertion-deletion channel* are both used to denote the channel where insertions and deletions are the only errors allowed.

The combination of deletion and duplication errors arises naturally for all the practical problems where timing errors can be caused by the varying rates of the sampling devices: faster sampling at the receiver causes duplication errors, whereas slower sampling causes deletions. In Figure 2.2, the transmitted string 1101110010110 is sampled at incorrect times by the receiver and received with two duplication errors and two deletion errors as 1100111100110.

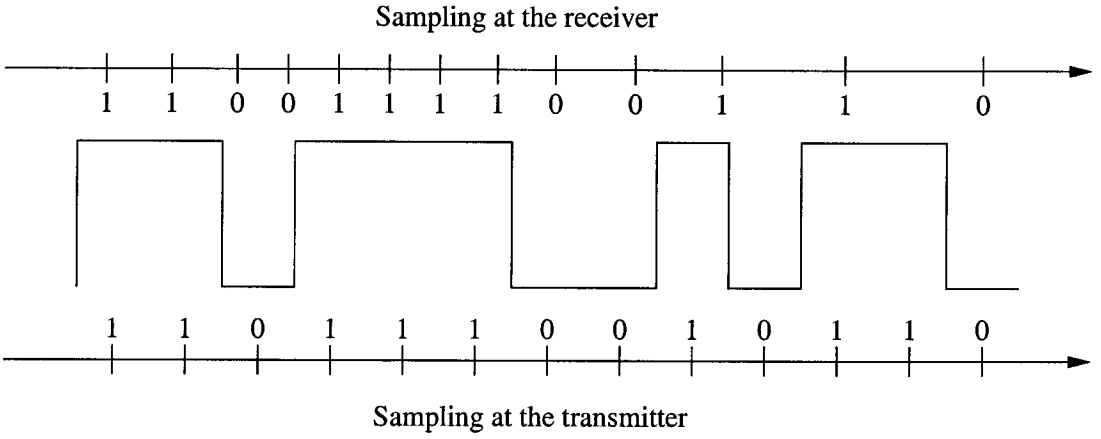

Figure 2.2: Varying sampling rate between the transmitter and receiver.

Part I

Deletion Channel Behavior

Chapter 3

Improved Numerical Bounds for the Capacity of the Binary Deletion Channel

3.1 Introduction

The *binary deletion channel* is a channel for which each input bit is deleted independently with probability p_d and transmitted correctly with probability $1 - p_d$. For example, the codeword 1011100 could be received as 1100. Although it is easier to analyze than its counterparts with insertions and deletions, the deletion channel has memory and is harder to understand than, for instance, the erasure channel. As a result, there is no known simple closed-form expression for its capacity, and no numerical approximation either.

Forty years ago, Dobrushin [24] proved the equivalent of the channel coding theorem and its weak converse for channels admitting synchronization errors. Vvedenskaya and Dobrushin [108] used this theoretical framework and tried to derive numerical lower bounds for the capacity of the binary deletion channel, but were limited by the processing power of the computers available at the time. Deletion channels have recently received renewed interest after Diggavi and Grossglauser [22, 23] used them to prove capacity lower bounds for packet-switched communication networks, where packets are either lost or dropped due to buffer overflow at finite-buffer queues. They also proved that for large alphabets, the capacity of the deletion channel is equal to the capacity of the erasure channel. Their framework was extended by Kavcic and Motwani [52] and Drinea and Mitzenmacher [29, 28], who computed improved lower bounds. Drinea and Mitzenmacher

[27] also derived a lower bound of $\frac{1-p_d}{9}$, which is the only nontrivial bound for the capacity of the binary deletion channel obtained without the use of simulations.

The capacity of the deletion channel with deletion probability p_d is upper bounded by the capacity of the erasure channel with erasure probability p_d. This bound is almost tight with large input alphabets, but quite weak in the binary case. There is no known other upper bound, either algebraic or obtained by simulation. If nothing else, upper bounds from Ullman [104] and Dolgopolov [26] were also used, but both use assumptions that cannot exactly be applied in the context of the binary deletion channel. For large values of p_d, codes with rates exceeding these last two upper bounds were recently constructed [28].

In this chapter, I present two numerical algorithms that asymptotically converge to the capacity of the binary deletion channel. The first algorithm uses convex programming on the probability of the input sequences; it converges when the length of the sequences approaches infinity. The second algorithm uses stochastic optimization on ergodic processes modeling the input source, and converges when the order of the optimal Markov chain approaches infinity. The fact that the two algorithms respectively converge to the capacity of the deletion channel from above and from below allows to improve on the existing bounds. I present the first nontrivial upper bound for the capacity of the binary deletion channel, as well as an approximate lower bound.

3.2 Preliminaries

It is assumed that the readers are familiar with the basic concepts of entropy, conditional entropy, mutual information, and channel capacity; if not there are several comprehensive presentations of these subjects like the one of Cover and Thomas [17]. All the logarithms are in base 2 unless otherwise specified, and $0 \log 0 \triangleq 0$. Let $\mathcal{X} = \mathcal{Y} = \{0, 1\}$. $\mathcal{X}^n = \mathcal{Y}^n$ is the set of all binary strings of length n, $\mathcal{X}^\infty$ the set of all binary strings of infinite length, and $\mathcal{Y}^* = \{y_1 y_2 \dots y_k | k \geq 0 \text{ and each } y_i \in \mathcal{Y}\}$ the set of all binary strings including the *empty string* ϵ[1]. The random variable X^n having possible outcomes $\mathcal{X}^n$ and probability distribution $\mathcal{P}_{X^n}$ is also defined. Finally, let $\overline{I^n} \triangleq \max_{\mathcal{P}_{X^n}} \frac{I(X^n;Y^*)}{n}$.

[1] $*$ is the *Kleene star*.

3.3 Upper Bound

Sending a single bit over a deletion channel is illustrated in Figure 3.1. Here, $\overline{I^1} = 1 - p_d$, which corresponds to the capacity of the erasure channel. The difference between erasure

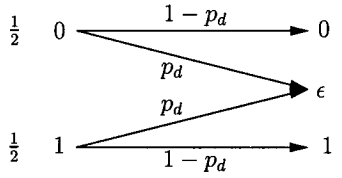

Figure 3.1: Binary deletion channel, case $n = 1$.

of bits and deletion of bits can be seen by sending a block of 2 bits over the deletion channel, as shown in Figure 3.2. One can show that $\overline{I^1} > \overline{I^2}$ for $0 < p_d < 1$, and this

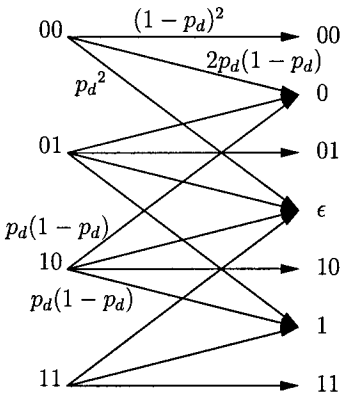

Figure 3.2: Binary deletion channel, case $n = 2$.

is different from the erasure channel for which $I(X;Y) = \frac{I(X^n;Y^n)}{n}$. This phenomenon occurs because the recipient does not know with certainty the position of the deleted bits. If, for example, 01 is sent and 0 is erased, the recipient knows that 01 or 11 was sent. On the other hand, if 0 is deleted, the recipient must decide between 01, 10, and 11. The longer the input sequence is, the harder it is to correct deletions as opposed to erasures. Furthermore, the mutual information is maximized when the probability of transmitting 00 or 11 is larger than the probability of transmitting 01 or 10. This can be explained by the fact that the number of subsequences when deleting bits from a string of length n depends on the structure of the string[2].

[2]Calculating the number of subsequences when deleting symbols from a string is an interesting problem in its own right. It is studied in the next chapter.

The channel coding theorem and its converse for channels admitting synchronization errors can be used to derive a numerical upper bound for the capacity of the deletion channel.

Theorem 3.1 (Dobrushin [24]).

$$C = \lim_{n\to\infty} \overline{I^n} = \lim_{n\to\infty} \max_{\mathcal{P}_{X^n}} \frac{I(X^n;Y^*)}{n}.$$

To illustrate the theorem, let us see how many distinguishable sequences of length n can be sent over the channel. Intuitively, the recipient can receive $\approx 2^{H(Y^*)}$ sequences, which can be divided in sets of size $\approx 2^{H(Y^*|X^n)}$ corresponding to the different inputs. The number of distinct sets is at most $\approx \frac{2^{H(Y^*)}}{2^{H(Y^*|X^n)}} = 2^{I(X^n;Y^*)}$, corresponding to the maximum number of distinguishable codewords. Hence,

$$C \approx \lim_{n\to\infty} \max_{\mathcal{P}_{X^n}} \frac{\log_2 2^{I(X^n;Y^*)}}{n}.$$

It is not hard to show that $\overline{I^n} > \overline{I^{n+1}}$ for $n \geq 1$, thus all the $\overline{I^k}$ are upper bounds for the capacity of the deletion channel. Since received sequences from the deletion channel are at most as long as the transmitted sequences, $\overline{I^k}$ can be computed for small values of k and a fixed probability of symbol deletion p_d, as done in the first algorithm.

Algorithm 1

function upper_bound(n, p_d)

1. For each input sequence $x \in \mathcal{X}^n$ beginning with a 0, find all the possible output sequences and their occurrence when bits are deleted. Store the results in the two-dimensional array A_n.
2. Using p_d, modify A_n so that its entries become output probabilities.
3. Using A_n, maximize the objective function $I(X^n;Y^*) = H(Y^*) - H(Y^* \mid X^n)$ over all the symmetric probability distributions over X^n.
4. **Return** $\overline{I^n}$.

As an example for Step 1, $x = 0001$ has the subsequences 0001, 000, 001 (3 times), 00 (3 times), 01 (3 times), 0 (3 times), 1, and ϵ. It should be clear that the mutual information is maximized for symmetric probability distributions with respect to 0 and 1.

Since $I(X^n; Y^*)$ is a concave function and the constraints are linear, the optimization problem in Step 3 can be solved using Kelley's cutting-plane algorithm [53]. An interesting property of cutting-plane algorithms is that the accuracy of the optimal solution can be specified.

Figure 3.3 shows the value of $\overline{I^n}$ for $p_d = 0.3$, $n = [1, 6]$ and a 10^{-10} accuracy. Let $\Delta_n \triangleq \overline{I^n} - \overline{I^{n-1}}$ for $n \geq 2$. Since the rate of convergence of $\overline{I^n}$ is slow, one can use the fact that $\frac{\Delta_{n+2}}{\Delta_{n+1}} \geq \frac{\Delta_{n+1}}{\Delta_n}$ to extrapolate an improved upper bound for C. Figure 3.3 shows this upper bound for $p_d = 0.3$. Algorithm 1 was executed several times with different values of n and p_d to get an upper bound for the capacity of the binary deletion channel; the results are shown in Figure 3.4. The optimization algorithm converges faster with large values of p_d, and as a result, input blocks of length up to 6 were used for $p_d \leq 0.6$, input blocks of length up to 7 for $0.6 < p_d \leq 0.8$, and input blocks of length up to 8 for $p_d \geq 0.9$.

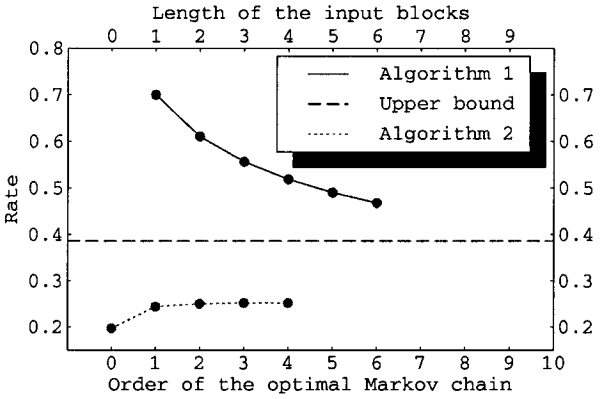

Figure 3.3: Convergence of the two algorithms to the capacity of the binary deletion channel when $p_d = 0.3$.

3.4 Approximate Lower Bound

The capacity of the binary deletion channel can also be expressed as in Theorem 3.2.

Theorem 3.2 (Dobrushin [24]).

$$C = (1 - p_d) \max_{\Xi} \lim_{n \to \infty} \frac{I(X^\infty; Y^n)}{n},$$

where the upper bound is taken over all stationary ergodic processes Ξ modeling the input source and which are Markov chains.

Let Ξ_m be the set of all Markov chains of order m, $\overline{I'} \triangleq \lim_{n\to\infty} \frac{I(X^\infty;Y^n)}{n}$, and $C_m \triangleq (1-p_d) \max_{\Xi_m} \overline{I'}$. Theorem 3.2 can be rewritten in terms of the order of the optimal Markov chains, i.e.,

$$C = \lim_{m\to\infty} C_m. \tag{3.1}$$

Since $\Xi_m \subseteq \Xi_{m+1}$, it follows that $C_1 \leq C_2 \leq \cdots < C$, thus all the C_k are lower bounds for the capacity of the deletion channel. The idea of using (3.1) to approximate the capacity is from Vvedenskaya and Dobrushin [108], and in this section their framework is extended to approximate C_m for small values of m using stochastic optimization.

Algorithm 2

function lower_bound(m, p_d)

1. Choose an arbitrary symmetric Markov chain Ξ of order m.
2. Using the Markov chain to generate long input sequences, approximate
$$\overline{I'} = \lim_{n\to\infty} \frac{H(Y^n)}{n} - \lim_{n\to\infty} \frac{H(Y^n \mid X^\infty)}{n}.$$
3. **If** Ξ does not maximize $\overline{I'}$ with the desired accuracy, **then** choose another Markov chain and go to Step 2.
4. **Return** C_m.

3.4.1 Computation of $\overline{I'}$

For a fixed Markov chain of order m modeling the input source, $\overline{I'}$ is a limit as the length of the input sequences approaches infinity and is hard to compute with accuracy. We use $\overline{I'} = \lim_{n\to\infty} \frac{H(Y^n)}{n} - \lim_{n\to\infty} \frac{H(Y^n|X^\infty)}{n}$ and approximate each term separately.

To compute $H(Y^n)$, the Markov chain is used to generate long binary sequences from which each bit is then randomly deleted with probability p_d. The resulting strings are then broken into i small output strings of length n, and the occurrence of all such strings is stored in a table used to approximate $H(Y^n)$. Although $\frac{H(Y^n)}{n}$ converges slowly,

$H(Y^{j+1}) - H(Y^j)$ is constant when j is larger than or equal to m, and this can be used to compute $\lim_{n\to\infty} \frac{H(Y^n)}{n}$ with a 10^{-5} accuracy in reasonable time.

The term $H(Y^n \mid X^\infty)$ is harder to compute accurately. The Markov chain is used to generate an input sequence x of length $N \gg n$. One needs to compute the probability of all its subsequences of length n when bits are randomly deleted with probability p_d, but the complexity of this task is exponential with respect to N. Instead, we randomly delete bits from x, take the first n bits of the resulting sequence as a possible subsequence and repeat this process k times. $H(Y^n|X^\infty = x)$ can be estimated using the k subsequences of x. By using the Markov chain to generate a large number s of sequences x_i of length N, it is possible to estimate $H(Y^n \mid X^\infty)$ by averaging all the values $H(Y^n|X^\infty = x_i)$. Again, $\frac{H(Y^n|X^\infty)}{n}$ converges slowly as n increases, but $H(Y^{l+1} \mid X^\infty) - H(Y^l \mid X^\infty)$ converges much faster and is used to speed up the calculations. In order to compute $\lim_{n\to\infty} \frac{H(Y^n|X^\infty)}{n}$ with a relative error of 10^{-2}, N, k, s and l have to be chosen such that the simulation takes several days on a 3.0GHz Pentium 4 processor. Also, when p_d increases, N has to be larger since an increasing fraction of the input bits are deleted.

3.4.2 Estimation of C_m

To estimate C_m, the concave objective function $\overline{I'}$ needs to be maximized over all Markov chains of order m. Since the channel is symmetric it is sufficient to consider symmetric Markov chains, thus for C_m there are 2^{m-1} variables. As mentioned above, computing $\overline{I'}$ accurately is prohibitive and as a result deterministic optimization techniques cannot be used. Instead, we use stochastic optimization. More precisely, we roughly approximate $\overline{I'}$ at each iteration with small values of i, j, k, l, N and s, and use the multidimensional extension of the Kiefer-Wolfowitz procedure [56]. At each iteration, the gradient can be estimated using 2^{m-1} evaluations of $\overline{I'}$. It is possible for the procedure to converge to C_m in mean square and with probability one at the price of a slow rate of convergence.

Once the Kiefer-Wolfowitz procedure gives a good approximation of the optimal Markov chain, it can be used to estimate C_m by computing $\overline{I'}$ once with a smaller relative error as explained in the previous subsection. Figure 3.3 shows C_m for $m = [0, 4]$ and $p_d = 0.3$. Algorithm 2 was executed for $m = [0, 4]$ and different values of p_d, as shown in Figure 3.4; it gives an approximate lower bound for the capacity of the binary deletion channel.

3.4.3 Discussion

Using Markov chains of order more than 1 improves on the previous lower bounds. In our simulations, there was no significant improvement when considering Markov chains of order more than 2, and in some cases using order chains of order 3 and 4 slightly reduced the capacity. This is partly due to the slow convergence of the Kiefer-Wolfowitz procedure and to the fact that the error when estimating $\lim_{n\to\infty} \frac{H(Y^n|X^\infty)}{n}$ is biased, causing the algorithm to converge slightly off the maximum. The impact of the bias could be reduced by gradually improving the accuracy of the estimation at each iteration instead of only once when the maximum is reached. Nevertheless, the capacity is certainly much closer to the lower bound than to the upper bound, since Algorithm 2 converges exponentially faster than Algorithm 1. Unfortunately, the estimation bias lower bounds $\lim_{n\to\infty} \frac{H(Y^n|X^\infty)}{n}$, thus the lower bounds for the capacity are not strictly provable and are instead approximations.

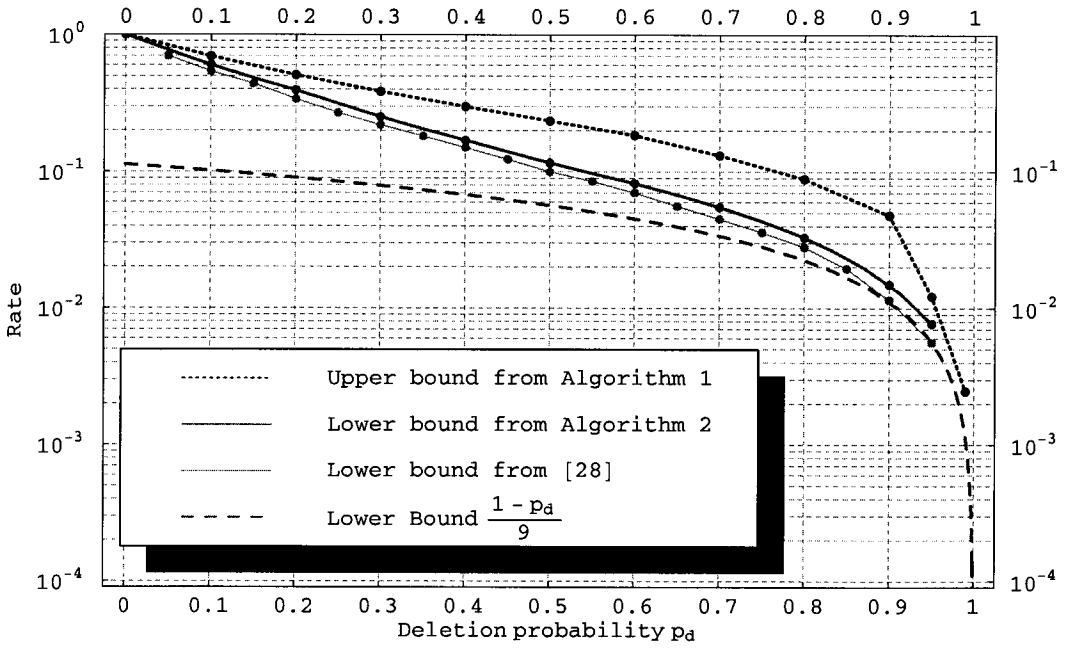

Figure 3.4: Improved bounds for the capacity of the binary deletion channel.

3.5 Summary

In this chapter, I have proposed two numerical algorithms that converge to the capacity of the binary deletion channel, one from above and one from below. This has allowed me to introduce the first nontrivial upper bound for the capacity and to obtain an approximate lower bound improving on the existing lower bounds.

Insertion-deletion error-correcting codes have not been studied as much as their counterparts for binary symmetric channels, and the structure of channels admitting synchronization errors is not completely understood. For instance, the capacity of the deletion channel can be approached by sending long messages in one block. Although it is debatable whether or not the capacity can be reached by a practical coding scheme dividing the input sequence into blocks of reasonable length, I show in Chapter 6 that it is possible to design codes for which the additional redundancy required to maintain block synchronization is not prohibitive.

Chapter 4

Subsequences, Sequences, and Supersequences

4.1 Introduction

Error-correcting codes capable of correcting insertions and deletions of symbols could be useful in several settings where such errors occur, including reading media, communication networks, multiuser optical channels, and reconciliation of correlated data. Unfortunately, insertions and deletions introduce synchronization errors that are difficult to analyze. The existing practical codes seem far from capacity and only work when deletion and insertion rates are small [20], or when there is a single error per block [64].

As a special case of a channel with synchronization errors, the deletion channel has each input symbol deleted independently with probability p_d and transmitted correctly with probability $1 - p_d$. For example, the codeword 0012101 could be received as 0211. The deletion channel can model packet-switched communication networks, where packets are either lost or dropped due to buffer overflow at finite buffer queues [21], and is also of interest because it is easier to analyze than its counterparts with insertions and deletions. Designing efficient codes for the deletion channel remains a challenge: several open questions remain even when a single deletion has to be corrected, and very little is known for more than one deletion [98]. One of the reasons why good codes are difficult to construct is that the spheres associated with codewords for the deletion channel do not have the same size, since the number of subsequences when deleting d bits from a string depends on the structure of the string itself. For instance, when two bits are deleted from 0000, the only subsequence is 00, whereas when two bits are deleted from

0101, the four 2-bit strings are possible subsequences. In fact, very few partial results are available on how to efficiently compute the number of subsequences when deleting symbols from a string. In this chapter, I study this problem to improve the understanding of deletion-correcting codes.

The rest of the article is structured as follows. In Section 4.2, I present the problem and existing results. In Section 4.3, I introduce the concepts of deletion pattern and sandwich and use them to prove closed-form expressions for the number of subsequences of a string with $d \in \{1, 2, 3, 4, 5\}$ deletions; the only existing results are for binary strings and $d \in \{1, 2\}$. The results presented in Section 4.3 can be extended for larger values of d, although the formula size increases exponentially with d. In Section 4.4, I present a simple recursive definition for the number of subsequences of a string, and use it to construct the first efficient algorithm working for any string, number of deletions, and alphabet size. A similar recursive definition for the multiplicity of a subsequence in a string and its corresponding algorithm are presented in Section 4.5. Finally, Section 4.6 concludes the chapter.

4.2 Preliminaries and Known Results

Let $\Sigma \triangleq \{0, 1, \ldots, Q-1\}$ be an alphabet with $|\Sigma| \geq 2$ symbols, and let Σ^n be the set of all strings of length n over Σ. We write $u \triangleq u_1 u_2 \ldots u_n$ to denote a Q-ary string of length n in Σ^n and $|u|$ for the length of u. When $\Sigma = \{0, 1\}$, we write $\omega(u)$ for the weight of u, i.e., the number of 1s in u. A *run* is a substring of identical symbols of maximum length, and $r(u)$ is used for the number of runs in u (we use r when no confusion is possible). A *d-subsequence* of u is a string of $n-d$ symbols that can be obtained by deleting d symbols from u. We write $D_d(u)$ for the set of d-subsequences of u and $|D_d(u)|$ for the number of d-subsequences of u. It is assumed that $|D_d(u)| = 0$ when $d > |u|$ and that $|D_d(u)| = 1$ when $d = |u|$. The *multiplicity* of a subsequence u in a string v is denoted by $M(u, v)$, with the assumption that $M(\epsilon, v) = 1$ for any string v.

Example 4.1. Consider the string $u = 01100$. The string has three runs, $|D_2(u)| = 5$, and
$D_2(u) = \{000, 010, 011, 100, 110\}$. The subsequence $v = 010$ appears four times in u, thus $M(v, u) = 4$.

The first thing to remark is that the number of d-subsequences varies widely depending

on the string. For example,

$$|D_4(000000000000000)| = 1 \text{ and } |D_4(012012012012012)| = 1233.$$

When a single symbol is deleted from u, Levenshtein [60] pointed out that the number of subsequences is equal to the number of runs in u, i.e.,

$$|D_1(u)| = r. \tag{4.1}$$

In his survey of single-deletion-correcting codes, Sloane [98] presented a formula for $|D_2(u)|$ and $\Sigma = \{0, 1\}$ using the *derivatives* of u. The first derivative of u is $u' \triangleq u'_0 u'_1 \ldots u'_{n-1}$, where $u'_i \triangleq u_i \oplus u_{i+1}$ for $1 \leq i \leq n-1$, and the higher-order derivatives of u are defined similarly. Sloane proved that

$$|D_2(u)| = \binom{r+d-1}{d} - 2\omega(u') + \omega(u'').$$

It is not clear how to generalize this result for more than two deletions, and even less so for nonbinary alphabets. Swart and Ferreira [102] proved a different but equivalent formula for $d = 2$ and binary alphabets, i.e.,

$$|D_2(u)| = \frac{1}{2}(p^2 + 2pq + q^2 - p + q) - t, \tag{4.2}$$

where p is the number of runs of length 1 in u, q the number of runs longer than 1, and t the number of runs of length 1 not located at one of the ends of u.

A trivial upper bound for the number of d-subsequences is given by (4.3), where the first term is the number of strings of length $n-d$ over Σ and the second term the number of d-tuples of runs from where the symbols can be deleted.

$$|D_d(u)| \leq \min(|\Sigma|^{n-d}, r^d). \tag{4.3}$$

Levenshtein [60] established the following bounds for a string u with r runs, i.e.,

$$\binom{r-d+1}{d} \leq |D_d(u)| \leq \binom{r+d-1}{d}. \tag{4.4}$$

The upper bound is tight if and only if all the runs of u are greater than or equal to d. The lower bound is not tight, but Hirschberg and Régnier [47] have established the

following lower bound, which is tight for the strings 010101... and 101010... .

$$|D_d(u)| \geq \sum_{i=0}^{d} \binom{r-d}{i}. \tag{4.5}$$

The maximum number of subsequences for a string of length n is also a problem of interest. Let $\mu_{|\Sigma|}(n,d) \triangleq \max_{u \in \Sigma^n} |D_d(u)|$. In an unpublished report, Calabi [11] (as cited in [12]) proved that

$$\mu_2(n,d) = \sum_{i=0}^{d} \binom{n-d}{i}, \tag{4.6}$$

which is much better than the upper bound from (4.4) and matches the lower bound from (4.5) for the strings 010101... and 101010.... Hirschberg and Régnier [47] generalized (4.6) for nonbinary alphabets, i.e.,

$$\mu_{|\Sigma|}(n,d) = \sum_{i=0}^{d} \binom{n-d}{i} \mu_{|\Sigma|-1}(d, d-i). \tag{4.7}$$

The value $\mu_{|\Sigma|}(n,d)$ is reached for $u = \sigma\sigma\sigma\ldots$, where σ is the concatenation of all the symbols of Σ in any order. Eq. (4.7) was recently used by Levenshtein [62] to study the number of subsequences required to reconstruct the original string.

Let $E_d(n) \triangleq \sum_{u \in \Sigma^n} \frac{|D_d(u)|}{|\Sigma|^n}$ be the expected number of d-subsequences over all strings of length n. Calabi and Hartnett [12] proved that

$$E_2(n) = \frac{n(|\Sigma|-1)+1}{|\Sigma|},$$

and Hirschberg and Régnier [47] generalized the result for $0 < d < n$, i.e.,

$$E_d(n) = \sum_{i=0}^{d} \binom{n-d-1+i}{i} \left(1 - \frac{1}{|\Sigma|}\right)^i. \tag{4.8}$$

4.3 Closed-Form Formulae for the Number of Subsequences when $d \leq 5$

In this section, I present a framework that can be used to derive closed-form expressions for the number of d-subsequences of any string over any alphabet. I use the framework to prove formulae for $d \in \{1,2,3,4,5\}$. Let $r_k(u)$ be the number of runs of length k in u (r_k is used when no confusion is possible), and let the d-tuple $R_d(u) \triangleq (r(u), r_1(u), r_2(u), \ldots, r_{d-1}(u)) \triangleq (r, r_1, r_2, \ldots, r_{d-1})$ be the *run distribution* of u. Since subsequences are uniquely determined by the number of symbols deleted from each run, there is no need to distinguish r_k from r_l for $k, l \geq d$; it is sufficient to know that there are $r - r_1 - r_2 - \cdots - r_{d-1}$ *large* runs.

4.3.1 Deletion Patterns and Sandwiches

Definition 4.2. A *deletion pattern* δ *of size* d is an ordered multiset of d integers such that each i in δ represents a deletion in the i-th run of u, and such that the multiplicity of i is smaller than or equal to the length of the i-th run. For example, the deletion pattern $\{1,1,2,4,4,4\}$ for the string $u = 0001221111$ results in the 6-subsequence 0221. Since the number of deletion patterns depends only on the run distribution of a string u, we write $\mathcal{P}_d(R_d(u))$ or $\mathcal{P}_d(r, r_1, r_2, \ldots, r_{d-1})$ for the set of deletion patterns of size d of a string u whose run distribution is $R_d(u)$, and $|\mathcal{P}_d(R_d(u))|$ or $|\mathcal{P}_d(r, r_1, r_2, \ldots, r_{d-1})|$ for the number of deletion patterns. It is assumed that $|\mathcal{P}_0(R_0(u))| = 1$.

Although the run distribution is sufficient to calculate the number of deletion patterns of a string, it is not descriptive enough to derive its number of subsequences. As shown in Table 4.1, the nine strings have the same run distribution, however, each has a different number of 3-subsequences. Hence, one needs to know more about the structure of u in order to determine $|D_d(u)|$. For binary strings, any extra information necessary can be obtained from the sequence of the lengths of its runs. On the other hand, this does not suffice for larger alphabets, and information about the actual symbols of each run must further be known. This situation is also depicted in Table 4.1, where all the strings contain three runs of length 1 followed by three runs of length 2. This happens because there are distinct deletion patterns that generate similar subsequences; we call these unnecessary deletion patterns *redundant deletion patterns*. In order to count the number of d-subsequences of a string, one needs to count how many redundant deletion patterns it contains. To do so, the concept of sandwich is introduced.

Definition 4.3. A *sandwich* is a sequence of consecutive symbols squeezed between identical symbols, such that its ends are different from the squeezing symbol. The runs of either side of the sandwich are called the *squeezing runs*, and the *length* of a sandwich is the number of symbols between the squeezing runs.

It is worth noting that when $\Sigma = \{0, 1\}$, each run not in the beginning nor the end of a string is also a sandwich; the difference appears for nonbinary alphabets. Figure 4.1 shows that the string 1120200130 has two sandwiches of length 1, one sandwich of length 2, and two sandwiches of length 5. The sandwich of length 2 has a squeezing run of length 1 and a squeezing run of length 2.

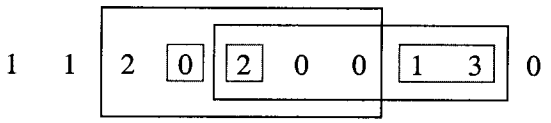

Figure 4.1: Sandwiches for the string $u = 1120200130$.

Figure 4.2: Redundant deletion pattern when deleting a sandwich of length 2 and one additional symbol.

When a sandwich is deleted from a string, redundant deletion patterns occur if symbols are deleted in the run formed by the concatenation of the merging squeezing runs. In Figure 4.2, if the sandwich of length 2 is deleted, then deleting a symbol in any of its

Table 4.1: Strings with a $R_3(3, 3, 0)$ run distribution and their respective number of 3-subsequences.

$u^{(1)} = 101001100$	$\lvert D_3(u^{(1)})\rvert = 26$
$u^{(2)} = 202001100$	$\lvert D_3(u^{(2)})\rvert = 27$
$u^{(3)} = 202001122$	$\lvert D_3(u^{(3)})\rvert = 28$
$u^{(4)} = 201001100$	$\lvert D_3(u^{(4)})\rvert = 29$
$u^{(5)} = 201001122$	$\lvert D_3(u^{(5)})\rvert = 30$
$u^{(6)} = 102001122$	$\lvert D_3(u^{(6)})\rvert = 31$
$u^{(7)} = 212001122$	$\lvert D_3(u^{(7)})\rvert = 32$
$u^{(8)} = 021001122$	$\lvert D_3(u^{(8)})\rvert = 33$
$u^{(9)} = 012001122$	$\lvert D_3(u^{(9)})\rvert = 34$

squeezing runs gives the same 3-subsequence 1120200. Thus, one of the deletion patterns $\{5, 6, 7\}$ and $\{6, 7, 8\}$ is redundant.

4.3.2 An Improved Upper Bound for $|D_d(u)|$

The upper bound in (4.4) counts the number of ways to delete d symbols from r runs of length at least d. If u contains smaller runs, then some combinations are invalid, for example deleting three symbols in a run of length 2. Consequently, the upper bound can be improved by considering deletion patterns and sandwiches, as described in Lemma 4.4.

Lemma 4.4.

$$|D_d(u)| \leq |\mathcal{P}_d(R_d(u))|,$$

where the bound is tight if and only if u has no sandwich of length smaller than d.

Proof. Clearly, each distinct d-subsequence of u corresponds to at least one distinct deletion pattern of size d, which proves the upper bound.

We prove both directions of the second statement by contrapositive, starting with the sufficient condition. Suppose that the bound is not tight. It follows that there are two different deletion patterns $\{a_1, a_2, \ldots, a_d\}$ and $\{b_1, b_2, \ldots, b_d\}$ generating the same subsequence. Since the deletion patterns are different then there is at least one of the pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_d, b_d)$ whose elements were not deleted in the same run. Let (a_k, b_k) be the first such pair. Without loss of generality, it is assumed that ρ^a, the run containing a_k, is located before ρ^b, the run containing b_k. Since the length of ρ^a after the deletions depends on which deletion pattern is applied, it follows that the only way for the subsequences to be equal is for the run(s) to the right of ρ^a to be deleted completely until a matching symbol is merged. This scenario corresponds to the deletion of a sandwich, which is a contradiction.

Finally, for the necessary condition, if u has a sandwich of length smaller than d, then, from the previous subsection, deleting the sandwich and any of its squeezing runs gives the same subsequence, hence the upper bound is not tight. The result follows. □

When all the runs of u are at least as long as d, then of course u has no short sandwich. Hence, the upper bound of (4.4) cannot be better than the bound of Lemma 4.4.

The number of deletion patterns of a string can be evaluated using generating functions, as shown next.

Lemma 4.5.

$$|\mathcal{P}_d(r, r_1, r_2, \ldots, r_{d-1})| = [z^d]((1+z)^{r_1} \cdot (1+z+z^2)^{r_2} \ldots (1+z+\cdots+z^d)^{r-r_1-\cdots-r_{d-1}}),$$

where $[z^n]f(z)$ *denotes the coefficient of* z^n *of the polynomial* $f(z)$.

Proof. Let $f(z) = (1+z)^{r_1}(1+z+z^2)^{r_2} \ldots (1+z+\cdots+z^d)^{\bar{r}_d}$. The i-th factor of $f(z)$ for $1 \leq i < d$ enumerates the ways to delete symbols in the runs of length i, and the d-th factor the number of ways to delete symbols in the runs of length greater than or equal to d. All the large runs are treated the same way, since with d deletions, the worst that can happen is for a single run to be deleted completely. The coefficient of the term of degree d in $f(z)$ when expanded gives the number of deletion patterns of size d of u. □

It is worth mentioning that the d factors of $f(z)$ can be approximated with the first $d+1$ terms of their respective Maclaurin series. Multiplying all the factors and taking the coefficient of the term of degree d of the resulting product, it is then possible to obtain closed-form expressions, although they do not reveal how the number of deletion patterns scales with d.

A more intuitive method for deriving closed-form formulae for the number of deletion patterns is to subtract the invalid patterns from the upper bound of (4.4) using a technique reminiscent of the inclusion-exclusion counting principle.

Definition 4.6. Let $\lambda = \{\lambda_1, \lambda_2, \ldots\}$ be an ordered multiset of integers. We write $|\lambda|_k$ for the number of occurrences of k in λ, $|\lambda| \triangleq \sum_k |\lambda|_k$ for the number of elements in λ, and $||\lambda||$ for the sum of all the elements in λ. Finally, let Λ be the set of all the multisets λ representing partitions of k, where $1 \leq k \leq d - |\lambda|$.

Lemma 4.7.

$$|\mathcal{P}_d(r, r_1, r_2, \ldots, r_{d-1})| = \binom{r+d-1}{d} - \sum_{\lambda \in \Lambda} \left[(-1)^{|\lambda|+1} \binom{r+d-||\lambda||-|\lambda|-1}{d-||\lambda||-|\lambda|} \prod_{1 \leq k \leq d-|\lambda|} \binom{r_k}{|\lambda|_k} \right].$$

Proof. The upper bound from (4.4) corresponds to the number of ordered multisets of size d with elements in $\{1, 2, \ldots, r\}$, or, equivalently, the number of deletion patterns of size d of a string when the length of all its runs is at least d. Let Γ be the set of all

such multisets. When u has small runs, the idea is to subtract the multisets that do not correspond to valid deletion patterns.

Suppose that the length of the i-th run of u is $k < d$. It follows that all the multisets in Γ with more than k occurrences of i are not valid deletion patterns, and there are $\binom{r+d-k-2}{d-k-1}$ of them. Repeating the same process with all the runs smaller than d, the total number of multisets that are subtracted from Γ is

$$\sum_{k=1}^{d-1} \binom{r+d-k-2}{d-k-1} r_k. \tag{4.9}$$

The problem with the previous result is that some multisets may be deleted more than once. For instance, consider the string 0110000011111 and $d = 5$. The multiset $\{1,1,2,2,2\}$ is subtracted from Γ twice, because it contains too many instances of the first run and too many instances of the second run. More precisely, for each partition $\lambda = \{a, b\}$ of k where $2 \leq k \leq d-2$,

$$\binom{r+d-a-b-3}{d-a-b-2} r_a r_b \tag{4.10}$$

multisets were deleted more than once in (4.9) and need to be added back if $a \neq b$, and

$$\binom{r+d-a-b-3}{d-a-b-2} \binom{r_a}{2} \tag{4.11}$$

multisets need to be added back if $a = b$. One can easily see that in (4.10) and (4.11), some multisets are added back more than once, so all the partitions of $k \leq d-3$ in three parts have to be considered, and so on. This is a variation of the inclusion-exclusion counting principle, and the result follows when considering all the partitions $\lambda \in \Lambda$.

□

Corollary 4.8. *The number of deletion patterns for $d \in \{1,2,3,4,5\}$ is given by*

$$\begin{aligned}
|\mathcal{P}_1(r)| &= \binom{r}{1};\\
|\mathcal{P}_2(r,r_1)| &= \binom{r+1}{2} - r_1;\\
|\mathcal{P}_3(r_1,r_2)| &= \binom{r+2}{3} - r_2 - r_1 r;\\
|\mathcal{P}_4(r,r_1,r_2,r_3)| &= \binom{r+3}{4} - r_3 - r_2 r - r_1\binom{r+1}{2} + \binom{r_1}{2};\\
|\mathcal{P}_5(r,r_1,r_2,r_3,r_4)| &= \binom{r+4}{5} - r_4 - r_3 r - r_2\binom{r+1}{2} - r_1\binom{r+2}{3}\\
&\quad + r\binom{r_1}{2} + r_1 r_2.
\end{aligned}$$

4.3.3 Sandwiches Revisited

The number of redundant deletion patterns caused by the deletion of a sandwich of length $l < d$ from a string u corresponds to the number of deletion patterns of size $d - l - 1$ of u when excluding the sandwich and a squeezing symbol on each side of it, which in turn depends on the run distribution of u, the run distribution of s, and the length of the squeezing runs. It should be noted that the merging squeezing runs still have to be treated separately, since in the original string, deletions in the left squeezing run are not equivalent to deletions in the right squeezing run. Moreover, it is not necessary to differentiate between the squeezing runs of length greater than $d - l - 1$, because they all have the same effect on the run distribution of the remaining substring.

Definition 4.9. A sandwich can be described using a tuple $s \triangleq (l, R_l, q_1, q_2)$, where l is the length of the sandwich, $r_i(s)$ is the number of runs of length i in the sandwich, $R_l \triangleq (r_1(s), r_2(s), \ldots, r_l(s))$ is the run distribution of the sandwich, and q_1 and q_2 are the length of the smallest and largest squeezing runs, respectively. We write $|\mathcal{P}_i(u,s)|$ for the number of deletion patterns of size i of the string u from which the sandwich s and a squeezing symbol on each side have been removed, and for which the merging squeezing runs are still treated as two different runs. Finally, let S be an ordered multiset of sandwiches, i.e.,

$$S \triangleq \{s^{[1]}, s^{[2]}, \ldots\} \triangleq \{(l^{[1]}, R^{[1]}_{l^{[1]}}, q_1^{[1]}, q_2^{[1]}), (l^{[2]}, R^{[2]}_{l^{[2]}}, q_1^{[2]}, q_2^{[2]}), \ldots\}.$$

We write $|\mathcal{P}_i(u, S)|$ for the number of deletion patterns of size i of the string u from which all the sandwiches in S and a squeezing symbol on each side have been removed, and for which all the merging squeezing runs are considered different.

In Figure 4.3, the sandwich of length 2 is again deleted, but this time we are looking for 5-subsequences. Subtracting two symbols to delete the sandwich and one symbol to be deleted in any of its squeezing runs leaves one redundant deletion pattern for each way to delete two symbols in the remaining substring. More precisely, each element in the set $\{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{2,4\},\{2,5\},\{3,4\},\{3,5\},\{4,5\}\}$ generates a redundant deletion pattern, because its union with $\{5,6,7\}$ and $\{6,7,8\}$ generates the same subsequence. Formally, the run distribution of the string is $R_5(u) = (8,6,2,0,0)$.

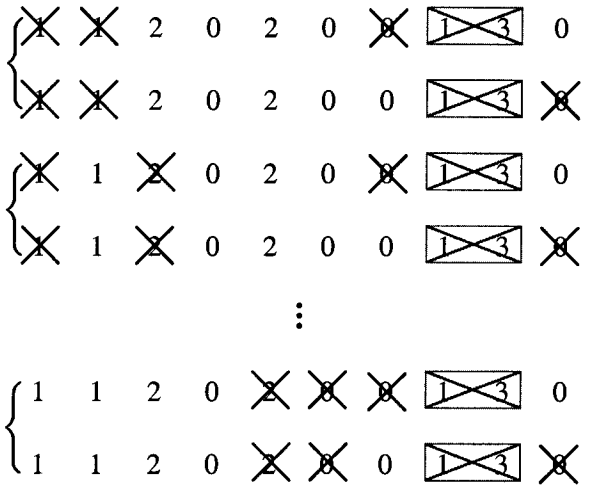

Figure 4.3: Redundant deletion patterns when deleting a sandwich of length 2 and three additional symbols.

Deleting the sandwich gets rid of two runs of length 1, deleting the right squeezing symbol eliminates another run of length 1, and deleting the left squeezing symbol transforms a run of length 2 into a run of length 1. Hence, the run distribution becomes $R_2(u') = (5,4)$. From Corollary 4.8, we can conclude that the number of redundant deletion patterns generated by the sandwich is $|\mathcal{P}_2(1120200130,(2,(2,0),1,2))| = |\mathcal{P}_2(5,4)| = 11$.

4.3.4 Closed-Form Expressions

In order to count the number of d-subsequences of a string, one needs to count how many of its deletion patterns are redundant by analyzing all the possible types of sandwiches and the redundant deletion patterns they introduce. It was shown in the previous

subsection that the number of redundant deletion patterns caused by the deletion of a sandwich from a string depends on the run distribution of the sandwich and the length of its squeezing runs. We therefore define the following variables.

Definition 4.10. Let s_l be the number of sandwiches of length l in the string u. Also, let $s^{q_1 q_2}_{l,r_1(s)r_2(s)\ldots r_l(s)}$ denote the number of sandwiches s of length l in u whose run distribution is $R_l = (r_1(s), r_2(s), \ldots, r_l(s))$, and whose squeezing runs are of length q_1 and q_2 where $q_1 \leq q_2$. We write $\overline{q_1}$ and $\overline{q_2}$ to indicate that the length of a squeezing run is at least q_1 or q_2, respectively. For example, the first sandwich of length 5 in Figure 4.1 is an instance of $s^{12}_{5,31}$, but also an instance of $s^{1\overline{1}}_{5,31}$, $s^{1\overline{2}}_{5,31}$, $s^{\overline{1}\overline{1}}_{5,31}$, $s^{\overline{1}2}_{5,31}$, and $s^{\overline{1}\overline{2}}_{5,31}$. Since there is only one type of sandwich of length 1, we write $s^{q_1 q_2}_1$ instead of $s^{q_1 q_2}_{1,1}$. Also, it should be clear that $s^{\overline{q_1} q_2}_{l,R_l} = s^{q_1 q_2}_{l,R_l} + s^{(q_1+1)q_2}_{l,R_l} + \cdots + s^{(q_1'-1)q_2}_{l,R_l} + s^{\overline{q_1'} q_2}_{l,R_l}$ for any $q_1' > q_1$.

We now prove closed-form expressions for $|D_d(u)|$ when $d \in \{1,2,3,4,5\}$, starting with a new proof for the case $d = 1$.

Lemma 4.11. *The number of subsequences when deleting one symbol from a string u is*

$$|D_1(u)| = r.$$

Proof. When a single symbol is deleted from u, then clearly all the deletion patterns are distinct, thus

$$|D_1(u)| = |\mathcal{P}_1(r)| - 0 = r.$$

□

Lemma 4.12. *The number of subsequences when deleting two symbols from a string u is*

$$|D_2(u)| = \binom{r+1}{2} - r_1 - s_1.$$

Proof. When two symbols are deleted from a string, the only redundant deletion patterns occur for sandwiches of length 1. For each such sandwich, one symbol is required to delete the sandwich, leaving one symbol that can be deleted in either of the squeezing runs. Thus,

$$|D_2(u)| = |\mathcal{P}_2(r, r_1)| - s_1,$$

and the result follows from Corollary 4.8. □

For binary alphabets, s_1 is exactly the number of runs of length 1 not located at one of the ends of the string, thus by using $r_1 = p$, $r = p + q$, and $s_1 = t$, it is easy to show that

(4.2) is a special case of Lemma 4.12. The fact that the number of d-subsequences of a string also depends on its sandwiches when $d > 2$ is one reason why efficient block codes capable of correcting more than one deletion error have proved difficult to construct.

Lemma 4.13. *The number of subsequences when deleting three symbols from a string u is*

$$|D_3(u)| = \binom{r+2}{3} - r_2 - r_1 r - s_2 - s_1^{11}(r-3) - s_1^{1\overline{2}}(r-2) - s_1^{\overline{22}}(r-1).$$

Proof. First, each sandwich of length 2 generates a redundant deletion pattern, as shown in Figure 4.2. Second, there are three types of sandwiches of length 1 that generate redundant deletion patterns: a symbol between two squeezing runs of length 1, a symbol between a squeezing run of length 1 and a squeezing run of length at least 2, and a symbol between two squeezing runs of length at least 2. Thus, from the previous subsection, the number of 3-subsequences of a string is

$$|D_3(u)| = |\mathcal{P}_3(r, r_1, r_2)| - s_2 - s_1^{11}|\mathcal{P}_1(r-3)| - s_1^{1\overline{2}}|\mathcal{P}_1(r-2)| - s_1^{\overline{22}}|\mathcal{P}_1(r-1)|,$$

and the result follows from Corollary 4.8. □

Unfortunately, the approach used so far cannot be used to compute $|D_d(u)|$ for $d > 3$, because some redundant deletion patterns are deleted more than once. Consider the number of 4-subsequences of the string $u = 0100022212$. As shown in Figure 4.4, from the first sandwich of length 1, we find that the deletion patterns $\{1,2,4,5\}$ and $\{2,3,4,5\}$ generate the same subsequence, as do $\{1,2,5,6\}$ and $\{2,3,5,6\}$. From the second sandwich of length 1, we find that the deletion patterns $\{1,2,4,5\}$ and $\{1,2,5,6\}$ generate the same subsequence, as do $\{2,3,4,5\}$ and $\{2,3,5,6\}$. The four deletion patterns generate the subsequence 000222, and if the redundant deletion patterns for each sandwich are deleted independently, four deletion patterns will be deleted instead of three. It is important to note that this does not occur for consecutive sandwiches of length 1, as in the string 0000202222, since each sandwich is also a squeezing symbol of the other sandwich.

Definition 4.14. Let s_{a-b} be the number of consecutive sandwiches of length a and b (in any order) in a string u. As previously, $s_{a-b}^{q_1 q_2}$ is used to specify the length of the squeezing runs. For example, s_{1-1} is the number of consecutive sandwiches of length 1 in

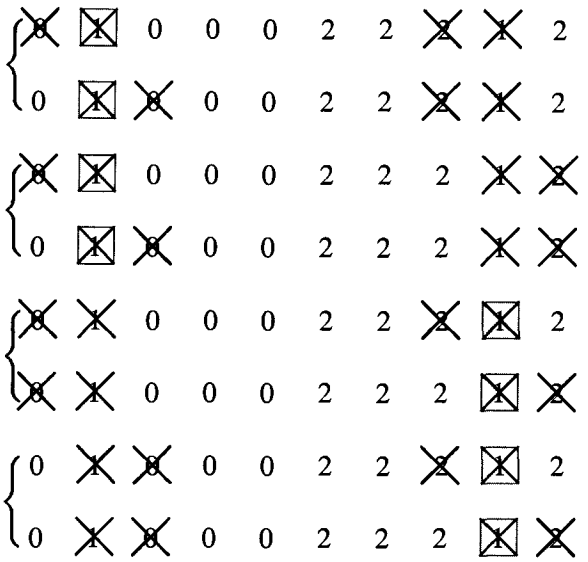

Figure 4.4: Multiple deletion of a redundant deletion pattern.

u; $u = 001011$ has one instance of consecutive sandwiches of length 1, and $u = 01010202$ has three such instances.

Lemma 4.15. *The number of subsequences when deleting 4 symbols from a string u is*

$$\begin{aligned}
|D_4(u)| \;=\; & \binom{r+3}{4} - r_3 - r_2 r - r_1\binom{r+1}{2} + \binom{r_1}{2} - s_3 - s_{2,2}^{11}(r-4) \\
& -(s_{2,01}^{11} + s_{2,2}^{1\overline{2}})(r-3) - (s_{2,01}^{1\overline{2}} + s_{2,2}^{\overline{22}})(r-2) - s_{2,01}^{\overline{22}}(r-1) \\
& -s_1^{11}\left(\binom{r-2}{2} - r_1 + 3\right) - s_1^{12}\left(\binom{r-1}{2} - r_1 + 1\right) \\
& -s_1^{1\overline{3}}\left(\binom{r-1}{2} - r_1 + 2\right) - s_1^{22}\left(\binom{r}{2} - r_1 - 1\right) - s_1^{2\overline{3}}\left(\binom{r}{2} - r_1\right) \\
& -s_1^{\overline{33}}\left(\binom{r}{2} - r_1 + 1\right) + \binom{s_1}{2} - s_{1-1}.
\end{aligned}$$

Proof. The number of redundant deletion patterns deleted when considering all the sandwiches independently is

$$\begin{aligned}
& s_3 + s_{2,2}^{11}|\mathcal{P}_1(r-4)| + (s_{2,01}^{11} + s_{2,2}^{1\overline{2}})|\mathcal{P}_1(r-3)| + (s_{2,01}^{1\overline{2}} + s_{2,2}^{\overline{22}})|\mathcal{P}_1(r-2)| \\
& + s_{2,01}^{\overline{22}}|\mathcal{P}_1(r-1)| + s_1^{11}|\mathcal{P}_2(r-3, r_1-3)| + s_1^{12}|\mathcal{P}_2(r-2, r_1-1)| \\
& + s_1^{1\overline{3}}|\mathcal{P}_2(r-2, r_1-2)| + s_1^{22}|\mathcal{P}_2(r-1, r_1+1)| + s_1^{2\overline{3}}|\mathcal{P}_2(r-1, r_1)| \\
& + s_1^{\overline{33}}|\mathcal{P}_2(r-1, r_1-1)|.
\end{aligned} \tag{4.12}$$

Furthermore, as mentioned before the lemma, there is a redundant deletion pattern that should not have been deleted for every pair of nonconsecutive sandwiches of length 1. Putting everything together, we get

$$|D_4(u)| = |\mathcal{P}_4(r, r_1, r_2, r_3)| - (4.12) + \binom{s_1}{2} - s_{1-1}.$$

The result follows from Corollary 4.8. □

Redundant deletion patterns that are deleted more than once are very similar to what occurred in Lemma 4.7 when counting the number of deletion patterns, hence closed-form expressions for the number of d-subsequences can also be derived using the inclusion-exclusion counting principle.

Definition 4.16. Let $\lambda = \{\lambda_1, \lambda_2, \dots\}$ be an ordered multiset of integers and Λ be the set of all the multisets λ representing partitions of $k \leq d - |\lambda|$, as presented in Definition 4.6. Moreover, let S be an ordered multiset of sandwiches as introduced in Definition 4.9, and let $|S|$ and $|S|_s$ be number of sandwiches in S and the multiplicity of the sandwich s in S, respectively. We use S_λ for the set of all the ordered multisets of sandwiches whose lengths correspond to the elements of λ, i.e., $S_\lambda \triangleq \{S \text{ such that } |S| = |\lambda| \text{ and } l^{[i]} = \lambda_i \text{ for } 1 \leq i \leq |\lambda|\}$.

Theorem 4.17. *The number of subsequences when deleting d symbols from a string u is given by*

$$|D_d(u)| = |\mathcal{P}_d(R_d(u))| - \sum_{\lambda \in \Lambda} \left[(-1)^{|\lambda|+1} \left(\sum_{S \in S_\lambda} \left[|\mathcal{P}_{d-|\lambda|-||\lambda||}(u, S)| \prod_{(l,R_l,q_1,q_2) \in S} \binom{s^{q_1 q_2}_{l,R_l}}{|S|_{(l,R_l,q_1,q_2)}} \right] - \Delta_\lambda \right) \right],$$

and all the correction terms Δ_λ are equal to 0 if every symbol of u belongs to at most one sandwich (including the squeezing runs).

Proof. First, recall that the number of d-subsequences can be found by counting the number of redundant deletion patterns that need to be subtracted from $|\mathcal{P}_d(R_d(u))|$. Following the discussion from the previous subsection, the number of redundant deletion patterns, if all the sandwiches of u are analyzed independently, is

$$\sum_{\{\lambda_1\} \in \Lambda} \sum_{(\lambda_1, R_{\lambda_1}, q_1, q_2) \in S_{\{\lambda_1\}}} |\mathcal{P}(u, (\lambda_1, R_{\lambda_1}, q_1, q_2))| s^{q_1 q_2}_{\lambda_1, R_{\lambda_1}}. \tag{4.13}$$

Second, consider two sandwiches in u that do not share any symbol, even when considering their squeezing runs. Without loss of generality, we consider two sandwiches made of the m-th and n-th runs of u, respectively. As mentioned above in the analysis of the string in Figure 4.4, when both sandwiches are deleted, the deletion patterns $\{m-1, m, m, \ldots, m, n-1, n, n, \ldots, n\}$, $\{m-1, m, m, \ldots, m, n, n, \ldots, n, n+1\}$, $\{m, m, \ldots, m+1, n-1, n, n, \ldots, n\}$ and $\{m, m, \ldots, m, m+1, n, n, \ldots, n, n+1\}$ generate the same subsequence, but four are deleted instead of three. Moreover, it is clear that adding identical deletions in the four deletion patterns will also create the same problem, i.e., four new deletion patterns generating a common subsequence that get deleted instead of three. In fact, the number of deletion patterns that should not have been deleted in (4.13) corresponds to the number of deletion patterns of the string minus the two sandwiches and a squeezing symbol on each side of them. It follows that when considering all the pairs of sandwiches, the number of deletion patterns that should not have been deleted from (4.13) is

$$\sum_{\{\lambda_1,\lambda_2\}\in\Lambda} \sum_{\{s^{[1]},s^{[2]}\}\in S_{\{\lambda_1,\lambda_2\}}} |\mathcal{P}_{d-\lambda_1-\lambda_2-2}(u, \{s^{[1]}, s^{[2]}\})| \cdot \alpha, \tag{4.14}$$

where $s^{[1]} = (l^{[1]}, R^{[1]}_{l^{[1]}}, q^{[1]}_1, q^{[1]}_2)$ and $s^{[2]} = (l^{[2]}, R^{[2]}_{l^{[2]}}, q^{[2]}_1, q^{[2]}_2)$ are two sandwiches, $\#s^{[1]}$ and $\#s^{[2]}$ are the number of occurrences of $s^{[1]}$ and $s^{[2]}$ in u, respectively, and $\alpha = \#s^{[1]} \cdot \#s^{[2]}$ if $s^{[1]} \neq s^{[2]}$ and $\alpha = \binom{\#s^{[1]}}{2}$ if $s^{[1]} = s^{[2]}$. If two sandwiches in u share some symbols, then subtracting the two sandwiches and a squeezing symbol on each side might not cause the problem occurring in Figure 4.4 and for which (4.14) was derived. This explains why a correction term $\Delta_{\{\lambda_1,\lambda_2\}}$ needs to be subtracted to account for sandwiches sharing some symbols.

Eq. (4.14) adds some deletion patterns more than once, so that all the ways to delete three sandwiches from u have to considered, and so on. This procedure is again an application of the inclusion-exclusion counting principle, and the result follows when accounting for all the partitions in Λ. □

Definition 4.18. Let s_{1-x-1} be the number of instances of sandwiches of length 1 separated by a single symbol. For instance, $s_{1-x-1} = 1$ for $u = 000102000$ and $s_{1-x-1} = 2$ for $u = 010101$.

We use Theorem 4.17 to find a closed-form expression for the number of 5-subsequences of a string. The theorem is only moderately useful because it does not describe how to derive the correction terms Δ_λ. Since the resulting closed-form formula is quite large

and can easily be obtained with Corollary 4.8, only the terms for each member of Λ are described.

The number of redundant deletion patterns when considering all the sandwiches of length 1 to 4 independently, corresponding to $\lambda = \{1\}, \lambda = \{2\}, \lambda = \{3\}$, and $\lambda = \{4\}$, is

$$\begin{aligned}
&s_4 + s_{3,3}^{11}|\mathcal{P}_1(r-5)| + (s_{3,3}^{1\overline{2}} + s_{3,11}^{11})|\mathcal{P}_1(r-4)| + (s_{3,001}^{11} + s_{3,11}^{1\overline{2}} + s_{3,3}^{\overline{22}})|\mathcal{P}_1(r-3)| \\
&+ (s_{3,001}^{1\overline{2}} + s_{3,11}^{\overline{22}})|\mathcal{P}_1(r-2)| + s_{3,001}^{\overline{22}}|\mathcal{P}_1(r-1)| + s_{2,01}^{22}|\mathcal{P}_2(r-1, r_1+2)| \\
&+ s_{2,01}^{2\overline{3}}|\mathcal{P}_2(r-1, r_1+1)| + s_{2,01}^{\overline{33}}|\mathcal{P}_2(r-1, r_1)| + s_{2,2}^{11}|\mathcal{P}_2(r-4, r_1-4)| \\
&+ (s_{2,01}^{11} + s_{2,2}^{12})|\mathcal{P}_2(r-3, r_1-2)| + s_{2,2}^{1\overline{3}}|\mathcal{P}_2(r-3, r_1-3)| \\
&+ (s_{2,01}^{12} + s_{2,2}^{22})|\mathcal{P}_2(r-2, r_1)| + (s_{2,01}^{1\overline{3}} + s_{2,2}^{2\overline{3}})|\mathcal{P}_2(r-2, r_1-1)| \\
&+ s_{2,2}^{\overline{33}}|\mathcal{P}_2(r-2, r_1-2)| + s_1^{11}|\mathcal{P}_3(r-3, r_1-3, r_2)| + s_1^{12}|\mathcal{P}_3(r-2, r_1-1, r_2-1)| \\
&+ s_1^{13}|\mathcal{P}_3(r-2, r_1-2, r_2+1)| + s_1^{1\overline{4}}|\mathcal{P}_3(r-2, r_1-2, r_2)| \\
&+ s_1^{22}|\mathcal{P}_3(r-1, r_1+1, r_2-2)| + s_1^{23}|\mathcal{P}_3(r-1, r_1, r_2)| + s_1^{2\overline{4}}|\mathcal{P}_3(r-1, r_1, r_2-1)| \\
&+ s_1^{33}|\mathcal{P}_3(r-1, r_1-1, r_2+2)| + s_1^{3\overline{4}}|\mathcal{P}_3(r-1, r_1-1, r_2+1)| \\
&+ s_1^{\overline{44}}|\mathcal{P}_3(r-1, r_1-1, r_2)|.
\end{aligned} \tag{4.15}$$

In (4.15), there is a nonredundant deletion pattern that gets deleted for each nonconsecutive combination of a sandwich of length 1 and a sandwich of length 2, thus we need to add

$$s_1 s_2 - s_{1-2} \tag{4.16}$$

deletion patterns; this corresponds to $\lambda = \{1, 2\}$, where $\Delta_{\{1,2\}} = s_{1-2}$. The last case to consider is $\lambda = \{1, 1\}$. Counting all the pairs of sandwiches of length 1 independently, we need to add

$$\begin{aligned}
&\binom{s_1^{11}}{2}|\mathcal{P}_1(r-6)| + \binom{s_1^{1\overline{2}}}{2}|\mathcal{P}_1(r-4)| + \binom{s_1^{\overline{22}}}{2}|\mathcal{P}_1(r-2)| \\
&+ s_1^{11}s_1^{1\overline{2}}|\mathcal{P}_1(r-5)| + s_1^{11}s_1^{\overline{22}}|\mathcal{P}_1(r-4)| + s_1^{1\overline{2}}s_1^{\overline{22}}|\mathcal{P}_1(r-3)|
\end{aligned} \tag{4.17}$$

redundant deletion patterns that were included twice in (4.15). The correction term $\Delta_{\{1,1\}}$ is more complicated to derive. First, since (4.17) includes consecutive sandwiches

of length 1 for which no deletion pattern was deleted twice in (4.15), we have to subtract

$$s_{1-1}^{11}|\mathcal{P}_1(r-6)| + s_{1-1}^{1\overline{2}}|\mathcal{P}_1(r-5)| + s_{1-1}^{\overline{22}}|\mathcal{P}_1(r-4)| \tag{4.18}$$

deletion patterns that should not have been counted in (4.17). Second, (4.17) also considers sandwiches of length 1 separated by a single symbol, and again in this case no redundant deletion pattern is deleted more than once in (4.15). Thus, we need to subtract

$$s_{1-x-1}^{11}|\mathcal{P}_1(r-6)| + s_{1-x-1}^{1\overline{2}}|\mathcal{P}_1(r-5)| + s_{1-x-1}^{\overline{22}}|\mathcal{P}_1(r-4)| \tag{4.19}$$

deletion patterns from (4.17). Third, we need to add

$$s_{1-x-1}^{11}|\mathcal{P}_1(r-5)| + s_{1-x-1}^{1\overline{2}}|\mathcal{P}_1(r-4)| + s_{1-x-1}^{\overline{22}}|\mathcal{P}_1(r-3)| \tag{4.20}$$

deletion patterns; this corresponds to redundant deletion patterns that were subtracted twice, once when deleting a sandwich of length 3 and once when deleting two sandwiches of length 1 separated by a single symbol. Consequently, $\Delta_{\{1,1\}} = (4.18) + (4.19) - (4.20)$.

Lemma 4.19. *The number of subsequences when deleting 5 symbols from a string u is*

$$|D_5(u)| = |\mathcal{P}_5(r, r_1, r_2, r_3, r_4)| - (4.15) + (4.16) + (4.17) - (4.18) - (4.19) + (4.20).$$

We find the number of 5-subsequences for the string $u = 021202112$. The run distribution of u is $(8, 7, 1, 0, 0)$, and the different sandwiches are $s_4 = 1$, $s_{3,3}^{11} = 2$, $s_{3,3}^{12} = 1$, $s_{2,01}^{11} = s_2 = 1$, $s_1^{11} = s_1 = 2$, and $s_{1-x-1}^{11} = 1$ (all the other variables are equal to 0). It follows that

$$\begin{aligned}
|D_5(021202112)| &= |\mathcal{P}_5(8,7,1,0,0)| - s_4 - s_{3,3}^{11}|\mathcal{P}_1(3)| - s_{3,3}^{1\overline{2}}|\mathcal{P}_1(4)| \\
&\quad - s_{2,01}^{11}|\mathcal{P}_2(5,5)| - s_1^{11}|\mathcal{P}_3(5,4,1)| + s_2 s_1 \\
&\quad + \binom{s_1^{11}}{2}|\mathcal{P}_1(2)| - s_{1-x-1}^{11}|\mathcal{P}_1(2)| + s_{1-x-1}^{11}|\mathcal{P}_1(3)| \\
&= 91 - 1 - 6 - 4 - 10 - 28 + 2 + 2 - 2 + 3 \\
&= 47.
\end{aligned}$$

To conclude this section, we mention that the different variables of the closed-form formulae can be computed in a few passes on the string from left to right with a sliding

window varying in size. For moderately long codewords, the closed-form expressions are faster than the algorithm presented in the next section. Unfortunately, it is not feasible to extend Theorem 4.17 for large values of d, because the correction terms become increasingly complex and the size of the formulae grows exponentially with d. Consider simply the number of different sandwiches of length $d/2$; this corresponds to the number of unrestricted partitions of $d/2$, which grows exponentially with d.

4.4 An Efficient Algorithm to Compute the Number of Subsequences

In this section, I present a simple and efficient algorithm to calculate the number of subsequences when deleting symbols from a string of length n. I take advantage of the common prefixes among the subsequences and express $|D_d(u)|$ using a recursion on d and the length of the string. Let $D_{d,q}(u)$ be the set of d-subsequences of u whose last symbol is q, and $|D_{d,q}(u)|$ the number of such subsequences. For example, $D_{2,0}(01120) = \{010, 020, 110, 120\}$.

Theorem 4.20. *Let $p, q \in \Sigma$. If $d < |u|$, then*

$$|D_d(u)| = \sum_{q \in \Sigma} |D_{d,q}(u)|,$$

where

$$|D_{d',p}(u_1 u_2 \dots u_{k-1} q)| = \begin{cases} 1 & \text{if } d' = 0 \text{ and } p = q; \\ 0 & \text{if } d' = 0 \text{ and } p \neq q; \\ |D_{d'}(u_1 u_2 \dots u_{k-1})| & \text{if } 0 < d' < k \text{ and } p = q; \\ |D_{d'-1,p}(u_1 u_2 \dots u_{k-1})| & \text{if } 0 < d' < k \text{ and } p \neq q. \end{cases}$$

Proof. Clearly, the sets $D_{d',q}(u)$ contain only d'-subsequences of u and each subsequence of u belongs to exactly one of the sets, which proves the summation. As for the recursion, the two cases when $d' = 0$ are trivial.

For the case $0 < d' < k$ and $p = q$, let $v_1 v_2 \dots v_{k-d'-1} q$ be a d'-subsequence of $u_1 u_2 \dots u_{k-1} q$. Since the string and its subsequence both have q as their last symbol, it follows that $v_1 v_2 \dots v_{k-d'-1}$ is a d'-subsequence of $u_1 u_2 \dots u_{k-1}$, thus

$$|D_{d',q}(u_1 u_2 \dots u_{k-1} q)| \leq |D_{d'}(u_1 u_2 \dots u_{k-1})|.$$

Table 4.2: Number of subsequences of the string $u = 0222021$ when deleting up to 5 symbols.

d	$u_1 = 0$	$u_2 = 2$	$u_3 = 2$	$u_4 = 2$	$u_5 = 0$	$u_6 = 2$	$u_7 = 1$
0	1 0 0 1	0 0 1 1	0 0 1 1	0 0 1 1	1 0 0 1	0 0 1 1	0 1 0 1
1	 1	1 0 1 2	0 0 2 2	0 0 2 2	2 0 1 3	1 0 3 4	0 4 1 5
2	 0	 1	1 0 1 2	0 0 2 2	2 0 2 4	2 0 4 6	1 6 3 10
3	 0	 0	 1	1 0 1 2	2 0 2 4	2 0 4 6	2 6 4 12
4	 0	 0	 0	 1	1 0 1 2	2 0 2 4	2 4 4 10
5	 0	 0	 0	 0	 1	1 0 1 2	2 2 2 6

The converse of the previous statement gives $|D_{d'}(u_1u_2\dots u_{k-1})| \leq |D_{d',q}(u_1u_2\dots u_{k-1}q)|$, thus both sets have the same cardinality.

For the case $0 < d' < k$ and $p \neq q$, assume that $v = v_1v_2\dots v_{k-d'-1}p$ is a d'-subsequence of $u_1u_2\dots u_{k-1}q$. Since the subsequence and its mother string do not end by the same symbol, q has to be deleted, leaving $d'-1$ symbols to be deleted from $u_1u_2\dots u_{k-1}$. Consequently, v is a $(d'-1)$-subsequence of $u_1u_2\dots u_{k-1}$ whose last symbol is p, thus

$$|D_{d',p}(u_1\dots u_{k-1}q)| \leq |D_{d'-1,p}(u_1\dots u_{k-1})|.$$

Suppose now that $v_1v_2\dots v_{k-d'-1}p$ is a $(d'-1)$-subsequence of $u_1u_2\dots u_{k-1}$ whose last symbol is p. It follows that $v_1v_2\dots v_{k-d'-1}p$ can be obtained by deleting d' symbols from $u_1u_2\dots u_{k-1}q$ (the last symbol and $d'-1$ among the first $k-1$ symbols), thus $|D_{d'-1,p}(u_1u_2\dots u_{k-1})| \leq |D_{d',p}(u_1u_2\dots u_{k-1}q)|$. The two sets have the same cardinality, and the result follows. □

Theorem 4.20 gives a de facto recursive algorithm that can be used to compute the number of subsequences of a string, but it is inefficient because several intermediate results have to be repeatedly calculated. It is much more efficient to store the intermediate calculations in a table and to compute $|D_d(u)|$ using dynamic programming. The table has $n \cdot (d+1)$ cells arranged in $d+1$ rows, corresponding to the number of deletions from 0 to d, and n columns, corresponding to the symbols of u. Each cell (i,j) has $|\Sigma|+1$ subcells: the first $|\Sigma|$ subcells contain the values $|D_{i,q}(u_1u_2\dots u_j)|$ for $q \in \Sigma$, and the

last subcell contains the value $|D_i(u_1u_2\dots u_j)|$. The cells (i,j) with $i > j$ correspond to $d > |u_1u_2\dots u_j|$ and only contain 0 in the last subcell; the cells (i,j) with $i = j$ correspond to $d = |u_1u_2\dots u_j|$ and only contain 1 in the last subcell. The rest of the table is filled column per column from left to right using Theorem 4.20. An example of the algorithm for the string $u = 0222021$ is presented in Table 4.2. Consider, for example, cell $(d = 2, u_7)$. Since $u_7 = 1$, we have

$$\begin{aligned} |D_2(u_1u_2\dots u_7)| &= |D_{2,0}(u_1u_2\dots u_7)| + |D_{2,1}(u_1u_2\dots u_7)| + |D_{2,2}(u_1u_2\dots u_7)| \\ &= |D_{1,0}(u_1u_2\dots u_6)| + |D_2(u_1u_2\dots u_6)| + |D_{1,2}(u_1u_2\dots u_6)| \\ &= 1 + 6 + 3 = 10. \end{aligned}$$

Hence, the string has ten 2-subsequences.

Lemma 4.21. *The space complexity of the algorithm is in*

$$O(\min(d^2|\Sigma|\log r, d(n-d)|\Sigma|\log|\Sigma|))$$

bits, and the time complexity of the algorithm is in

$$O(\min(nd^2|\Sigma|\log r, n(n-d)d|\Sigma|\log|\Sigma|).$$

Proof. From (4.3), the number of d-subsequences of u is in $O(\min(r^d, |\Sigma|^{n-d}))$, thus each element in the table requires $O(\min(d\log r, (n-d)\log|\Sigma|))$ bits of storage space. Furthermore, each nontrivial cell (i,j) where $0 < i < j$ requires the information located in the cells $(i, j-1)$ and $(i-1, j-1)$. Consequently, the table can be computed column per column, only requiring the storage of $O(d)$ cells. Since each cell has $|\Sigma|+1$ entries, the space complexity is in $O(\min(d^2|\Sigma|\log r, d(n-d)|\Sigma|\log|\Sigma|))$ bits.

The algorithm requires a sum of $|\Sigma|$ numbers and $|\Sigma|$ copies per cell. Since all the numbers have $O(\min(d\log r, (n-d)\log|\Sigma|))$ bits and the table has nd cells, the time complexity is in $O(nd|\Sigma|\min(d\log r, (n-d)\log|\Sigma|))$. □

If $|\Sigma|$ and d are constant, then the number of d-subsequences of u can be calculated in $O(n\log n)$ time and using $O(\log n)$ bits of storage space; if $|\Sigma|$ is constant and $d = \alpha n$ where $0 < \alpha < 1$, then the time complexity is in $O(n^3)$ and the space complexity in $O(n^2)$ bits. The algorithm is not efficient when $|D_d(u)|$ is small. However, the strings with a small number of subsequences have very few runs, including very large ones. Since the entries in the table do not change after the $(d+1)$-th symbol of a long run, we can

modify the algorithm accordingly and reduce its time complexity to

$$O(\min((n+dr)d|\Sigma|\log r, (n+dr)(n-d)|\Sigma|\log|\Sigma|).$$

Theorem 4.20 and the corresponding dynamic programming algorithm are quite general and can be modified or extended to obtain other interesting results and simpler proofs of existing results. For example, they can easily be modified to obtain a recurrence relation for the expected number of d-subsequences over all strings of length n. We conclude this section with a simpler proof describing strings whose number of d-subsequences is maximum, which follows directly from the next lemma.

Lemma 4.22. *Let last(p) be the last position where the symbol p appears in the string u, with last(p) = −1 if p does not appear in u. If last(q) > last(p), then* $|D_{d,q}(u)| \geq |D_{d,p}(u)|$.

Proof. We prove the lemma by induction on the cells (i,j) of the table given by the dynamic programming algorithm used to compute $|D_d(u)|$. Clearly, the lemma holds for the cells $(0,j)$ and the cells (i,j) where $i \geq j$. Assume that $0 < i < j$ and that the result holds for the cells $(i-1, j-1)$ and $(i, j-1)$; we need to prove that the lemma also holds for the cell (i,j). From the inductive hypothesis and the fact that $|D_{i,p}(u_1u_2\ldots u_j)| = |D_{i-1,p}(u_1u_2\ldots u_{j-1})|$ for all $p \neq u_j$ from Theorem 4.20, the lemma holds for all the corresponding values in cell (i,j).

Since $\text{last}(u_j) = j$, the last thing to prove is that $|D_{i,u_j}(u_1u_2\ldots u_j)| \geq |D_{i,p}(u_1\ldots u_j)|$ for every $p \in \Sigma$. $|D_{i,u_j}(u_1u_2\ldots u_j)| = |D_i(u_1u_2\ldots u_{j-1})|$ from Theorem 4.20, and $|D_i(u_1u_2\ldots u_{j-1})|$ is greater than or equal to $\max_{p\in\Sigma}(|D_{i-1,p}(u_1u_2\ldots u_{j-1})|)$, since for each substring in $D_{i-1,p}(u_1u_2\ldots u_{j-1})$, the deletion of the last symbol gives a substring in $D_i(u_1u_2\ldots u_{j-1})$. □

From Lemma 4.22, if one wants to maximize the number of descendants of a string, then identical symbols should be as far apart as possible.

Corollary 4.23 (Hirschberg and Régnier [47]). *Let u be a string of length n.* $|D_d(u)|$ *is maximized for* $u = \sigma\sigma\ldots\sigma$, *where* σ *is the concatenation of all the symbols of* Σ *in any order.*

4.5 An Efficient Algorithm to Compute the Multiplicity of a Subsequence in a String

Using a simplified version of the technique used in the previous section, it is possible to give a recursive definition for the multiplicity of a subsequence in a string.

Lemma 4.24.

$$M(u_1 \dots u_k, v_1 \dots v_l) = \begin{cases} 0 & \text{if } k > l; \\ 1 & \text{if } k = 0; \\ M(u_1 \dots u_k, v_1 \dots v_{l-1}) & \text{if } 0 < k \leq l \text{ and } u_k \neq v_l \\ M(u_1 \dots u_{k-1}, v_1 \dots v_{l-1}) & \\ \quad + M(u_1 \dots u_k, v_1 \dots v_{l-1}) & \text{if } 0 < k \leq l \text{ and } u_k = v_l. \end{cases}$$

Proof. Let $u \triangleq u_1 \dots u_k$ and $v \triangleq v_1 \dots v_l$. The case $k > l$ is trivial and the case $k = 0$ comes from the definition. For the third case, it is sufficient to see that since $u_k \neq v_l$, all the bits u must be taken from the first $l-1$ bits of v, thus the multiplicity of u in v is equal to the multiplicity of u in $v_1 \dots v_{l-1}$. For the last case, all the instances of u in v can be divided into two distinct sets: the first set includes the instances for which the last bit is v_l, and the second set the instances for which the last bit is $v_{l'}$, where $l' < l$. For all the instances of u in the first set, the first $k-1$ bits are taken from $v_1 \dots v_{l-1}$, hence the cardinality of the set is the multiplicity of $u_1 u_2 \dots u_{k-1}$ in $v_1 v_2 \dots v_{l-1}$. The instances of u in the second set are equivalent to the third case, and the result follows. □

Again, simple recursive and dynamic programming algorithms can be derived from Lemma 4.24. Table 4.3 illustrates the dynamic-programming algorithm and shows that the subsequence $u = 0010$ appears 19 times in the string $v = 000100110$. An interesting feature of the algorithm is that it calculates $M(u_1 \dots u_{k'}, v_1 \dots v_{l'})$ for $k' \leq k$ and $l' \leq l$ on its way to $M(u_1 \dots u_k, v_1 \dots v_l)$. This will turn out to be very useful when studying maximum-likelihood decoding of synchronization error-correcting codes in Chapter 6.

4.6 Summary

In this chapter, I have studied the number of subsequences when deleting d symbols from a string. I have defined the concepts of deletion pattern and sandwich and used them to construct a framework for finding closed-form expressions for the number of

Table 4.3: Multiplicity of the subsequence $u = 0010$ in the string $v = 00010010$.

	ϵ	$v_1 = 0$	$v_2 = 0$	$v_3 = 0$	$v_4 = 1$	$v_5 = 0$	$v_6 = 0$	$v_7 = 1$	$v_8 = 0$
ϵ	1	1	1	1	1	1	1	1	1
$u_1 = 0$	0	1	2	3	3	4	5	5	6
$u_2 = 0$	0	0	1	3	3	6	10	10	15
$u_3 = 1$	0	0	0	0	3	3	3	13	13
$u_4 = 0$	0	0	0	0	0	3	6	6	19

subsequences. I have presented the first closed-form expressions for nonbinary alphabets and the first closed-form expressions for $d \in \{3, 4, 5\}$. Unfortunately, due to the size of the closed-form expressions, it is not feasible to extend the framework for very large values of d. Nevertheless, I have proved a simple and efficient algorithm to numerically compute the number of subsequences. The algorithm works for any string, number of deletions and alphabet size. I have also presented a simple algorithm to calculate the multiplicity of a subsequence in a string.

Part II

Synchronization Error-Correcting Codes

Chapter 5

Variable-Length Synchronization Error-Correcting Block Codes

In this short chapter, I describe Levenshtein's well-known single-synchronization block codes and present a simple extension for multiple insertion and deletion errors.

5.1 Levenshtein's Single-Synchronization Error-Correcting Codes

Error-correcting codes for channels with synchronization errors were first studied by Levenshtein [60], who gave bounds on the size of codebooks able to correct a constant number of synchronization errors per block. Levenshtein also explained how to maintain synchronization during long transmissions by inserting synchronization bits between blocks, and studied the equivalent of the Z-channel for synchronization errors [59].

Levenshtein [60], using a result from Varshamov and Tenengolts [106], provided a very nice construction and decoding algorithm for single-synchronization error-correcting codes of length n. Consider all the binary codewords of length n such that

$$\sum_{i=1}^{n} i x_i \equiv 0 \pmod{n+1};$$

this set forms a codebook C_n capable of correcting one synchronization error. For example, $C_5 = \{00000, 10001, 01010, 11011, 00111, 11100\}$. The minimum distance of the codebooks C_n is 2 (0^n and $10^{n-2}1$ are always members of C_n), and as such they can detect a substitution error but cannot correct it. Let $|C_n|$ be the size of the codebook

C_n; the first few terms of the sequence $\{|C_n|\}$ for $n \geq 1$ are

$$1, 2, 2, 4, 6, 10, 16, 30, \ldots$$

Combining results from Levenshtein and Varshamov [105], it is possible to show that

$$\begin{aligned} |C_{n-1}| &= \frac{1}{2n} \sum_{\text{odd } d|n} \phi(d) 2^{\frac{n}{d}} \\ \Rightarrow |C_n| &\sim \frac{2^n}{n}, \end{aligned} \tag{5.1}$$

where $\phi(n)$, Euler's totient function, is the number of positive integers less than or equal to n that are relatively prime to n, and $f(n) \sim g(n)$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$. Hence, the rate of these codes is asymptotically optimal.

Levenshtein's codes were subsequently studied by several researchers. In a survey of single-deletion-correcting codes, Sloane [98] pointed out that several interesting questions remain unanswered, the main ones in my opinion being the possible connection between the maximum size of single-deletion-correcting codebooks and the number of output sequences from the complemented cycling shift register, as well as the difference between the size of the largest codebooks and the size of the largest linear codebooks. Unfortunately, it does not seem that Levenshtein's construction can be generalized to correct multiple synchronization errors in an asymptotically optimal way. Helberg and Ferreira [46] extended Levenstein's construction for multiple insertion and deletion errors, but the loss in rate is significant, they do not provide efficient encoding and decoding algorithms, and their technique does not scale well for large block sizes.

Perfect deletion-correcting-codes were also defined by Levenshtein [61]. A e-deletion-correcting code of length n is perfect if the e-subsequences of all the codewords partition the set $\{0,1\}^{n-e}$.

Result 5.1 (Levenshtein [61]). *For every $n \geq 1$, the single-deletion-correcting codebook C_n is perfect.*

Following Levenshtein, Mahmoodi [65] showed that for every v there exists a perfect 3-deletion-correcting code with codebook of size v. In fact, several papers [8, 65, 66, 35, 113, 92] studied perfect deletion-correcting-codes using design theory, but all the constructions are mostly derived using ad hoc techniques and without efficient encoding or decoding algorithms.

As mentioned in the previous chapters, the balls associated with synchronization codewords do not all have the same size, thus upper bounds for codes correcting synchronization errors are much more difficult to derive than upper bounds for codes over the erasure and binary symmetric channels. For instance, the maximal size of codebooks correcting more than 1 deletion of bit is not known, even for very small block lengths [99]. In theory, this question can be answered by computing the size of the largest independent set in the characteristic graph for this problem. Unfortunately, it was proved [44] that if $\mathcal{NP} \not\subseteq \mathcal{BPP}$ [84], then for every $\epsilon > 0$, it is impossible to estimate the size of the largest independent set in a graph of n vertices in polynomial time, even within a factor $n^{1-\epsilon}$ from the answer.

5.2 Repetition Synchronization Error-Correcting Codes

In this section, I present a simple extension of Levenshtein's single-synchronization codes to correct more than one error by increasing the length of the runs of the original single-synchronization codewords.

Lemma 5.2. *The total number of runs for* $\{0,1\}^n$ *is* $(n+1)2^{n-1}$.

Proof. This can be easily computed by solving the recursion

$$a_n = \begin{cases} 2 & \text{if } n = 2; \\ a_{n-1} + 2^{n-1} & \text{if } n > 2. \end{cases}$$

□

Lemma 5.3. *The total number of runs for all the codewords in the codebook* C_n *is* 2^{n-1}.

Proof. First, from Result 5.1, the total number of 1-subsequences for all the codewords of C_n is 2^{n-1}. Second, from (4.1), the number of subsequences when deleting one symbol from a string corresponds to its number of runs. Hence, the number of runs in C_n is 2^{n-1}. □

There are many interesting similarities between sequences originating from insertion-deletion error-correcting codes and other famous integer sequences. For instance, let $p(n)$ be the number of partitions of n into distinct parts; the first few terms of the sequence

Table 5.1: Extensions of C_5 for two synchronization errors.

	C_5	C_5^a	C_5^b	C_5^c	C_5^d	C_5^e
codebook	00000	0000000000	000000	000000	000000	0000
	10001	1100000011	11000011	11000011	110000011	110000011
	01010	0011001100	0011001100	0011001100	0011001100	0011001100
	11011	1111001111	11100111		11100111	11100111
	11100	1111110000	1111000	1111000	1111000	1111000
	00111	0000111111	0001111	0001111	0001111	0001111
Cap.	1 sync.	2 sync.	1 sync.	2 del.	2 del.	2 sync.
Rate	0.517	0.258	.337	.306	.330	.345

$P = \{p(n)\}$ for $n \geq 0$ are $1, 1, 1, 2, 2, 3, 4, 5, 6, \ldots$. Furthermore, we define the sequence $Q = \{q(n)\}$, where $q(n) = \sum_{m=0}^{n} p(m)$. In other words, the sequence Q corresponds to the partial sums of the sequence P, and its first few terms are $1, 2, 3, 5, 7, 10, 14, 19, 25, \ldots$. One can see that the number of infinite sequences whose weighted sum is n is $p(n)$, and that the total number of runs for the strings whose weighted sum is n is $q(n)$. Another interesting similarity is that the recurrence from Lemma 5.2 corresponds to the number of states of the briefcase planning domain, a famous problem first introduced by Pednault [85]. If is unclear whether or not this is more than a coincidence.

By duplicating each symbol k times, an e-synchronization-correcting code with codewords of length n becomes an $(e + k)$-synchronization-correcting code with codewords of length $n \cdot (k + 1)$. A more promising approach, instead of duplicating each symbol k times, is to add k bits in each run. Using this technique in Levenshtein's original codes, it follows from (5.1) and Lemma 5.3 that the codebook C_n can be transformed into a codebook of average length

$$\sim n + \frac{k \cdot 2^{n-1} \cdot n}{2^n} = n\left(1 + \frac{k}{2}\right).$$

Although the resulting codebooks are not necessarily able to correct $k + 1$ synchronization errors, they can easily be modified to do so, either by removing codewords and by increasing or decreasing the length of appropriate runs. This simple technique to increase the error-correction capability of Levenshtein's codes is far from optimal, although it is better than several codes recently presented [46, 13]. Interestingly, the resulting codes are variable-length codes, and as a result they can be more efficient against one type of synchronization error (either insertions or deletions) than they are when insertion and deletion errors can occur together. This will be discussed in more detail in Chapter 6.

Table 5.2: Double-deletion-correcting codebooks obtained by increasing the length of the runs of Levenshtein's codewords.

C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
000	0000	00000	000000	0000000	00000000	000000000	0000000000
111	110011	0011100	0001111	0001100111	00110011111	00000111100	00000011111
		1100011	0011001100	00110001100	001110011100	000111100111	00000110011100
		11111	11100111	001111100	00111100011	001100111001100	0000011100011
			1111000	11001110011	1100000011	0011100011100	0001100000111
				111000111	11000111100	00111001100011	000110011110011
				1110011000	11001100110011	00111100000	00011110001100
				1111111	1111001111	00111111100	00110000001100
					11111001100	11000000011	00110001111000
						11000011111	001100110001111
						110011000110011	001100110011001100
						1110011100111	00111001100000
						111001111000	0011100111111
						11110011001100	001111100110011
						11111000011	001111110000
						111111111	11000011001111
							1100011100000
							110001111111
							110011001111100
							110011110011000
							111000000111
							1110000011000
							11100011110011
							1110011001100111
							111100011001100
							11110011000011
							11111000000
							111110011111
							1111110011100
							111111100011

Table 5.3: Codebook size versus average block length for double-deletion-correcting codes.

block length	6	7	7.6	8	9	9.25	10	10.9	11	12	13	13.5	14
increased runs	4		5			8		9		16		30	
maximum size	4	5		7	11		16		≥ 24	?	?		?
HF2002 [46]	3	4		5	6		8		9	11	15		18

An illustration of the extensions proposed above for the codebook C_5 is presented in Table 5.1. The double-synchronization-correcting codebook C_5^a is obtained by duplicating every bit of C_5. Since the reduction in rate from C_5 to C_5^a is prohibitive, C_5^b is instead obtained by adding one bit in each run of the codewords of C_5. C_5^b cannot correct two deletion errors because both 11000011 and 11100111 have the 2-subsequence 110011; this can be solved either by removing one of the two codewords from the codebook, as in C_5^c, or by adding a single bit in the middle run of 11000011, as in C_5^d. Neither C_5^c nor C_5^d can correct two synchronization errors, because 00110000 can be obtained by deleting two bits from 0011001100 and by inserting two bits in 000000. However, this can be solved by removing two bits from 000000, and the resulting code C_5^e can correct two synchronization errors. It should be noted that the rate of C_5^e is one third higher than the rate of C_5^a.

Double-deletion-correcting codebooks obtained by extending C_n for $2 \leq n \leq 9$ are included in Table 5.2. One bit was added in each run, and as it was done for C_5^c, the codewords too close from each other were removed from the codebook. Note that codes with higher rates could be found by adding or deleting bits in some of the runs instead of discarding entire codewords. In Table 5.3, the size of the resulting codebooks is compared to the maximum size of double-deletion-correcting codebooks [99] and to the double-synchronization-correcting codes of Helberg and Ferreira [46] with comparable block lengths. It should be mentioned that the size of the largest double-deletion codebook with codewords of length 11 is at least 24, but the exact value is unknown. The largest double-deletion codebooks with codewords of length greater than 11 are also unknown.

5.3 Summary

In this chapter, I have presented a technique to construct reasonably good variable-length $(k+1)$-synchronization error-correcting codes. Codebooks are first preprocessed by adding k bits in each run of Levenshtein's single-correcting codewords. The resulting

codebooks cannot correct $k+1$ synchronization errors, but due to the good properties of Levenshtein's codes, this can be done either by removing codewords or by varying the length of the problematic runs. The number of codewords in Levenshtein's codebooks increases exponentially with the block length, and as an unfortunate consequence the codebook derivation cannot be done for very large block lengths.

The codes constructed in this chapter outperform several codes previously published but are still far from optimal. Although increasing the length of the runs could also improve the correction capability of any code robust against synchronization errors, it seems that designing more efficient multiple-synchronization error-correcting codes will have to be done without extending Levenshtein's single-synchronization codes. This certainly is a challenging task.

Chapter 6

Nonlinear Trellis Codes for Symbol Synchronization Errors

6.1 Introduction

In modern communication systems, additive noise and synchronization are treated as different problems and overcome using different techniques. This being said, both have the same effect on communication channels, i.e., reducing their capacity. Unfortunately, although it has often been conjectured that error-correcting codes capable of correcting timing errors could improve the overall performance of communication systems, they are extremely challenging, which partly explains why a large collection of synchronization techniques not based on coding were developed and implemented over the years.

One of the reasons why coding is difficult is that channels with timing errors have memory, hence the techniques developed for memoryless channels and additive noise can seldom be used. Besides the work pioneered by Levenshtein and discussed in the previous chapter, there are few mathematical tools that can be used for code design, and as a result the vast majority of known codes capable of correcting more than one synchronization error per block have no chance of being of any practical use. For instance, Schulman and Zuckerman [89] showed the existence of "simple, polynomial-time encodable and decodable codes which are asymptotically good for channels allowing insertions, deletions and transpositions", although their constructions are far from explicit and seem to be only of theoretical interest. They use a concatenated scheme consisting of a Reed-Solomon outer code and an inner code found by a greedy algorithm. Another major challenge of synchronization error-correcting codes is that the boundaries of the blocks might be

unknown to the receiver, thus symbol and word synchronization must be considered.

Several researchers have juggled with ways to use coding in one form or another to protect communication systems against synchronization errors, for instance using synchronizable codes like comma-free codes [18, 41], prefix codes [38], and codes with bounded synchronization delay [40]. Synchronizable codes are designed to recover synchronization efficiently and sometimes to correct a few substitution errors, but not to correct synchronization errors per se. An interesting survey of synchronizable codes was written by Cummings [19]. Liu and Mitzenmacher [64] recently modified Levenshtein's single-deletion codes so they can be used in a concatenated fashion. Their construction is very similar to comma-free codes, and the decoding works as long as there is at most one error per block, this limit again coming from the fact that extending Levenshtein's construction for more that one error seems very difficult.

A technique to detect and correct synchronization errors is to insert periodic markers in the codewords. Marker codes were first introduced by Sellers [90], who used the marker 001. Sellers's construction can correct one insertion or deletion error between successive markers. Ferreira et al. [34] used markers based on Levenshtein's codes [60] that can recover synchronization if there is at most one substitution or synchronization error per block, where a block contains the bits between consecutive markers plus a marker. Their algorithm can correct synchronization and substitution errors if there is one of them per two blocks. Markers can also be used to avoid synchronization errors from propagating from block to block, which can happen when the decoder does not know the exact block boundaries [14]. In a remarkable paper, Davey and Mackay [20] designed a concatenated scheme using a nonlinear inner code named *watermark code* and a low-density parity-check outer code over a nonbinary field. The output of the LDPC decoder is mapped into a sparse binary string, which is then added modulo 2 to the a watermark vector known to the transmitter and receiver. The watermark can be seen as a marker uniformly distributed throughout the codeword and is used to recover synchronization. One of the codes they presented has a rate of 0.7 and can correct 30 synchronization errors per block of length 5000. Davey and Mackay's approach was further studied by Ratzer [86], who showed that irregular watermarks gave better performance at very low error rates.

A (d, k)-constrained runlength-limited code is a code such that there is at least d zeros and and most k zeros between consecutive ones. Runlength-limited codes, commonly used in magnetic and optical recording systems [48], were also studied for channels affected by synchronization errors. Roth and Siegel [87] designed constrained BCH codes based on the Lee metric [58] able to correct bitshifts as well as insertions and deletions of zeros

(a bitshift can be seen as the deletion of a zero followed by an insertion of a zero on the other side of a 1). Their framework was extended by Bours [9], who obtained codes capable of correcting a larger number of insertions and deletions of zeros. Similarly, Levenshtein considered codes able to correct insertions and deletions of ones [59]. In these settings, synchronization cannot be completely lost because the receiver knows the boundaries between the runs. In fact, there is a one-to-one correspondence between error patterns from a binary channel with bitshift errors and deletion of zeros, and error patterns from a binary channel where each run is transformed into a run of nonzero bits, i.e., channels with duplication and deletion errors with the additional assumption that no run is completely deleted. This "light synchronization" does not occur if runs can be completely deleted or if random insertions can occur, creating new runs.

Another approach is to adapt convolutional codes for synchronization error-correction. In 1961, Gallager [37] suggested to add pseudo-random sequences to the output of convolutional encoders and to correct synchronization errors using a sequential decoding algorithm. Gallager's main motivation was that codes capable of correcting insertions and deletions of symbols could likely increase the throughput of synchronous systems by reducing the number of required synchronization pulses. One should note that Gallager's work precedes Viterbi's famous decoding algorithm [107]. A practical technique used to facilitate block synchronization of convolutional codes is to invert alternate symbols at the output of the encoder to avoid long blocks of 0s and 1s, with the assumption that long runs of 0s and 1s are more frequent than long sequences of alternate bits. Baumert, McEliece, and van Tilborg [7] studied which convolutional encoders can output infinitely many consecutive alternate bits, and for the other codes provided upper bounds on the maximum length of such sequences. Using a similar idea, Swart and Ferreira [101] modified convolutional encoders to correct a small number of insertions and deletions of bits by inverting some of the output bits and eliminating large runs of 0s and 1s. They designed codes of rate $\frac{1}{4}$ and $\frac{1}{3}$ that can correct one deletion error every 8 and 12 bits, respectively. Mori and Imai [76] used convolutional encoders and a metric based on the Levenshtein distance for the Viterbi decoding algorithm, but they only considered very small insertion and deletion error rates and made the false assumption that synchronization errors can only affect small blocks of bits in the final codewords. Swart, Ferreira, and dos Santos [103] used convolutional codes and parallel Viterbi decoders to correct a small number of insertions and deletions of bits.

In this chapter, I present an extensive study of convolutional codes for synchronization errors. I discuss how to modify the Viterbi decoding algorithm to correct several types

of synchronization errors, and formally prove that the algorithm can find the "closest" codeword from the received sequence, which, surprisingly, had never been done before. The computational complexity of the modified versions of the algorithm depends on the type of errors it is correcting: it is roughly the same as the complexity of the original Viterbi algorithm for insertion and deletion errors, but much larger if insertion and deletion errors can occur together. I also explain how to obtain maximum-likelihood codewords using improved versions of the algorithms and formally prove that this cannot be done without increasing their complexity.

Intuitively, it is a good idea to consider convolutional codes to protect communications against time noise, mainly because convolutional encoders can compensate for our lack of understanding of the behavior of synchronization errors. However, I prove that the performance synchronization-correcting codes largely depends on the structure of the underlying encoder graphs and that convolutional codes are fundamentally ill-suited to correct synchronization errors. It explains why the researchers who previously tried this approach obtained mitigated results. I prove that there is a tradeoff between the number of synchronization errors a code can correct and the capacity it has to recover synchronization, and present a family of rectangular graphs very robust against synchronization errors. For instance, I present a code with four states that can correct one deletion error every seven bits. It uses an almost trivial rectangular graph but nevertheless surpasses all the codes discussed above, and this convinces me that trellis codes are a promising approach to correct synchronization errors.

The rest of the chapter is organized as follows. In Section 6.2, I introduce the distances required to study synchronization error-correcting codes. In Section 6.3, I prove that the Viterbi decoding algorithm can be modified to correct several types of synchronization errors. A family of rectangular graphs well suited to correct synchronization errors when used as encoders is presented in Section 6.4. In Section 6.5, I discuss the tradeoff between recovering synchronization and correcting synchronization errors, and a few simulation results for trellis codes based on simple rectangular graphs are discussed in Section 6.6. In Section 6.7, I discuss ways to obtain maximum-likelihood decoding for the deletion channel, and I conclude the chapter in Section 6.8.

6.2 Preliminaries and Useful Distances

In this section, I introduce the notation and all the distances required throughout this chapter. Let $\mathbf{u} \triangleq u_1 u_2 \dots u_n$ be a Q-ary string of length n and $|\mathbf{u}|$ the length of $\mathbf{u}$. We

use ϵ to denote the *empty string* of length 0, $\mathbf{u}[k..l]$ for the substring $u_k u_{k+1} \dots u_l$, and $\mathbf{u} \circ \mathbf{v}$ for the concatenation of the strings $\mathbf{u}$ and $\mathbf{v}$. A *run* is a substring of identical symbols of maximum length in $\mathbf{u}$. A *d-subsequence* of $\mathbf{u}$ is a string of $n - d$ symbols that can be obtained by deleting d symbols from $\mathbf{u}$, and a *d-supersequence* of $\mathbf{u}$ is a string of $n + d$ symbols that can be obtained by inserting d symbols in $\mathbf{u}$. A *prefix* of $\mathbf{u}$ is a string that can be obtained by deleting symbols at the end of $\mathbf{u}$. A string $\mathbf{v}$ is *deletion-reducible* to a string $\mathbf{u}$ if $\mathbf{u}$ is a subsequence of $\mathbf{v}$. A string $\mathbf{v}$ is *insertion-reducible* to $\mathbf{u}$ if $\mathbf{u}$ supersequence of $\mathbf{v}$, or alternatively if $\mathbf{u}$ is deletion-reducible to $\mathbf{v}$. For example, the string $\mathbf{u} = 00101$ has one run of length two followed by three runs of length one, 000 is a 2-subsequence of $\mathbf{u}$, 00110011 is a 3-supersequence of $\mathbf{u}$, 001 is a prefix of $\mathbf{u}$, and 011001 is insertion-reducible to $\mathbf{u}$. Finally, as defined in the previous chapters, a synchronization error is either a symbol deletion or the insertion of a random symbol, and a duplication error is the insertion of a bit similar to the bit preceding it.

Definition 6.1. The *insertion-deletion distance* (or *synchronization distance*) between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $d_s(\mathbf{u}, \mathbf{v})$, is the smallest number of insertions and deletions of symbols required to change $\mathbf{u}$ into $\mathbf{v}$. The *minimum insertion-deletion distance* (or *minimum synchronization distance*) of a code is the smallest insertion-deletion distance between any two of its codewords.

The synchronization distance has been studied a lot in the literature and is well understood. It should be clear that it is a metric and that a code with minimum insertion-deletion distance $2e + 1$ can detect up to $2e$ insertions or deletion errors and correct e synchronization errors.

Definition 6.2. The *deletion distance* between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $d_d(\mathbf{u}, \mathbf{v})$, is the minimum number of symbols that have to be deleted from the longest string in order to become a subsequence of the shortest string. If the two strings have the same length, then the deletion distance can be calculated by deleting symbols from any of the two strings. For example, $d_d(10001, 011) = 3$ and $d_d(01110, 11011) = 2$. The *minimum deletion distance* of a code is the smallest deletion distance between any pair of codewords. The *insertion distance* between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $d_i(\mathbf{u}, \mathbf{v})$, is the minimum number of symbols that have to be inserted in the shortest string in order to become a supersequence of the longest string. If the two strings have the same length, then the insertion distance can be calculated by inserting symbols in any of the two strings. The *minimum insertion distance* of a code is the smallest insertion distance between any two of its codewords.

The synchronization, deletion and insertion distances have the following properties.

1. $d_d(\mathbf{u},\mathbf{v}) \geq ||\mathbf{u}| - |\mathbf{v}||$.
   $d_i(\mathbf{u},\mathbf{v}) \geq ||\mathbf{u}| - |\mathbf{v}||$.

2. The deletion and insertion distances are metrics, i.e.,

   (a) $d_d(\mathbf{u},\mathbf{u}) = d_i(\mathbf{u},\mathbf{u}) = 0$;

   (b) $d_d(\mathbf{u},\mathbf{v}) = d_d(\mathbf{v},\mathbf{u})$;
   $d_i(\mathbf{u},\mathbf{v}) = d_i(\mathbf{v},\mathbf{u})$;

   (c) $d_d(\mathbf{u},\mathbf{v}) \leq d_d(\mathbf{u},\mathbf{a}) + d_d(\mathbf{a},\mathbf{v})$;
   $d_i(\mathbf{u},\mathbf{v}) \leq d_i(\mathbf{u},\mathbf{a}) + d_i(\mathbf{a},\mathbf{v})$.

3. $d_d(\mathbf{u},\mathbf{v}) = d_i(\mathbf{u},\mathbf{v})$ for every pair of strings $\mathbf{u}$ and $\mathbf{v}$.

4. $d_s(\mathbf{u},\mathbf{v}) = 2d_d(\mathbf{u},\mathbf{v}) - ||\mathbf{u}| - |\mathbf{v}||$;
   $d_s(\mathbf{u},\mathbf{v}) = 2d_i(\mathbf{u},\mathbf{v}) - ||\mathbf{u}| - |\mathbf{v}||$;

Proof. The properties 1, 2(a) and 2(b) are trivial. We prove 2(c) for the deletion distance. Let $|\mathbf{u}| \leq |\mathbf{a}| \leq |\mathbf{v}|$. Since there is a subsequence $\mathbf{a}'$ of $\mathbf{a}$ that can be obtained by deleting $d_d(\mathbf{a},\mathbf{v})$ symbols from $\mathbf{v}$, it is clear that it is possible to obtain a subsequence $\mathbf{u}'$ of $\mathbf{u}$ by deleting at most $d_d(\mathbf{u},\mathbf{a}') \leq d_d(\mathbf{u},\mathbf{a})$ symbols from $\mathbf{a}'$, and that $\mathbf{u}'$ is a subsequence of $\mathbf{v}$. The cases $|\mathbf{a}| \leq |\mathbf{u}| \leq |\mathbf{v}|$ and $|\mathbf{u}| \leq |\mathbf{v}| \leq |\mathbf{a}|$ for the deletion distance and the three cases for the insertion distance are proved similarly.

To prove the third property, we prove that $d_d(\mathbf{u},\mathbf{v}) \geq d_i(\mathbf{u},\mathbf{v})$. Suppose, without loss of generality, that $|\mathbf{u}| \leq |\mathbf{v}|$, and that $\mathbf{u}'$ is a subsequence of $\mathbf{u}$ obtained by deleting $d_d(\mathbf{u},\mathbf{v})$ symbols from $\mathbf{v}$. It is clear that we get a supersequence $\mathbf{v}'$ of $\mathbf{u}$ by inserting the $d_d(\mathbf{u},\mathbf{v})$ symbols that were deleted from $\mathbf{v}$ in the corresponding positions in $\mathbf{u}$. The case $d_d(\mathbf{u},\mathbf{v}) \leq d_i(\mathbf{u},\mathbf{v})$ is done similarly.

To prove the last property, we again suppose that $|\mathbf{u}| \leq |\mathbf{v}|$ and that $\mathbf{u}'$ is a subsequence of $\mathbf{u}$ obtained by deleting $d_d(\mathbf{u},\mathbf{v}) = k$ symbols from $\mathbf{v}$. If follows that $\mathbf{u}'$ can be changed into $\mathbf{u}$ by inserting $|\mathbf{u}| - |\mathbf{u}'| = |\mathbf{u}| - (|\mathbf{v}| - k)$ symbols in $\mathbf{u}'$, thus $d_s(\mathbf{u},\mathbf{v}) \leq k + |\mathbf{u}| - (|\mathbf{v}| - k) = 2d_d(\mathbf{u},\mathbf{v}) - ||\mathbf{u}| - |\mathbf{v}||$. Suppose now that $d_s(\mathbf{u},\mathbf{v}) = k$ with $|u| \leq |v|$. It means that there exists a string $\mathbf{u}'$ such that $\mathbf{v}$ can be changed into $\mathbf{u}'$ with $|\mathbf{v}| - |\mathbf{u}| + \frac{k-(|\mathbf{v}|-|\mathbf{u}|)}{2}$ deletions and $\mathbf{u}'$ can be changed into $\mathbf{u}$ with $\frac{k-(|\mathbf{v}|-|\mathbf{u}|)}{2}$ insertions. □

One can note that the deletion and insertion distances are not metrics if the condition on the length of $\mathbf{u}$ and $\mathbf{v}$ is removed. The reason to compute the distances by deleting symbols from the longest sequence and by inserting symbols in the shortest sequence will become obvious after the next lemma.

Lemma 6.3. *Let C be a code whose minimum deletion (insertion) distance is e. If a codeword is a subsequence of another codeword, then the code can detect at least $e-1$ deletion (insertion) errors, otherwise it can detect all the deletion (insertion) error patterns. In both cases, C can correct $e-1$ deletion (insertion) errors.*

Proof. We prove the result for deletion errors. The only way for a deletion pattern to remain undetected is if a codeword is transformed into another codeword. This is only possible if there is at least one codeword which is a subsequence of another codeword, and the maximum number of deletions that can be detected in this case is one subtracted from the difference between the length of two such codewords, which is lower bounded by $e-1$.

For the error-correction capability of the code, suppose that the received sequence $\mathbf{u}'$ is obtained by deleting at most $e-1$ symbols from a codeword $\mathbf{u}$. Since the minimum deletion distance of the code is e, $\mathbf{u}'$ is not a subsequence of any of the codewords whose length is smaller than or equal to the length of $\mathbf{u}$. Furthermore, although $\mathbf{u}'$ can be a subsequence of some of the codewords longer than $\mathbf{u}$, at least e symbols must be deleted from them to get $\mathbf{u}'$. Hence, $\mathbf{u}$ is the only codeword that can be obtained by inserting at most $e-1$ symbols in $\mathbf{u}'$, and the code can correct up to $e-1$ deletion errors. □

A direct consequence of the lemma is that when all its codewords have the same length, the minimum deletion distance of a code is exactly half the minimum synchronization distance, and as a result it can correct as many synchronization errors as deletion or insertion errors. At the other end of the spectrum, if for any pair of codewords one is a subsequence of the other, then the synchronization and deletion minimum distances are equal, thus the code can correct roughly half as many synchronization errors as deletion (or insertion) errors. More precisely, if the deletion and synchronization distances are e, then the code can correct $e-1$ deletions and $\lfloor \frac{e-1}{2} \rfloor$ synchronization errors. Consider the code $\{0000, 00000000, 000000000000, 0000000000000000\}$. It can correct three deletion or insertion errors, but only one if insertions and deletions can occur together. This discrepancy is caused by the asymmetric nature of insertions and deletions: a received sequence like 00000 cannot be obtained by deleting bits from the codeword 0000 even though both strings are very close to each other. This explains why, in Chapter 5, it was

more difficult to extend Levenshtein's single-synchronization codes to correct multiple synchronization errors than to correct multiple deletions.

Definition 6.4. The *prefix deletion distance* between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $pr_d(\mathbf{u}, \mathbf{v})$, is the minimum number of symbols that have to be deleted from $\mathbf{v}$ to obtain a prefix of $\mathbf{u}$. The *prefix insertion distance* between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $pr_i(\mathbf{u}, \mathbf{v})$, is the minimum number number of symbols that have to be inserted in $\mathbf{v}$ to obtain a prefix of $\mathbf{u}$. If $\mathbf{v}$ cannot be a prefix of $\mathbf{u}$ by inserting bits in it, then $pr_i(\mathbf{u}, \mathbf{v}) = \infty$. Finally, the *prefix duplication distance* between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $pr_t(\mathbf{u}, \mathbf{v})$, is the minimum number of symbols by which the runs of $\mathbf{v}$ must be extended to obtain a prefix of $\mathbf{u}$. If $\mathbf{v}$ cannot be a prefix of $\mathbf{u}$ by increasing the length of its runs, then $pr_t(\mathbf{u}, \mathbf{v}) = \infty$, and it is not hard to show that for binary alphabets, $pr_t(\mathbf{u}, \mathbf{v}) < \infty$ if and only if both strings begin with the same bit and all the runs of one of the strings are smaller than or equal to the corresponding runs of the other string. For example, $pr_d(010, 111) = 3$, $pr_d(111, 010) = 2$, $pr_i(0010010, 01) = 1$, $pr_i(111000, 01) = \infty$, $pr_t(000111, 011) = 2$, and $pr_t(000111, 1) = \infty$.

If the first k bits of $\mathbf{u}$ can be obtained by deleting l bits from $\mathbf{v}$, then for $0 \leq k' \leq k$ the first $k-k'$ bits of $\mathbf{u}$ can be obtained by deleting $l+k'$ bits from $\mathbf{v}$. A similar statement holds for insertion and duplication errors, however, when deletion and insertion errors can occur together, more precise prefix distances are required.

Definition 6.5. The *k-prefix synchronization distance* between two strings $\mathbf{u}$ and $\mathbf{v}$, noted $pr_s^k(\mathbf{u}, \mathbf{v})$ is the minimum number of synchronization errors required to transform $\mathbf{v}$ into the first k bits of $\mathbf{u}$. It should be clear that the $|\mathbf{u}|$-prefix synchronization distance between $\mathbf{u}$ and $\mathbf{v}$ is also the synchronization distance between $\mathbf{u}$ and $\mathbf{v}$. The *prefix synchronization distance* between strings $\mathbf{u}$ and $\mathbf{v}$ is the tuple $pr_s(\mathbf{u}, \mathbf{v}) \triangleq (pr_s^0(\mathbf{u}, \mathbf{v}), pr_s^1(\mathbf{u}, \mathbf{v}), \ldots, pr_s^{|\mathbf{u}|}(\mathbf{u}, \mathbf{v}))$ of the k-prefix synchronization distances for all the values of k. The *generalized k-prefix synchronization distance* between a string $\mathbf{u}$ and a codebook C is

$$pr_s^k(u, C) \triangleq \min_{\mathbf{c} \in C} \{pr_s^k(\mathbf{u}, \mathbf{c})\},$$

and the *generalized prefix synchronization distance* between $\mathbf{u}$ and C is

$$pr_s(\mathbf{u}, C) \triangleq (pr_s^0(\mathbf{u}, C), pr_s^1(\mathbf{u}, C), \ldots, pr_s^{|\mathbf{u}|}(u, C)).$$

To illustrate the previous definitions, if $\mathbf{u} = 11101$ and $C = \{1, 011, 00011101\}$, then $pr_s(\mathbf{u}, C) = (1, 0, 1, 2, 3, 3)$. More precisely, deleting one bit from 1 is the fastest way to

get the first 0 symbols of $\mathbf{u}$, the codeword 1 is the first bit of $\mathbf{u}$, inserting one bit in 1 is the fastest way to obtain the first two bits of $\mathbf{u}$, inserting two bits in 1 is the fastest way to obtain the first three bits of $\mathbf{u}$, one deletion followed by two insertions in 011 is the fastest way to obtain the first four bits of $\mathbf{u}$, and finally the fastest way to change a codeword of C into $\mathbf{u}$ is to delete the first three bits from 00011101.

The reason to define the prefix distances as above will be clear in the next section when describing how to modify the Viterbi decoding algorithm to correct synchronization errors.

6.3 Convolutional Codes and Viterbi Decoding Algorithm for Synchronization Errors

Convolutional codes were first introduced by Elias in 1955 [32] as an alternative to block codes, and a maximum-likelihood decoding for convolutional codes was presented by Viterbi in 1967 [107] and used in several applications of digital communications. It is assumed that the readers are familiar with convolutional codes and the Viterbi decoding algorithm, if not there are several references with detailed presentation of these subjects (e.g. [63, 50]). In this section, I show how the Viterbi decoding algorithm can be modified to correct synchronization errors when convolutional codes are used to encode the information sequences. A (n, k, v) convolutional encoder with encoding rate $R = \frac{k}{n}$, constraint length v, and memory order m that starts and ends in the "all-zero" state s_0 is used. A finite information sequence $\mathbf{u} = u_1 u_2 \dots u_{k \cdot \alpha}$ generates the finite encoded sequence $\mathbf{v} = v_1 v_2 \dots v_{n \cdot (\alpha+m)}$, transmitted over a channel subjected to synchronization errors, and received as $\mathbf{r} = r_1 r_2 \dots r_l$.

Since the transmitted codeword and the received sequence do not necessarily have the same length, we use metrics based on the prefix deletion and insertion distances defined in the previous section. This requires to compare branches of the trellis with shifted versions of the received sequence.

The decoding process is divided into *time units* of n bits and starts at time 0. Let $\mathbf{b} = b_0 b_1 \dots b_n$ be the bits of a branch in the decoding trellis, and let $m(s_a, s_b, t)$ be the *branch metric* for the branch going from state s_a to state s_b in the trellis at time t. Among all the paths from state s_0 at the start of the trellis to state s_b at time t, the *survivor path* is the one with the lowest metric. The *survivor metric* is the metric of the survivor path to state s_b at time t and denoted by $M(s_b, t)$, with the initial condition

that $M(s_0, 0) = 0$.

6.3.1 Correction of Deletion Errors

For deletion errors, we modify the Viterbi decoding algorithm so it chooses the smallest codeword deletion-reducible to the received sequence, i.e., the codeword such that the number of bits that need to get deleted from it to get the received sequence is minimized. This can be done in a recursive manner by considering the following branch metric making use of the prefix deletion distance between the bits of the branch and a shifted substring of the received sequence:

$$m(s_a, s_b, t) = pr_d(\mathbf{r}[(n \cdot (t-1) + 1 - M(s_a, t-1))..|\mathbf{r}|], \mathbf{b}). \tag{6.1}$$

Using this metric, the algorithm tries to include the bits of the received sequence from left to right in the survivor paths; the bits in the survivor paths that cannot be part of the received sequence correspond to bits that were deleted by the channel. The number of bits by which the received sequence has to be shifted when compared to the bits of a branch in the trellis corresponds to the number of bits that have to be deleted in the survivor path leading to the branch to get a prefix of the received sequence. Furthermore, since the length of the codewords is not fixed, the algorithm stops decoding when the entire received sequence is a subsequence of the survivor path ending in state s_0, i.e., when $M(s_0, t) + |\mathbf{r}| = t \cdot n$. This path ending in state s_0 at time t is the final survivor $\hat{\mathbf{v}}$.

To illustrate how the modified Viterbi algorithm works, consider, for instance, the (2,1,2) nonsystematic feedforward convolutional encoder whose generator matrix is given by

$$G(D) = [1 + D^2 \quad 1 + D + D^2],$$

and whose state diagram is shown in Figure 6.1. We suppose that the codeword 00 11 10 10 11 00 is transmitted and that the first, tenth, and eleventh bits are deleted by the channel, resulting in the received sequence 01 11 01 01 0. The trellis and the path metrics for this convolutional encoder and received sequence are presented in Figure 6.2. Consider the branch going from state s_0 to state s_0 at time $t = 3$. Since the survivor metric at state s_0 and time $t = 2$ is 3, three bits have to be deleted from the path $s_0 s_0 s_0$ to cover the first bit of the received sequence. Hence, the branch metric is the prefix deletion distance between the received string minus its first bit (11101010) and the bits of the branch (00). Since the next bit in the received sequence that has to be included in

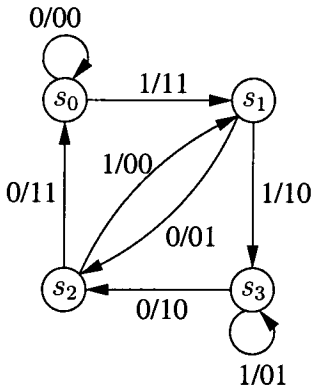

Figure 6.1: State diagram for the $G(D) = [1 + D^2 \quad 1 + D + D^2]$ convolutional encoder.

the path is a 1, the two 0s of the branch have to be deleted, thus the branch metric is 2 and the total metric for the path $s_0s_0s_0s_0$ is 5. Consider now the branch going from state s_2 to state s_0 at time $t = 3$. Since the survivor metric at state s_2 and time $t = 2$ is 2, the branch metric is the prefix deletion distance between the received substring 1101010 and the bits of the branch (11), thus the branch metric is 0, the total metric for the path $s_0s_1s_2s_0$ is 2, and the branch s_2s_0 is the survivor at time $t = 3$. The final survivor in the trellis is the codeword 00 11 10 10 11 00, which means that the algorithm was able to correct the three deletions.

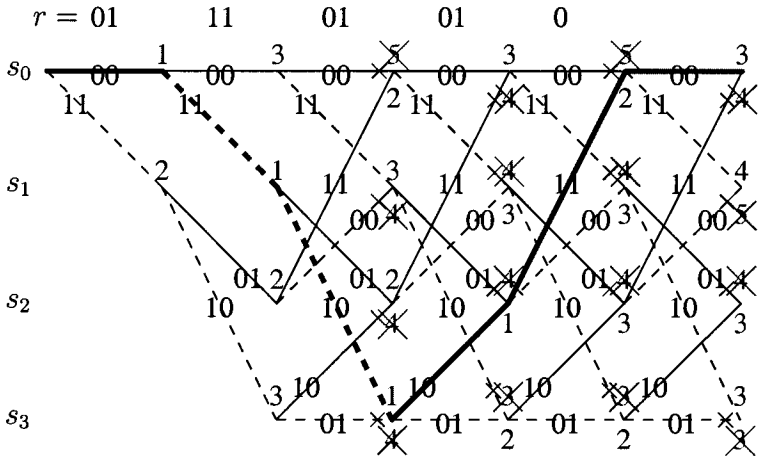

Figure 6.2: Deletion trellis for the $G(D) = [1 + D^2 \quad 1 + D + D^2]$ convolutional encoder and received sequence 01 11 01 01 0.

We now formally prove that the prefix deletion distance can be computed recursively and that the final survivor minimizes the number of deletions required to obtain the

received sequence.

Lemma 6.6. *If* $pr_d(\boldsymbol{u}, \boldsymbol{v}) = k_1$ *and* $pr_d(\boldsymbol{u}[(|\boldsymbol{v}| - k_1 + 1)..|\boldsymbol{u}|], \boldsymbol{w}) = k_2$, *then*

$$pr_d(\boldsymbol{u}, \boldsymbol{v} \circ \boldsymbol{w}) = k_1 + k_2.$$

Proof. From the assumptions, the first $|\mathbf{v}| - k_1$ symbols of $\mathbf{u}$ can be obtained by deleting k_1 symbols from $\mathbf{v}$, and the first $|\mathbf{w}| - k_2$ symbols of $\mathbf{u}[(|\mathbf{v}| - k_1 + 1)..|\mathbf{u}|]$ can be obtained by deleting k_2 symbols from $\mathbf{w}$. It follows that the first $|\mathbf{v}| + |\mathbf{w}| - k_1 - k_2$ symbols of $\mathbf{u}$ can be obtained by deleting $k_1 + k_2$ symbols from $\mathbf{v} \circ \mathbf{w}$, thus $pr_d(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) \leq k_1 + k_2$.

Suppose now that $pr_d(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) = k' < k_1 + k_2$. It follows that the first $|\mathbf{v}| + |\mathbf{w}| - k'$ symbols of $\mathbf{u}$ can be obtained either by deleting less than k_1 symbols from $\mathbf{v}$ or less than k_2 symbols from $\mathbf{w}$. If the former is true, then it is possible to obtain more than the first $|v| - k_1$ symbols of $\mathbf{u}$ by deleting symbols from $\mathbf{v}$, which contradicts the assumption that $pr_d(\mathbf{u}, \mathbf{v}) = k_1$. Deleting less than k_2 symbols from $\mathbf{w}$ leads to a similar contradiction, thus $pr_d(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) \geq k_1 + k_2$. □

Theorem 6.7. *The final survivor* $\hat{\mathbf{v}}$ *of the modified Viterbi algorithm for deletion errors is the smallest codeword deletion-reducible to the received sequence.*

Proof. From the previous lemma, every survivor metric in the trellis is the prefix deletion distance between the received sequence $\mathbf{r}$ and the bits of the corresponding survivor path. Furthermore, since the decoding stops when $M(s_0, t) + |\mathbf{r}| = t \cdot n$ for the survivor path ending in state s_0, the received sequence can be obtained by deleting $M(s_0, t)$ bits from $\hat{\mathbf{v}}$.

We now prove that no codeword smaller than $\hat{\mathbf{v}}$ is deletion-reducible to $\mathbf{r}$. First, the survivor paths ending at state s_0 with $t \cdot n < M(s_0, t) + |\mathbf{r}|$ are not deletion-reducible to $\mathbf{r}$. Second, halting the algorithm later than when $M(s_0, t) + |\mathbf{r}| = t \cdot n$ for the survivor path ending in state s_0 for the first time can only generate codewords longer than $\hat{\mathbf{v}}$. Third, it has to be proved that no discarded path can generate a sequence smaller than $\hat{\mathbf{v}}$ and deletion-reducible to $\mathbf{r}$. Suppose the opposite, i.e., that the shortest path deletion-reducible to $\mathbf{r}$, $\mathbf{v}'$, must keep one of the nonsurvivor paths at time t, and that the metrics of the nonsurvivor path $\mathbf{v}'[1..tn]$ and survivor path $\mathbf{p}[1..tn]$ are $k + \delta$ and k, respectively. From the previous lemma, the first $tn - (k + \delta)$ bits of $\mathbf{r}$ can be obtained by deleting $k + \delta$ bits from $\mathbf{v}'[1..tn]$, and the last $|\mathbf{r}| - tn + k + \delta$ bits of $\mathbf{r}$ can be obtained by deleting $|\mathbf{v}'| - |\mathbf{r}| - k - \delta$ bits from $\mathbf{v}'[tn + 1..|\mathbf{v}'|]$. It follows that the last $|\mathbf{r}| - tn + k$ bits of $\mathbf{r}$ can be obtained by deleting $|\mathbf{v}'| - |\mathbf{r}| - k$ bits from $\mathbf{v}'[tn + 1..|\mathbf{v}'|]$, but since the first $tn - k$ bits

of $\mathbf{r}$ can be obtained by deleting k bits from $\mathbf{p}[1..tn]$, it means that $\mathbf{p}[1..n] \circ \mathbf{v}'[tn+1..|\mathbf{v}'|]$ is deletion-reducible to $\mathbf{r}$, has the same length as $\mathbf{v}'$ and only uses survivor paths, which is a contradiction. □

Since computing $pr_d(\mathbf{a}, \mathbf{b})$ requires $|\mathbf{b}|$ comparisons, the modified Viterbi decoding algorithm for deletion errors has the same computational complexity and storage requirements as the original Viterbi decoding algorithm for substitution errors. The main difference is that the bit r_i in the received sequence has to be buffered and kept in memory until $M(s_a, t) + i \leq t \cdot n$ for all the survivor paths at time t in the trellis.

6.3.2 Correction of Insertion Errors

The Viterbi decoding algorithm can also be modified to correct insertion errors by choosing the largest codeword insertion-reducible to the received sequence. Again, this can be done recursively by using the following metric based on the prefix insertion distance:

$$m(s_a, s_b, t) = pr_i(\mathbf{r}[(n \cdot (t-1) + 1 + M(s_a, t-1))..|\mathbf{r}|], \mathbf{b}). \tag{6.2}$$

It is not always possible to obtain prefixes of the received sequence by inserting bits in the paths of the trellis, and it should be clear that any path longer than the received sequence cannot be one of its prefixes. Consequently, the main difference between the Viterbi algorithm for insertion and deletion errors is the stopping condition. For insertions, the algorithm can stop when none of the survivor paths in the trellis has a finite metric; the final survivor is the longest survivor path ending in state s_0 and with a finite metric.

An example of the trellis for the $G(D) = [1+D^2 \quad 1+D+D^2]$ convolutional encoder from Figure 6.1 is shown in Figure 6.3, where the codeword 11 10 10 00 01 11 is transmitted over an insertion channel and received as 11 11 10 10 00 01 11 0. Consider, for instance, the branch going from state s_2 to state s_0 at time $t = 4$. Since $M(s_2, 3) = 2$, it means that two bits need to be inserted in the survivor path ending at state s_0 and time $t = 3$ to obtain the first eight bits of the received sequence. Thus, the branch metric is the prefix insertion distance between the received sequence minus its first eight bits (00 01 11 0) and the bits of the branch (11). This requires the insertion of three zeros, thus the branch metric is 3 and the metric for the path $s_0 s_1 s_3 s_2 s_0$ is 5. Now examine the branch going from state s_0 to state s_0 at time $t = 4$. Since $M(s_0, 3) = 7$, the branch metric is the prefix insertion distance between the last two bits of the received sequence (10) and the bits of the branch (11). The distance is infinite because it is not possible

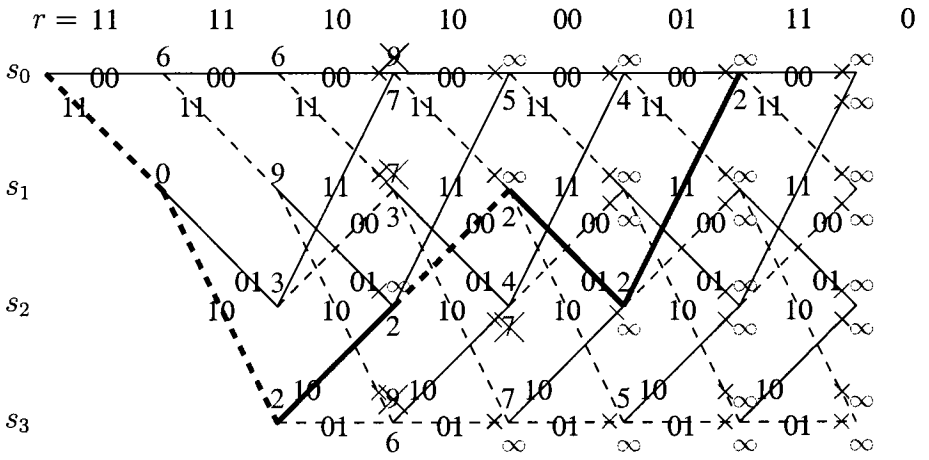

Figure 6.3: Insertion trellis for the $G(D) = [1 + D^2 \quad 1 + D + D^2]$ convolutional encoder and received sequence 11 11 10 10 00 01 11 0.

to obtain a prefix of 10 by inserting bits in 11, thus the branch $s_0 s_0$ is discarded and the survivor branch for state s_0 at time $t = 4$ is the branch $s_2 s_0$. The algorithm stops after seven time units when all the survivor metrics are infinite; the longest path with a finite metric ending in state s_0 is the final survivor 11 10 10 00 01 11. Although the path metric for the final survivor is 2, it does not include the insertion located at the very end of the received sequence, hence the three insertions were corrected by the algorithm.

As we did for deletion errors, we prove that the prefix insertion distance can be computed recursively and that the final survivor is the largest possible codeword which is a subsequence of $\mathbf{r}$.

Lemma 6.8. *If* $pr_i(\boldsymbol{u}, \boldsymbol{v}) = k_1$ *and* $pr_i(\boldsymbol{u}[(|\boldsymbol{v}| + k_1 + 1)..|\boldsymbol{u}|], \boldsymbol{w}) = k_2$, *then*

$$pr_i(\boldsymbol{u}, \boldsymbol{v} \circ \boldsymbol{w}) = k_1 + k_2.$$

Proof. If $k_1 < \infty$ and $k_2 < \infty$, then from the assumptions, the first $|\mathbf{v}| + k_1$ symbols of $\mathbf{u}$ can be obtained by inserting k_1 symbols in $\mathbf{v}$, and the first $|\mathbf{w}| + k_2$ symbols of $\mathbf{u}[(|\mathbf{v}| + k_1 + 1)..|\mathbf{u}|]$ can be obtained by inserting k_2 symbols in $\mathbf{w}$. It follows that the first $|\mathbf{v}| + |\mathbf{w}| + k_1 + k_2$ symbols of $\mathbf{u}$ can be obtained by inserting $k_1 + k_2$ symbols in $\mathbf{v} \circ \mathbf{w}$, thus $pr_i(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) \leq k_1 + k_2$. Suppose now that $pr_i(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) = k' < k_1 + k_2$. It follows that the first $|\mathbf{v}| + |\mathbf{w}| + k'$ symbols of $\mathbf{u}$ can be obtained either by inserting less than k_1 symbols in $\mathbf{v}$ or less than k_2 symbols in $\mathbf{w}$. If the former is true, then it is possible to obtain less than the first $|v| + k_1$ symbols of $\mathbf{u}$ by inserting symbols in $\mathbf{v}$,

which contradicts the assumption that $pr_i(\mathbf{u}, \mathbf{v}) = k_1$. Inserting less than k_2 symbols in $\mathbf{w}$ leads to a similar contradiction, thus $pr_i(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) \geq k_1 + k_2$ and the lemma holds when k_1 and k_2 are finite.

We prove the case when $k_1 = \infty$ and/or $k_2 = \infty$ by contrapositive. Suppose that $pr_i(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) = k_1' + k_2' < \infty$. It follows that a finite number of symbols k_1' can be inserted in $\mathbf{v}$ to obtain the first $|\mathbf{v}| + k_1'$ symbols of $\mathbf{u}$ and that the subsequent $|\mathbf{w}| + k_2'$ symbols of $\mathbf{u}$ can be obtained by inserting k_2' symbols in $\mathbf{w}$. This contradicts the assumption that $\mathbf{u}$ has an infinite prefix insertion distance with $\mathbf{v}$ and/or $\mathbf{w}$. The lemma follows. □

Theorem 6.9. *The final survivor $\hat{\mathbf{v}}$ of the modified Viterbi algorithm for insertion errors is the longest codeword insertion-reducible to the received sequence.*

Proof. From the previous lemma, every survivor metric in the trellis is the prefix insertion distance between the received sequence $\mathbf{r}$ and the bits of the corresponding survivor path. Thus, a survivor path in the trellis is insertion-reducible to the received sequence if and only if its metric is finite.

Similar to what we did in Theorem 6.7, we have to prove that no codeword longer than $\hat{\mathbf{v}}$ is insertion-reducible to $\mathbf{r}$. Since the smallest survivor metric at time $t+1$ cannot be smaller than the smallest survivor metric at time t, the algorithm can stop when all the survivor metrics are infinite. Among all the survivor paths ending in state s_0 with a finite metric, the one that requires the least number of insertions to be changed into the received sequence is the longest one. The last thing to prove is that no discarded path in the trellis can generate a codeword ending in the all-zero state, with a finite metric, and longer than $\hat{\mathbf{v}}$, which we do by contradiction. Suppose that $\mathbf{v}'$, the longest path insertion-reducible to $\mathbf{r}$ and ending in state s_0, must make use of a discarded path at, say, time t. We also assume that the metric of the path $\mathbf{v}'[1..tn]$ is $k + \delta$ and that the metric of the corresponding survivor path $\mathbf{p}[1..tn]$ at time t is k. Thus, the first $tn+k+\delta$ bits of $\mathbf{r}$ can be obtained by inserting $k + \delta$ bits in $\mathbf{v}'[1..tn]$ and the last $|\mathbf{r}| - tn - k - \delta$ bits of $\mathbf{r}$ by inserting $|\mathbf{r}| - |\mathbf{v}'| - k - \delta$ bits in $\mathbf{v}'[tn + 1..|\mathbf{v}'|]$. It follows that the last $|\mathbf{r}| - tn - k$ bits of $\mathbf{r}$ can be obtained by inserting $|\mathbf{r}| - |\mathbf{v}'| - k$ bits in $\mathbf{v}'[tn + 1..|\mathbf{v}'|]$ (just insert the bits $\mathbf{r}[tn + 1..tn + \delta]$ in front of $\mathbf{v}'[tn + 1..|\mathbf{v}'|]$ and the other $|\mathbf{r}| - |\mathbf{v}'| - k - \delta$ bits as before. Hence, since the first $tn + k$ bits of $\mathbf{r}$ can be obtained by inserting k bits in $\mathbf{p}[1..tn]$, the codeword $\mathbf{p}[1..tn] \circ \mathbf{v}'[tn + 1..|\mathbf{v}'|]$ has the same length as $\mathbf{v}'$ but only uses survivor paths, which is a contradiction. □

The complexity of the decoding algorithm is harder to calculate for insertions than for deletions because the number of comparisons varies: a branch with n bits and whose

metric is $m < \infty$ requires $n + m$ comparisons. This might seem prohibitive, but toward the end of the decoding process, several states have very high or infinite path metrics and as a result their outgoing branches can be discarded without comparisons. Overall, simulation results indicate that the modified Viterbi algorithm for insertions requires slightly more comparisons than the original version. For example, it can be shown that the trellis with 46 branches in Figure 6.3 requires 94 comparisons. Regarding the memory requirements, the algorithm might have to buffer a large number received bits before a branch metric can be calculated, and the bit r_i in the received sequence cannot be discarded until $i - M(s_a, t) \leq t \cdot n$ for all the survivor paths at time t in the trellis.

6.3.3 Correction of Duplication Errors

Surprisingly, it seems much more difficult to modify the Viterbi algorithm to correct duplication errors than to correct insertion errors. A metric based on the prefix duplication distance can be used, but a problem occurs because computing the prefix duplication distance between the received sequence and the paths in the trellis cannot be done recursively without memory. Consider the strings $\mathbf{u} = 0011$, $\mathbf{v} = 0$, and $\mathbf{w} = 1$. It can be seen that $pr_t(\mathbf{u}, \mathbf{v}) = 0$, $pr_t(\mathbf{u}[2..4], \mathbf{w}) = \infty$, and $pr_t(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) = 2$, hence

$$pr_t(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) \neq pr_t(\mathbf{u}, \mathbf{v}) + pr_t(\mathbf{u}[(|\mathbf{v}| + pr_t(\mathbf{u}, \mathbf{v}) + 1)..|\mathbf{u}|]).$$

It is not possible to increase the run of the string 1 to obtain 011, but the problematic leading 0 could have been inserted in $\mathbf{v}$ instead of $\mathbf{w}$. This problem is caused by the constraint on the insertions for duplication errors and formalized in Lemma 6.10.

Lemma 6.10. *If $\boldsymbol{v} = \boldsymbol{v}' \circ a$ and $\boldsymbol{w} = b \circ \boldsymbol{w}'$ and $a \neq b$ and $pr_t(\boldsymbol{u}, \boldsymbol{v}) = k_1 < \infty$ and $\alpha \geq 1$ and $u_i = a$ for $|\boldsymbol{v}| + k_1 + 1 \leq i \leq |\boldsymbol{v}| + k_1 + \alpha$ and $u_i \neq a$ for $|\boldsymbol{v}| + k_1 + \alpha + 1$ and $pr_t(\boldsymbol{u}[(|\boldsymbol{v}| + k_1 + \alpha + 1)..|\boldsymbol{u}|, \boldsymbol{w}]) = k_2 < \infty$, then*

$$pr_t(\boldsymbol{u}, \boldsymbol{v} \circ \boldsymbol{w}) = k_1 + k_2 + \alpha.$$

For all the other cases, if $pr_t(\boldsymbol{u}, \boldsymbol{v}) = k_1$ and $pr_t(\boldsymbol{u}[(\boldsymbol{v} + k_1 + 1)..|\boldsymbol{u}|], \boldsymbol{w}) = k_2$, then

$$pr_t(\boldsymbol{u}, \boldsymbol{v} \circ \boldsymbol{w}) = k_1 + k_2.$$

Proof. We prove the first case; all the other cases are similar to those proved in Lemma 6.8 and left to the reader. From $pr_t(\mathbf{u}, \mathbf{v}) = k_1$, the first $|\mathbf{v}| + k_1$ symbols of $\mathbf{u}$ can be

obtained by adding k_1 symbols in the runs of $\mathbf{v}$. Furthermore, since $\mathbf{v}_{|\mathbf{v}|} = a$ and $u_i = a$ for $|\mathbf{v}| + k_1 + 1 \leq i \leq |\mathbf{v}| + k_1 + \alpha$, the next α symbols of $\mathbf{u}$ can be obtained by further increasing the length of the last run of $\mathbf{v}$. Finally, from the assumption that $pr_t(\mathbf{u}[(|\mathbf{v}| + k_1 + \alpha + 1)..|\mathbf{u}|, \mathbf{w}]) = k_2$, the next $|\mathbf{w}| + k_2$ symbols of $\mathbf{u}$ can be obtained by increasing the runs of $\mathbf{w}$, hence the first $|\mathbf{v}| + |\mathbf{w}| + k_1 + k_2 + \alpha$ symbols of u can be obtained by inserting $k_1 + k_2 + \alpha$ symbols in $\mathbf{v} \circ \mathbf{w}$ and $pr_t(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) \leq k_1 + k_2 + \alpha$. Suppose now that $pr_t(\mathbf{u}, \mathbf{v} \circ \mathbf{w}) = k' < k_1 + k_2 + \alpha$. We prove that it is not possible to increase the runs of $\mathbf{v}$ while satisfying the lemma without violating one of the assumptions. First, inserting less than k_1 symbols in $\mathbf{v}$ cannot give a prefix of u because it contradicts the fact that $pr_t(\mathbf{u}, \mathbf{v}) = k_1$. Second, if $k_1 \leq l < k_1 + \alpha$ symbols are inserted in $\mathbf{v}$ to obtain a prefix of $\mathbf{u}$, then it is not possible to obtain a prefix of $\mathbf{u}[|\mathbf{v}| + l + 1..|\mathbf{u}|]$ by increasing the runs of $\mathbf{w}$ because the two strings do not begin with the same symbol. Third, if exactly $k_1 + \alpha$ symbols are added in the runs of $\mathbf{v}$ to obtain a prefix of $\mathbf{u}$, then at most $k' - k_1 - \alpha < k_2$ symbols can be inserted in $\mathbf{w}$, contradicting the assumption that $pr_t(\mathbf{u}[(|\mathbf{v}| + k_1 + \alpha + 1)..|\mathbf{u}|, \mathbf{w}]) = k_2$. Finally, since $pr_t(\mathbf{u}, \mathbf{v}) = k_1$, it follows that $\mathbf{v}$ and $\mathbf{u}[1..(|\mathbf{v}| + k_1)]$ have the same number of runs. Moreover, since $\mathbf{u}_{|\mathbf{v}|+k_1+\alpha} \neq \mathbf{u}_{|\mathbf{v}|+k_1+\alpha+1}$, it is not possible to obtain a prefix of $\mathbf{u}$ by inserting more than $k_1 + \alpha$ in $\mathbf{v}$, because the two strings do not have the same number of runs. □

Using Lemma 6.10, the branch metric used for the modified Viterbi algorithm for duplication errors is as follows. Let $\mathbf{p}$ be a path ending in state s_a at time $t-1$ and whose metric is k_m, and let $\mathbf{b}$ be the string of bits for the branch $s_a s_b$ at time t. If $\mathbf{b}$ starts with the symbol a and if the last symbol of $\mathbf{p}$ is the symbol b different from a and if $\mathbf{r}[(|\mathbf{p}| + k_m + 1)..|\mathbf{r}|]$ starts with a run of α bs, then

$$m(s_a, s_b, t) = \alpha + pr_t(\mathbf{r}[(|\mathbf{p}| + k_m + \alpha + 1)..|\mathbf{r}|], \mathbf{b}),$$

otherwise,

$$m(s_a, s_b, t) = pr_t(\mathbf{r}[(|\mathbf{p}| + k_m + 1)..|\mathbf{r}|], \mathbf{b}).$$

This definition insures that the metric of a path is the prefix duplication distance between the received sequence and the said path. However, the undesired consequence is that if the Viterbi algorithm is used with this metric and if only one survivor path per state is kept at each step of the decoding process, then it is not guaranteed to output the largest codeword that can be changed into the received sequence by increasing the length of its runs. In fact, in some cases the algorithm might even fail to return a valid

codeword. Consequently, a more sophisticated approach is needed differing from the modified algorithm for insertion errors in three ways. First, the algorithm must keep in memory more than one survivor path per state. For state s_b at time t in the trellis, the algorithm must store the tuple (s_a, k_m) if there is at least one path going from the beginning of the trellis to state s_b at time t via state s_a at time $t-1$ and whose metric is $k_m < \infty$. Second, the branch metrics must be calculated according to Lemma 6.10 as explained in the previous paragraph. Third, the final survivor $\hat{\mathbf{v}}$ is the longest path in the trellis ending in the all-zero state and with a finite metric of $k_m < \infty$ and with $\mathbf{r}_i = \mathbf{r}_{|\hat{\mathbf{v}}|+k_m}$ for $i > |\hat{\mathbf{v}}| + k_m$.

A trellis sample for the modified Viterbi decoding algorithm for duplication errors using the convolutional encoder from Figure 6.1 is presented in Figure 6.4, where the codeword 11 01 11 00 11 01 11 is transmitted over a channel with duplication errors and received as 11 11 11 10 01 11 10 00 11 01 11. Consider, for example, the branch going from states s_2 to s_1 at time $t = 3$, which is an instance of the problematic case from Lemma 6.10. Although the metric for the path $s_0 s_1 s_2$ is 6 and the prefix duplication distance between $\mathbf{r}[11..|\mathbf{r}|]$ and the bits of the branch is infinite, the three bits $\mathbf{r}[11..13]$ could have been inserted in the last run of the branch $s_1 s_2$ at time $t = 2$, thus $m(s_2, s_1, 3) = 3 + pr_t(\mathbf{r}[14..|\mathbf{r}|], 00) = 3 + 0 = 3$. Now consider the branch going from s_1 to s_2 at time $t = 6$. Since there are two paths with a finite metric ending in state s_1 at time $t = 5$, the algorithm must compute the branch metric twice. For the path with metric 6 ending by the branch $s_2 s_1$, $m(s_a, s_b, 6)$ is infinite, whereas $m(s_a, s_b, 6) = 0$ for the path with metric 8 ending by the branch $s_0 s_1$. This means that the beginning of the best path to state s_1 at time $t = 6$ is not the best path to state s_2 at time $t = 5$. Had only a single survivor path been kept, the algorithm would have failed to find the longest codeword duplication-reducible to the received sequence.

When backtracking to find the final survivor path, the algorithm matches the symbols of the received sequence with the symbols of the path from right to left. When it reaches a state with more than one possible survivor, it chooses the one whose metric matches the number of duplication errors not yet covered by the tail end of the final survivor. Going back to the trellis of Figure 6.4, the algorithm backtracks until it reaches state s_1 at time $t = 5$ where two survivor paths appear. Since the end of $\mathbf{v}'$ perfectly matches the last four bits or $\mathbf{r}$ (0111), a path with ten bits must cover the first eighteen bits of $\mathbf{r}$, thus the algorithm must keep backtracking following the survivor whose metric is 8. The final survivor is 11 10 01 01 10 11, thus the algorithm was able to correct all eight duplication errors introduced by the channel. One final remark: if at any point there are two or more

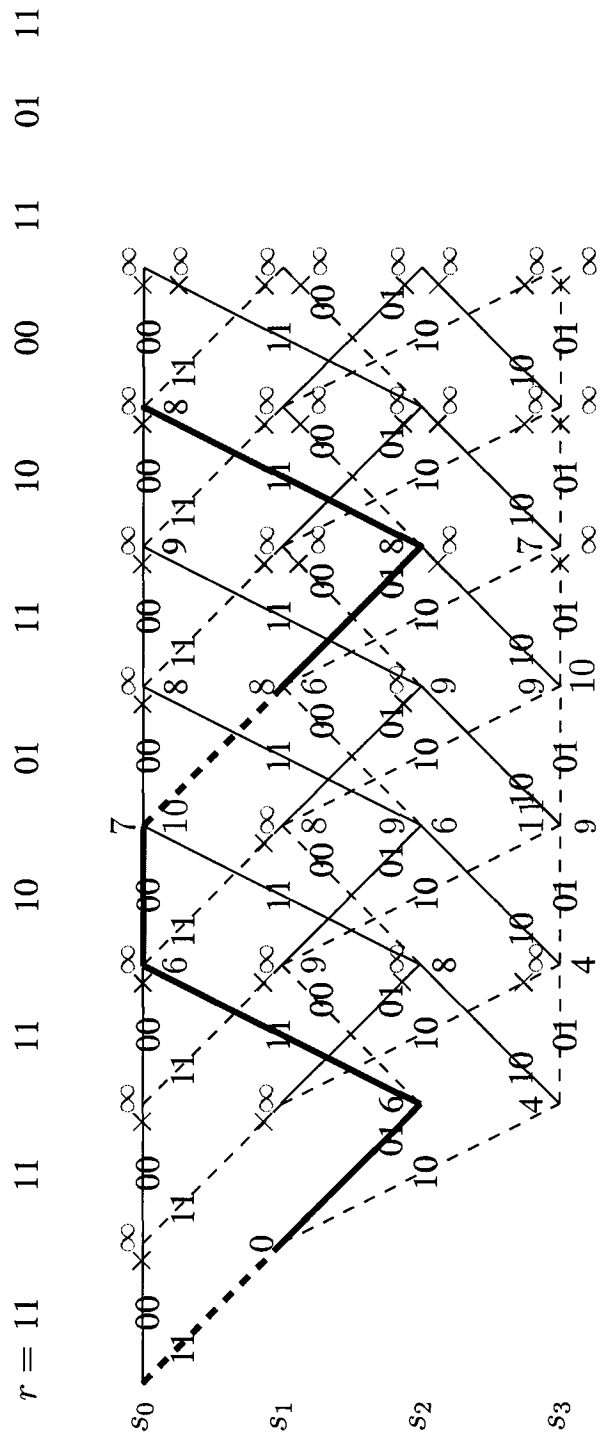

Figure 6.4: Duplication trellis for the $G(D) = [1 + D^2 \quad 1 + D + D^2]$ convolutional encoder and received sequence 11 11 11 10 01 11 10 00 11 01 11.

survivors with the same metric and different ancestors, one of them is chosen randomly because they all lead to codewords duplication-reducible to the received sequence.

Theorem 6.11. *The final survivor $\hat{\mathbf{v}}$ of the modified Viterbi algorithm for duplication errors is the longest codeword duplication-reducible to the received sequence.*

Proof. Since every path metric in the trellis is the prefix duplication distance between the received sequence $\mathbf{r}$ and the bits of the corresponding path, and since only the paths with infinite metrics are discarded, it is clear that the longest codeword duplication-reducible to the received sequence is not discarded by the algorithm. □

It seems that the complexity of the algorithm for duplication errors is higher than the complexity of the original Viterbi algorithm, but it is not clear how much higher. On one end, it must compute some of the branch metrics several times, but on the other end for several branches no comparison is necessary. For the trellis presented in Figure 6.4, 130 comparisons for 54 branches are required.

6.3.4 Correction of Synchronization Errors

When only deletion (or insertion) errors are present, the modified Viterbi algorithm knows that the bits in the trellis that do not match the received sequence were deleted (inserted). This means that at any point during the decoding process, the algorithm knows how much ahead (or behind) the optimal decoded sequence is with respect to the received sequence. This asymmetry also explains why the complexity of the algorithm is the same as the complexity of the original Viterbi algorithm. However, when more than one type of error among insertions, deletions and substitutions occur together, the algorithm does not know if a bit in the trellis that does not match the received sequence has been inserted, deleted or flipped. For instance, although a bit deletion gives a better metric than two insertions, it is possible if the bits were indeed inserted in the codeword that inserting the two bits might lead to a better metric at the end of the decoding process. Consequently, in order for the Viterbi algorithm to find the codeword whose synchronization distance from the received sequence is minimized, it has to keep track of the best way to obtain the first k bits of the received sequence for $0 \leq k \leq |\mathbf{r}|$, which is precisely what the generalized prefix synchronization distance provides. Before describing the algorithm in detail, I first prove that all the required metric calculations can be done recursively and efficiently.

Lemma 6.12. *Let $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ be strings, and let w' be a symbol from the alphabet Σ.*

1. *If* $|\boldsymbol{v}| = 0$ *and* $0 \leq k \leq |\boldsymbol{u}|$, *then*

$$pr_s^k(\boldsymbol{u}, \boldsymbol{v}) = k;$$

2. *if* $\boldsymbol{v} = \boldsymbol{w} \circ w'$, *then*

$$pr_s^0(\boldsymbol{u}, \boldsymbol{v}) = pr_s^0(\boldsymbol{u}, \boldsymbol{w}) + 1;$$

3. *if* $\boldsymbol{v} = \boldsymbol{w} \circ w'$ *and* $0 < k \leq |\boldsymbol{u}|$ *and* $u_k \neq w'$, *then*

$$pr_s^k(\boldsymbol{u}, \boldsymbol{v}) = \min(pr_s^{k-1}(\boldsymbol{u}, \boldsymbol{v}) + 1, pr_s^k(\boldsymbol{u}, \boldsymbol{w}) + 1);$$

4. *if* $\boldsymbol{v} = \boldsymbol{w} \circ w'$ *and* $0 < k \leq |\boldsymbol{u}|$ *and* $u_k = w'$, *then*

$$pr_s^k(\boldsymbol{u}, \boldsymbol{v}) = pr_s^{k-1}(\boldsymbol{u}, \boldsymbol{w}).$$

Proof. If $|\mathbf{v}| = 0$, then k symbols must be inserted to get the first k symbols of $\mathbf{u}$, which proves the first case. For the rest of the proof we assume that $\mathbf{v} = \mathbf{w} \circ w'$. $|\mathbf{w}|$ and $|\mathbf{w}| + 1$ symbols must respectively be deleted from $\mathbf{w}$ and $\mathbf{v}$ to get the first zero bits of $\mathbf{u}$, which proves the second case.

Suppose now that $u_k \neq w'$. The first k bits of $\mathbf{u}$ can be obtained by transforming $\mathbf{v}$ into the first $k-1$ bits of $\mathbf{u}$ followed by inserting u_k at the end of the string, or by deleting w' followed by transforming $\mathbf{w}$ into the first k bits of $\mathbf{u}$. Thus,

$$pr_s^k(\mathbf{u}, \mathbf{v}) \leq \min(pr_s^{k-1}(\mathbf{u}, \mathbf{v}) + 1, pr_s^k(\mathbf{u}, \mathbf{w}) + 1).$$

Suppose now that $pr_s^k(\mathbf{u}, \mathbf{v}) < \min(pr_s^{k-1}(\mathbf{u}, \mathbf{v}) + 1, pr_s^k(\mathbf{u}, \mathbf{w}) + 1)$. Since $u_k \neq w'$, either w' has to be deleted from $\mathbf{v}$ or u_k has to be inserted at the end of $\mathbf{v}$. If it is the former, then $\mathbf{w}$ can be changed into the first k bits of $\mathbf{u}$ with less than $pr_s^k(\mathbf{u}, \mathbf{w})$ insertions and deletions, which is a contradiction. If it is the latter, then $\mathbf{v}$ can be changed into the first $k-1$ bits of $\mathbf{u}$ with less than $pr_s^{k-1}(\mathbf{u}, \mathbf{v})$ insertions and deletions, which is another contradiction, and this proves the third case.

For the fourth case, the first k bits of $\mathbf{u}$ can be obtained by transforming $\mathbf{w}$ into the first $k-1$ bits of $\mathbf{u}$ while leaving w' unchanged, thus $pr_s^k(\mathbf{u}, \mathbf{v}) \leq pr_s^{k-1}(\mathbf{u}, \mathbf{w})$. Finally, it should be clear that $pr_s^k(\mathbf{u}, \mathbf{v}) < pr_s^{k-1}(\mathbf{u}, \mathbf{w})$ leads to a contradiction, which completes the proof of the lemma. □

Table 6.1: Prefix synchronization distance between the strings $\mathbf{u} = 01101$ and $\mathbf{v} = 110$.

	$u_0 = \epsilon$	$u_1 = 0$	$u_2 = 1$	$u_3 = 1$	$u_4 = 0$	$u_5 = 1$
$v_0 = \epsilon$	0	1	2	3	4	5
$v_1 = 1$	1	2	1	2	3	4
$v_2 = 1$	2	3	2	1	2	3
$v_3 = 0$	3	2	3	2	1	2

As done several times in Chapter 4, the prefix synchronization distance can be calculated using dynamic programming. Table 6.1 shows an example for $u = 01101$ and $v = 110$. The prefix synchronization distance is the last row of the table, i.e., $pr_s(01101, 110) = (3, 2, 3, 2, 1, 2)$. A consequence of Lemma 6.12 is that the k-prefix synchronization distances between $\mathbf{u}$ and $\mathbf{v} \circ \mathbf{w}$ can be calculated using only the symbols of $\mathbf{u}$, the symbols of $\mathbf{w}$, and the k-prefix synchronization distances between $\mathbf{u}$ and $\mathbf{v}$, i.e., without knowing precisely what the string $\mathbf{v}$ is. More importantly, this is also holds for the generalized prefix synchronization distance, as shown next.

Lemma 6.13. *Let $\boldsymbol{u}$ be a string, C a codebook with codewords of length n, and $C' \triangleq \{\mathbf{c} \circ w | \mathbf{c} \in C\}$ the codebook consisting of all the codewords of C with the symbol $w \in \Sigma$ appended at the end.*

$$pr_s^k(\boldsymbol{u}, C') = \begin{cases} pr_s^k(\boldsymbol{u}, C) + 1 & \text{if } k = 0; \\ \min(pr_s^k(\boldsymbol{u}, C) + 1, pr_s^{k-1}(\boldsymbol{u}, C') + 1) & \text{if } 0 < k \leq |\boldsymbol{u}| \text{ and } u_k \neq w; \\ pr_s^{k-1}(\boldsymbol{u}, C) & \text{if } 0 < k \leq |\boldsymbol{u}| \text{ and } u_k = w. \end{cases}$$

Proof. Changing a codeword of C into the empty string requires the deletion of n symbols, whereas $n + 1$ deletions are required for the codewords of C'. This proves the first case. Let $0 < k \leq |\mathbf{u}|$, $u_k \neq w$, and let $\mathbf{c} \in C$ and $\mathbf{c'} \in C'$ be codewords such that $pr_s^k(\mathbf{u}, \mathbf{c}) = pr_s^k(\mathbf{u}, C)$ and $pr_s^{k-1}(\mathbf{u}, \mathbf{c'}) = pr_s^{k-1}(\mathbf{u}, C')$, respectively. It follows that the codeword $\mathbf{c} \circ w \in C'$ can be changed into the first k symbols of $\mathbf{u}$ by deleting w and by changing $\mathbf{c}$ into the first k symbols of $\mathbf{u}$ with $pr_s^k(\mathbf{u}, C)$ operations, thus

$$pr_s^k(\mathbf{u}, C') \leq pr_s^k(\mathbf{u}, C) + 1. \tag{6.3}$$

The codeword $\mathbf{c'}$ can also be changed into the first k symbols of $\mathbf{u}$ by changing it into first $k - 1$ symbols of $\mathbf{u}$ with $pr_s^{k-1}(\mathbf{u}, C')$ operations followed by the insertion of u_k at

the end of the string, thus

$$pr_s^k(\mathbf{u}, C') \leq pr_s^{k-1}(\mathbf{u}, C') + 1, \tag{6.4}$$

and $pr_s^k(\mathbf{u}, C') \leq \min(pr_s^k(\mathbf{u}, C) + 1, pr_s^{k-1}(\mathbf{u}, C') + 1)$ follows from (6.3) and (6.4).

Suppose now that $pr_s^k(\mathbf{u}, C') < \min(pr_s^k(\mathbf{u}, C) + 1, pr_s^{k-1}(\mathbf{u}, C') + 1)$. It follows that there is a codeword $\mathbf{c} \circ w \in C'$ with $pr_s^k(\mathbf{u}, \mathbf{c} \circ w) < \min(pr_s^k(\mathbf{u}, C) + 1, pr_s^{k-1}(\mathbf{u}, C') + 1)$. If the fastest way to change $\mathbf{c} \circ w$ into the first k bits of $\mathbf{u}$ must delete w, then the first k bits of u can be obtained with $pr_s^k(\mathbf{u}, C) - 1$ synchronization operations from $\mathbf{c}$, which is a contradiction. On the other end, if the fastest way to change $\mathbf{c} \circ w$ into the first k bits of $\mathbf{u}$ must insert the symbol u_k, then it is possible to obtain the first $k-1$ bits of u from $\mathbf{c} \circ w$ with $pr_s^{k-1}(\mathbf{u}, C') - 1$ synchronization operations, which again is a contradiction. Hence, $pr_s^k(\mathbf{u}, C') \geq \min(pr_s^k(\mathbf{u}, C) + 1, pr_s^{k-1}(\mathbf{u}, C') + 1)$.

As for the third case, if $\mathbf{c} \in C$ is a codeword such that $pr_s^{k-1}(\mathbf{u}, \mathbf{c}) = pr_s^{k-1}(\mathbf{u}, C)$, then $\mathbf{c} \circ w \in C'$ can be transformed into the first bits of $\mathbf{u}$ by changing $\mathbf{c}$ into the first $k-1$ symbols of $\mathbf{u}$ while leaving w unchanged, which leads to $pr_s^k(\mathbf{u}, C') \leq pr_s^{k-1}(\mathbf{u}, C)$. Assuming that $pr_s^k(\mathbf{u}, C') < pr_s^{k-1}(\mathbf{u}, C)$ leads to a contradiction, and the result follows. □

Another important property is that the generalized prefix synchronization distance of the union of two codebooks can easily be calculated from the generalized prefix synchronization distance of each of them, as shown next.

Lemma 6.14. *Let $\boldsymbol{u}$ be a string. If C and C' are codebooks with codewords of length n, then*

$$pr_s^k(\boldsymbol{u}, C \cup C') = \min(pr_s^k(\boldsymbol{u}, C), pr_s^k(\boldsymbol{u}, C')).$$

Proof. Let $\mathbf{c} \in C$ and $\mathbf{c'} \in C'$ be codewords such that $pr_s^k(\mathbf{u}, \mathbf{c}) = pr_s^k(\mathbf{u}, C)$ and $pr_s^k(\mathbf{u}, \mathbf{c'}) = pr_s^k(\mathbf{u}, C')$, respectively. Clearly, $\mathbf{c} \in C \cup C'$ and $\mathbf{c'} \in C \cup C'$, and it follows that $pr_s^k(\mathbf{u}, C \cup C') \leq pr_s^k(\mathbf{u}, C)$ and $pr_s^k(\mathbf{u}, C \cup C') \leq pr_s^k(\mathbf{u}, C')$. Consequently,

$$pr_s^k(\mathbf{u}, C \cup C') \leq \min(pr_s^k(\mathbf{u}, C), pr_s^k(\mathbf{u}, C')).$$

Suppose now that $pr_s^k(\mathbf{u}, C \cup C') < \min(pr_s^k(\mathbf{u}, C), pr_s^k(\mathbf{u}, C'))$. It follows that there exists $\mathbf{c} \in C \cup C'$ with $pr_s^k(\mathbf{u}, \mathbf{c}) < \min(pr_s^k(\mathbf{u}, C), pr_s^k(\mathbf{u}, C'))$. If $\mathbf{c} \in C$, then $pr_s^k(\mathbf{u}, \mathbf{c}) \leq \min(pr_s^k(\mathbf{u}, C), pr_s^k(\mathbf{u}, C')) \leq pr_s^k(\mathbf{u}, C)$, which is a contradiction. A simi-

lar contradiction appears if $\mathbf{c} \in C'$, therefore

$$pr_s^k(\mathbf{u}, C \cup C') \geq \min(pr_s^k(\mathbf{u}, C), pr_s^k(\mathbf{u}, C')).$$

□

The properties of Lemmas 6.13 and 6.14 are exactly what is required for the prefix synchronization distance to be used as a metric in yet another modified version of the Viterbi decoding algorithm for synchronization errors. More precisely, the metric $m(s_a, s_b, t)$ for the branch $s_a s_b$ in the trellis at time t is the generalized prefix synchronization distance between the received sequence and the codebook consisting of the strings for all the paths starting at the beginning of the trellis and ending with that branch, and from Lemma 6.13 this can easily be calculated recursively. The survivor metric $M(s_a, t)$ for state s_a in the trellis at time t is the generalized prefix synchronization distance between the received sequence and the codebook consisting of the strings of all the paths starting at the beginning of the trellis and ending in that state, and this can be calculated recursively using Lemma 6.14. The metric $M(s_0, 0)$ for the all-zero state at time $t = 0$ is initialized to $M(s_0, 0) \triangleq pr_s(\mathbf{u}, \epsilon) = (0, 1, 2, \ldots, |\mathbf{r}|)$.

It is much easier to understand how the algorithm works with an example using the $G(D) = [1 + D^2 \quad 1 + D + D^2]$ convolutional encoder whose state diagram is shown in Figure 6.1. Suppose that the codeword 00 11 01 11 00 is transmitted over a channel with synchronization errors and that two insertions and one deletion corrupt it, resulting in the received sequence $\mathbf{r} = 01\ 01\ 11\ 11\ 01\ 0$. Since each metric is a 12-tuple, the empty trellis for the received sequence is shown in Figure 6.5, a sample of the trellis with the branch and state metrics is presented in Figure 6.6, and all the metrics are included in Table 6.2.

Consider first Figure 6.6, showing the two paths going from the start of the trellis to state s_1 at time $t = 3$. The metric for the all-zero state at time $t = 0$ is initialized as mentioned above and can be used to calculate the two metrics at time $t = 1$, which in turn are used to calculate the metrics for the states s_0 and s_2 at time $t = 2$, which are used to find the metrics for the paths $s_0 s_1$ and $s_2 s_1$ at time $t = 3$. Since there are two merging paths at state s_1 and $t = 3$, the state metric consists of the minimum value, between both path metrics, of the generalized k-prefix synchronization distances. When the two of them are equal, the algorithm chooses one randomly, thus $M(s_1, 3) = (6, 5, 4, 3, 4, 3, 4, 5, 6, 7, 8, 7)$. It is important to understand that there are $|\mathbf{r}| + 1$ survivor paths for each state in the trellis: the best path that can be changed into

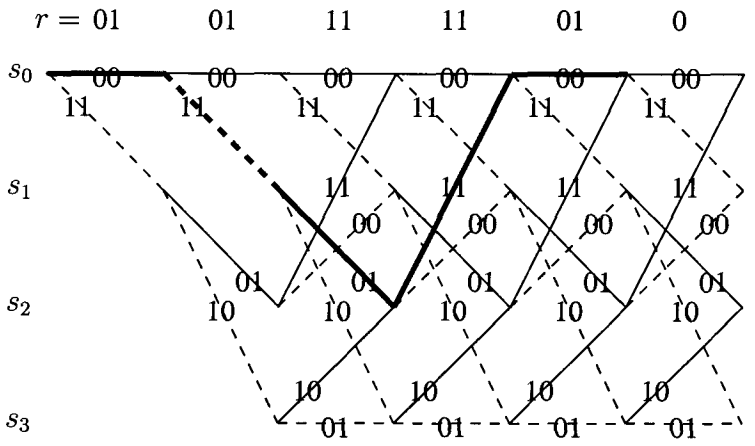

Figure 6.5: Empty trellis for the $G(D) = [1 + D^2 \quad 1 + D + D^2]$ convolutional encoder and received sequence 01 01 11 11 01 0.

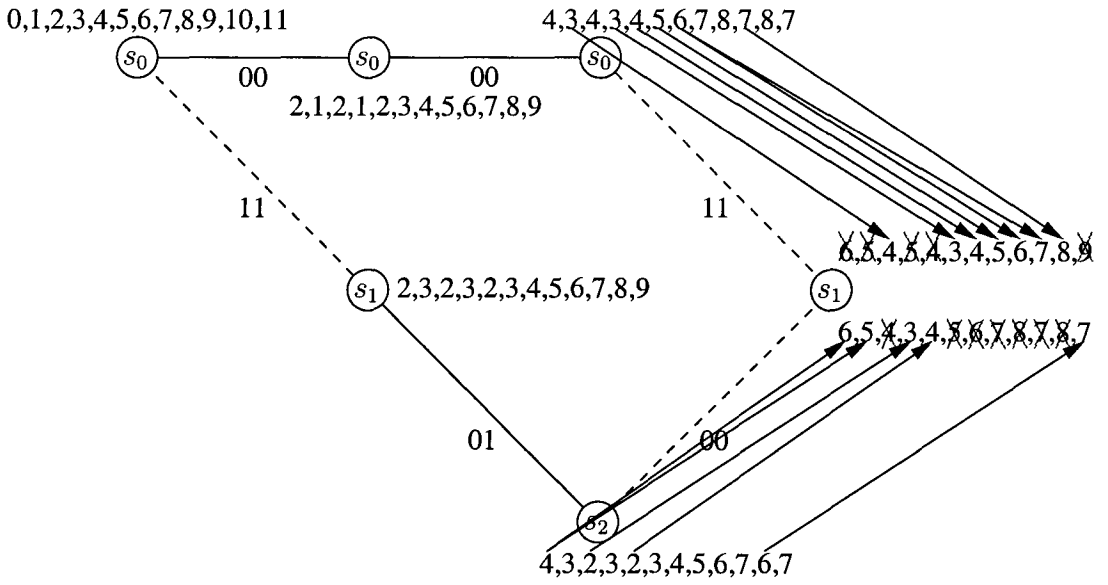

Figure 6.6: Trellis sample for the modified Viterbi decoding algorithm for synchronization errors with the $G(D) = [1+D^2 \quad 1+D+D^2]$ convolutional encoder and received sequence 01 01 11 11 01 0.

Table 6.2: Metrics, prefix survivors, and state survivors for the modified Viterbi decoding algorithm for synchronization errors with the convolutional encoder $G(D) = [1 + D^2 \quad 1 + D + D^2]$ and the received sequence 01 01 11 11 01 0.

t	s	$pr_s^0(\mathbf{r}, C)$	$pr_s^1(\mathbf{r}, C)$	$pr_s^2(\mathbf{r}, C)$	$pr_s^3(\mathbf{r}, C)$	$pr_s^4(\mathbf{r}, C)$	$pr_s^5(\mathbf{r}, C)$	$pr_s^6(\mathbf{r}, C)$	$pr_s^7(\mathbf{r}, C)$	$pr_s^8(\mathbf{r}, C)$	$pr_s^9(\mathbf{r}, C)$	$pr_s^10(\mathbf{r}, C)$	$pr_s^11(\mathbf{r}, C)$
0	s_0	(0,0,-)	(1,1,-)	(2,2,-)	(3,3,-)	(4,4,-)	(5, 5,-)	(6,6,-)	(7,7,-)	(8,8,-)	(9,9,-)	(10,10,-)	(11,11,-)
1	s_0	(2,0,s_0)	(1,0,s_0)	(2,0,s_0)	(1,0,s_0)	(2,0,s_0)	(3,0,s_0)	(4,0,s_0)	(5,0,s_0)	(6,0,s_0)	(7,2,s_0)	(8,2,s_0)	(9,8,s_0)
1	s_1	(2,0,s_0)	(3,1,s_0)	(2,1,s_0)	(3,1,s_0)	(2,1,s_0)	(3,3,s_0)	(4,4,s_0)	(5,5,s_0)	(6,6,s_0)	(7,6,s_0)	(8,7,s_0)	(9,7,s_0)
2	s_0	(4,0,s_0)	(3,0,s_0)	(4,2,s_0)	(3,2,s_0)	(4,4,s_0)	(5,5,s_0)	(6,6,s_0)	(7,7,s_0)	(8,8,s_0)	(7,8,s_0)	(8,8,s_0)	(7,8,s_0)
2	s_1	(4,0,s_0)	(3,1,s_0)	(2,1,s_0)	(3,3,s_0)	(2,3,s_0)	(1,3,s_0)	(2,4,s_0)	(3,5,s_0)	(4,6,s_0)	(5,6,s_0)	(6,7,s_0)	(7,7,s_0)
2	s_2	(4,0,s_1)	(3,0,s_1)	(2,0,s_1)	(3,2,s_1)	(2,2,s_1)	(3,4,s_1)	(4,5,s_1)	(5,6,s_1)	(6,7,s_1)	(7,8,s_1)	(6,8,s_1)	(7,8,s_1)
2	s_3	(4,0,s_1)	(3,0,s_1)	(4,1,s_1)	(3,1,s_1)	(4,3,s_1)	(3,4,s_1)	(4,5,s_1)	(5,6,s_1)	(6,7,s_1)	(5,7,s_1)	(6,7,s_1)	(7,9,s_1)
3	s_0	(6,0,s_0)	(5,0,s_0)	(4,1,s_2)	(5,2,s_0)	(4,3,s_2)	(3,3,s_2)	(2,4,s_2)	(3,5,s_2)	(4,6,s_2)	(5,6,s_2)	(6,7,s_2)	(7,7,s_2)
3	s_1	(6,0,s_2)	(5,0,s_2)	(4,1,s_0)	(3,2,s_2)	(4,4,s_2)	(3,3,s_0)	(4,4,s_0)	(5,5,s_0)	(6,6,s_0)	(7,6,s_0)	(8,9,s_0)	(7,10,s_2)
3	s_2	(6,0,s_1)	(5,0,s_1)	(4,0,s_1)	(3,2,s_1)	(2,2,s_1)	(3,4,s_1)	(2,5,s_1)	(3,6,s_1)	(4,7,s_1)	(5,8,s_1)	(4,8,s_1)	(5,8,s_1)
3	s_3	(6,0,s_1)	(5,0,s_1)	(4,1,s_1)	(3,1,s_1)	(4,3,s_1)	(3,4,s_1)	(2,5,s_1)	(3,6,s_1)	(4,7,s_1)	(3,7,s_1)	(4,7,s_1)	(5,9,s_1)
4	s_0	(8,0,s_0)	(7,0,s_0)	(6,2,s_0)	(5,2,s_0)	(4,3,s_2)	(3,3,s_2)	(2,4,s_2)	(3,5,s_2)	(2,6,s_2)	(3,6,s_2)	(4,7,s_2)	(5,8,s_0)
4	s_1	(8,0,s_0)	(7,1,s_0)	(6,1,s_0)	(5,2,s_2)	(4,4,s_2)	(5,3,s_0)	(4,4,s_0)	(3,5,s_0)	(2,6,s_0)	(3,6,s_0)	(4,7,s_0)	(5,7,s_0)
4	s_2	(8,0,s_1)	(7,0,s_1)	(6,0,s_1)	(5,2,s_1)	(4,2,s_1)	(5,4,s_1)	(4,5,s_1)	(3,6,s_3)	(4,7,s_3)	(3,7,s_3)	(4,9,s_3)	(3,9,s_3)
4	s_3	(8,0,s_1)	(7,0,s_1)	(6,1,s_1)	(5,1,s_1)	(4,3,s_1)	(5,4,s_1)	(4,5,s_1)	(3,6,s_3)	(4,7,s_3)	(5,7,s_1)	(4,8,s_3)	(5,10,s_3)
5	s_0	(10,0,s_0)	(9,0,s_0)	(8,2,s_0)	(7,2,s_0)	(6,4,s_0)	(5,5,s_0)	(4,6,s_0)	(5,7,s_0)	(4,8,s_0)	(3,8,s_0)	(4,8,s_0)	(3,8,s_0)
5	s_1	(10,0,s_0)	(9,1,s_0)	(8,1,s_0)	(7,3,s_0)	(6,3,s_0)	(5,3,s_0)	(4,4,s_0)	(3,5,s_0)	(2,6,s_0)	(3,6,s_0)	(4,9,s_0)	(5,9,s_0)
5	s_2	(10,0,s_1)	(9,0,s_1)	(8,0,s_1)	(7,2,s_1)	(6,2,s_1)	(5,4,s_1)	(6,5,s_1)	(5,6,s_1)	(4,7,s_1)	(3,8,s_1)	(2,8,s_1)	(3,8,s_1)
5	s_3	(10,0,s_1)	(9,0,s_1)	(8,1,s_1)	(7,1,s_1)	(6,3,s_1)	(5,4,s_1)	(6,5,s_1)	(5,6,s_1)	(4,7,s_1)	(3,7,s_1)	(4,9,s_1)	(3,9,s_1)
6	s_0	(12,0,s_0)	(11,0,s_0)	(10,2,s_0)	(9,2,s_0)	(8,4,s_0)	(7,5,s_0)	(6,6,s_0)	(5,5,s_2)	(6,8,s_0)	(5,8,s_0)	(4,9,s_2)	(5,10,s_0)
6	s_1	(12,0,s_0)	(11,1,s_0)	(10,1,s_0)	(9,3,s_0)	(8,3,s_0)	(7,3,s_0)	(6,4,s_0)	(5,5,s_0)	(4,6,s_0)	(5,9,s_0)	(4,9,s_0)	(3,10,s_2)
6	s_2	(12,0,s_1)	(11,0,s_1)	(10,0,s_1)	(9,2,s_1)	(8,2,s_1)	(7,4,s_1)	(6,5,s_1)	(5,6,s_1)	(4,7,s_1)	(3,8,s_1)	(2,8,s_1)	(3,8,s_1)
6	s_3	(12,0,s_1)	(11,0,s_1)	(10,1,s_1)	(9,1,s_1)	(8,3,s_1)	(7,4,s_1)	(6,5,s_1)	(5,6,s_1)	(4,7,s_1)	(3,7,s_1)	(4,9,s_1)	(3,9,s_1)

the first k bits of $\mathbf{r}$ for $0 \leq k \leq |\mathbf{r}|$. Furthermore, the k-th survivor path at state s_a and time t in the trellis comes from one of the survivor paths at time $t-1$ called the *prefix survivor*. The arrows in Figure 6.6 show the prefix ancestors for the 12 survivor paths at state s_1 and time $t = 3$. For instance, the survivor state and prefix survivor for the third element of $M(s_1, 3)$ are s_0 and 1, respectively. In other words, the survivor path to obtain the first two bits of $\mathbf{r}$ at state s_1 and time $t = 3$ is the concatenation of the survivor path to obtain the first bit of $\mathbf{r}$ at state s_1 and time $t = 2$ with the bits of the branch $s_0 s_1$ (11). The prefix survivors can easily be calculated at the same time as the generalized k-prefix synchronization distances when applying Lemma 6.13. They must be stored by the algorithm, otherwise it won't be possible to backtrack to find the final survivor at the end of the decoding process. For the received sequence 01 01 11 11 01 0, the twelve metrics, prefix survivors, and survivor states for all the states in the trellis are included in Table 6.2.

Let C be the set of all codewords generated by the convolutional encoder, and let $M(s_0, t)[|\mathbf{r}|]$ be the last element of the metric tuple $M(s_0, t)$ corresponding to the number of insertions and deletions required to change the best codeword of length tn into the received sequence. Since $\mathbf{r}$ can be obtained by inserting $|\mathbf{r}|$ symbols in the empty string ϵ, and since the codewords longer than $2r$ are at least $|\mathbf{r}|$ deletions away from $\mathbf{r}$, it follows that $pr_s^{|\mathbf{r}|}(\mathbf{r}, C) \leq |\mathbf{r}|$, hence the algorithm can stop at time $t = 2\lceil \frac{|\mathbf{r}|}{n} \rceil$ in the worst case (in practice the algorithm stops much before that). Once the algorithm stops increasing the length of the codewords, it must backtrack starting from the point where $M(s_0, t)[|\mathbf{r}|]$ is minimized to find the final survivor. In the example presented in Table 6.2, $M(s_0, 5)[|\mathbf{r}|] = 3$ is the minimum value. It follows that the algorithm can stop at $t = 6$, because all the codewords longer than 12 are at least three deletions away from $\mathbf{r}$. The backtracking process is illustrated by boxed metrics in the table. The metric for state s_0 at time $t = 5$ is $(3, 8, s_0)$, thus the algorithm must backtrack to the ninth survivor at state s_0 and $t = 4$, and so on so forth. The final survivor is 00 11 01 11 00, and the algorithm was able to correct the three synchronization errors.

Theorem 6.15. *The final survivor $\hat{\mathbf{v}}$ of the modified Viterbi algorithm for synchronization errors is the codeword whose synchronization distance with the received sequence is minimized.*

Proof. From Lemmas 6.12, 6.13, and 6.14, the survivor path associated to the last element of $M(s_0, t)$ corresponds to the codeword of length nt whose synchronization distance with the received sequence is minimized. The result follows directly from the fact that the

algorithm chooses the length of the codeword such that the last element of $M(s_0, t)$ is minimized. □

It should come as no surprise that the computational complexity of the algorithm is higher than the computational complexity of the original Viterbi decoding algorithm, mostly because the complexity of calculating $pr_s(\mathbf{u}, \mathbf{v})$ is $2 \cdot |\mathbf{u}| \cdot |\mathbf{v}|$. Since the generalized prefix synchronization distance is calculated recursively, it requires $2 \cdot 2^k \cdot |r| \cdot n$ operations per state for a (n, k, v) convolutional encoder instead of $n \cdot 2^k$ for the original Viterbi algorithm, thus the modified algorithm for synchronization errors is $2 \cdot |\mathbf{r}|$ times computationally more expensive. The algorithm needs to store $|\mathbf{r}| + 1$ metrics, survivor states and prefix survivor for each state during the decoding process, and as a result the amount of space required by the algorithm for synchronization errors is also approximately $3 \cdot |\mathbf{r}|$ times the space needed by the original algorithm. Finally, the algorithm must store the received sequence for the entire duration of the decoding process.

6.4 Bypasses and Shortcuts

Convolutional codes used in conjunction with the decoding algorithms described in the previous section are ill-suited to correct synchronization errors and as a result their performance is mediocre. In this section, I define bypasses, and shortcuts in graphs and show that there is a tradeoff between them. This leads to a family of graphs whose structure is very robust against synchronization errors. These graphs can be used as encoders in place of convolutional codes and can be decoded with the modified versions of the Viterbi algorithm.

Definition 6.16. An *k-graph* is a graph $G = (V, \Sigma)$ with the following properties:

1. G is oriented and strongly connected;
2. There are exactly k outgoing edges from each vertex $v \in V$;
3. It is possible to have several edges joining the same vertices (Σ can be a multiset);
4. Edges (v_i, v_i) are possible (cycles of length one).

Convolutional codes are a special case of trellis codes where the structure and the bits of the state diagram of the encoder are defined by a convolution. k-graphs can also be used as encoders by assigning information and output symbols to their edges. A

(k, n)-*encoder graph* is a k-graph with n output symbols associated to each of its edges. The (k, n)-*encoder graphs* can also include input symbols, as shown in the $(2, 2)$-encoder graph of Figure 6.7.

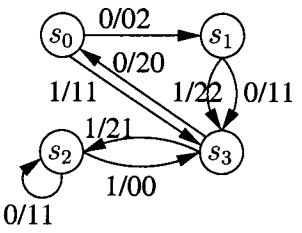

Figure 6.7: A (2,2)-encoder graph with input alphabet $\{0, 1\}$ and output alphabet $\{0, 1, 2\}$.

Definition 6.17. A graph G has a *bypass* of length k if there exists two nonidentical paths of the same length k between a pair of vertices of G. We use b_{min} for the length of the smallest bypass of G. A graph G has a *shortcut* of length k if there exists two paths of different length between a pair of vertices such that k is the difference between the length of the longest path and the length of the shortest path. It is assumed that the length of the shortest path can be zero, thus a cycle of length k is also a shortcut of length k. We use s_{min} to denote the length of the smallest shortcut of G. As an example, the graph in Figure 6.8 has, among others, a shortcut of length one between the vertex s_1 and itself from the cycle of length one, a bypass of length two between the vertices s_0 and s_3 from the paths $s_0s_1s_3$ and $s_0s_2s_3$, and a shortcut of length 2 between s_0 and s_3 from the paths $s_0s_1s_1s_3$ and s_0s_3, For this graph, $b_{min} = 2$ and $s_{min} = 1$.

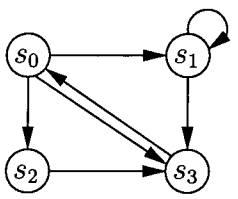

Figure 6.8: A simple graph with bypasses and shortcuts.

There is a very nice tradeoff between the length of the smallest shortcut and the length of the smallest bypass of encoder graphs, as shown by Theorem 6.18.

Theorem 6.18. *Let G be an k-graph with n vertices. If $s_{min} \geq b_{min}$, then*

$$s_{min} \cdot k^{b_{min}-1} \leq n.$$

Proof. Let s_0 be an arbitrary vertex in an oriented graph G. We dispose all the vertices of G in columns, with s_0 in column 0, such that a vertex s is put in column i if and only if the length of the smallest path from s_0 to s is i. For a fixed s_0, there is one and only one way to dispose the vertices of G in columns as described since each vertex belongs to exactly one column.

It should be clear that there can be no edges going from column i to column $i+\alpha$ for $\alpha > 1$. Furthermore, if there is an edge (s_i, s_i') joining two vertices s_i and s_i' in column i, then G has a shortcut of length 1 since there is at least a path of length i and a path of length $i+1$ from s_0 to s_i'. Similarly, if there is an edge from a vertex s_i in column i to a vertex $s_{i-\alpha}$ in column $i-\alpha \geq 0$, then it follows that G has a shortcut of length $\alpha+1$.

Let G be a k-graph with $b_{min} \triangleq b$ and $s_{min} \triangleq s$ and whose vertices are disposed in columns as described two paragraphs ago. First, from the constraint s, no outgoing edge coming from a vertex in the first $s-1$ columns can point to a vertex in the same column or in a preceding column, thus all the outgoing edges from vertices in column i for $0 \leq i \leq s-2$ go to vertices in column $i+1$. Second, from the constraint b and from the fact that each vertex has exactly k outgoing edges, it follows that there are exactly k^i vertices in column i for $0 \leq i \leq b-2$ and at least k^{b-1} vertices in column j for $b-1 \leq j \leq s-1$. Indeed, if column j for $b-1 \leq j \leq s-1$ contains less than k^{b-1} vertices, then there is a vertex in column $j-b+1$ and a vertex in column j with at least two different paths of length $b-1$ between them, which means that G has a bypass of length $b-1$. A similar argument can be made for the first $b-1$ columns. Hence, the total number of vertices in the first s columns is

$$\begin{aligned} n' &\geq 1 + k + k^2 + \cdots + k^{b-2} + (s-b+1) \cdot k^{b-1} \\ &= \frac{k^b - 1}{k-1} + (s-b) \cdot k^{b-1}. \end{aligned} \tag{6.5}$$

We now prove the theorem by contradiction. Let $s \cdot k^{b-1} > n$. We show that there is no possible way to dispose the outgoing edges from column $s-1$ and to the satisfy the constraints b and s while adding only $n - n'$ additional vertices in the graph. It is assumed that there are exactly 2^{b-1} vertices in columns i for $b-1 \leq j \leq s-1$ (it is not difficult to prove that the argument is also true if those columns contain more vertices).

Since there is a limited number of vertices that can be put in column i for $i > s-1$, we need to point as many outgoing edges as possible from column $s-1$ to column 0, as many outgoing edges as possible from column s to column 1, etc. Although it could be decided to wait until column $i > s-1$ before starting to point outgoing edges towards the first columns of G, this does not help because the outgoing edges from a column cannot point to vertices in more than one of the first s columns without creating a shortcut smaller than s.

From the constraint b, there can be at most k vertices in column $s-b+1$ whose respective set of descendants in column $s-1$ are disjoint. Since column 0 has only one vertex, it means that at most k outgoing edges, one from each set, can go from column $s-1$ to column 0. Otherwise, there would more than one path of length $b-1$ from a vertex in column $s-b+1$ to the vertex s_0 in column 0, contradicting the fact that the length of the smallest bypass of G is b. It follows that the remaining $k^{b-1}-1$ outgoing edges from the vertices of each set in column $s-1$ can only point to distinct vertices in column s, hence at least $k^{b-1}-1$ vertices have to be placed in column s. Using a similar argument for all the subsequent columns of G, it follows that the number of vertices required in column $s+i$ for $0 \leq i \leq b-1$ is at least $k^{b-1}-k^i$, thus the total number of vertices in the last columns is

$$\begin{aligned} n'' &\geq (k^{b-1}-k^0)+(k^{b-1}-k^1)+\cdots+(k^{b-1}-k^{b-1}) \\ &= b\cdot k^{b-1}-\frac{k^b-1}{k-1}. \end{aligned} \tag{6.6}$$

Combining (6.5) and (6.6), the total number of vertices in the graph must be at least as large as

$$n'+n'' \geq s\cdot k^{b-1} > n,$$

which is a contradiction. □

Definition 6.19. Let $k \geq 2$, $b \geq 1$, and $\alpha \geq 1$ be integers, let $i\backslash j \triangleq \lfloor \frac{i}{j} \rfloor$ be the quotient (integer part) of $\frac{i}{j}$, and let δ_i be the Kronecker delta, i.e.,

$$\delta_i \triangleq \begin{cases} 1 & \text{if } i=0; \\ 0 & \text{if } i \neq 0. \end{cases}$$

The (k,b,α)-*rectangular graph* is the k-graph with $n = \alpha\cdot(b-1+\delta_{b-1})\cdot k^{b-1}$ vertices numbered from 0 to $n-1$, and for which the k outgoing edges from vertex i point to the

vertices

$$\left[(i \backslash k^{b-1} + 1) \cdot k^{b-1} + \left[i + j \cdot k^{i \backslash k^{b-1} (\text{mod } b-1+\delta_{b-1})}\right] (\text{mod } k^{b-1})\right] (\text{mod } n)$$
$$\text{for } 0 \leq j \leq k. \qquad (6.7)$$

It is not hard to prove that the (k, b, α)-rectangular graphs can be disposed in $\alpha \cdot (b - 1 + \delta_{b-1})$ columns of k^{b-1} vertices such that all the outgoing edges from vertices in column i point to vertices in column $i + 1 \pmod{\alpha \cdot (b - 1 + \delta_{b-1})}$. In fact, the graphs contain the sets of edges

$$\{(\beta, \beta + k^{b-1}), (\beta + k^{b-1}, \beta + 2 \cdot k^{b-1}), (\beta + 2 \cdot k^{b-1}, \beta + 3 \cdot k^{b-1}), \ldots,$$
$$(\beta + (\alpha \cdot (b - 1 + \delta_{b-1}) - 1) \cdot k^{b-1}, \beta)\}$$

for $0 \leq \beta < k^{b-1}$, so they can also be disposed as k^{b-1} parallel cycles, with additional edges between the cycles. The (4,1,3)-rectangular graph, the (3,2,4)-rectangular graph, and the (2,3,2)-rectangular graph are shown in Figures 6.9, 6.10, and 6.11, respectively. Rectangular graphs have several interesting properties, one of them being that they reach the bound of Theorem 6.18, as shown next.

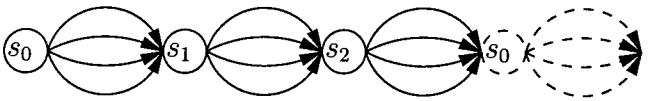

Figure 6.9: The (4,1,3)-rectangular graph.

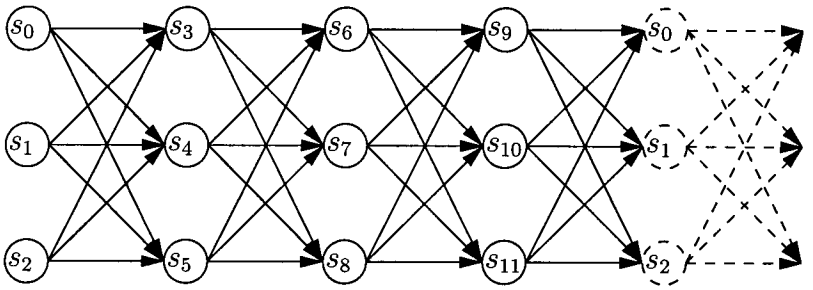

Figure 6.10: The (3,2,4)-rectangular graph.

Theorem 6.20. *The bound from Theorem 6.18 is tight for every (k, b, α)-rectangular graph. More precisely, $b_{min} = b$ and $s_{min} = \alpha \cdot (b - 1 + \delta_{b-1})$.*

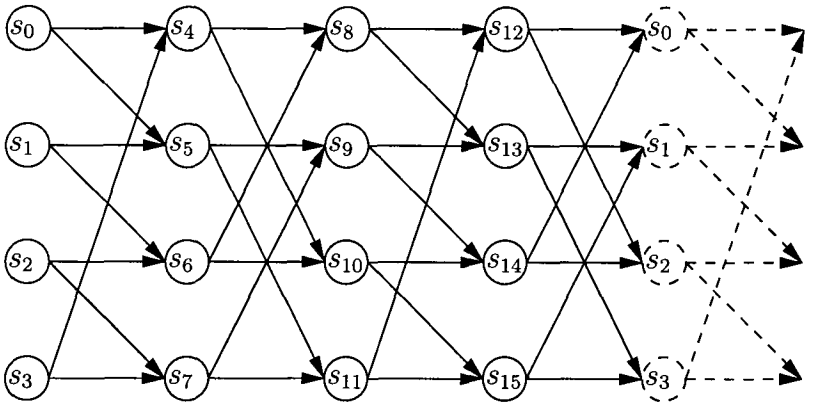

Figure 6.11: The (2,3,2)-rectangular graph.

Proof. From the discussion above, the vertices of a (k, b, α)-rectangular graph can be arranged in $\alpha \cdot (b-1+\delta_{b-1})$ columns such that outgoing edges from vertices in one column all point to vertices in the next column, thus the graph has no shortcut of length smaller than $\alpha \cdot (b - 1 + \delta_{b-1})$. Moreover, since the graph has a cycle of length $\alpha \cdot (b - 1 + \delta_{b-1})$, it follows that $s_{min} = \alpha \cdot (b - 1 + \delta_{b-1})$.

Using s_{min} in Theorem 6.18, we can write that

$$\begin{aligned} s_{min} \cdot k^{b_{min}-1} &\leq n \\ \alpha \cdot (b - 1 + \delta_{b-1}) \cdot k^{b_{min}-1} &\leq \alpha \cdot (b - 1 + \delta_{b-1}) \cdot k^{b-1} \\ b_{min} &\leq b, \end{aligned}$$

thus it remains to be shown that the graph has no bypass of length smaller than b. This certainly is the case for $b = 1$, so for the rest of the proof we assume that $b > 1$. Furthermore, since the (k, b, α)-rectangular graph is, loosely speaking, the concatenation of α copies of the $(k, b, 1)$-rectangular graph, we assume without loss of generality that $\alpha = 1$. The graph can therefore be disposed in $b - 1$ columns numbered from 0 to $s - 2$, with Column i containing the set of vertices

$$\{0 + i \cdot k^{b-1}, 1 + i \cdot k^{b-1}, \ldots, k^{b-1} - 1 + i \cdot k^{b-1}\}$$

whose edges all point to vertices in column $i + 1 \pmod{b-1}$. To simplify the proof, we identify the vertices in a column by their value modulo k^{b-1}, so each column has the set of vertices $\{0, 1, 2 \ldots, k^{b-1} - 1\}$. We have to prove that, from any vertex j in column i, there is at most one path of length β ending in state l in column $i + \beta \pmod{b-1}$ for

$0 \leq i < b-1$, $0 \leq j < k^{b-1}$, and $1 \leq \beta < b$. First, if $i = 0$, then from (6.7) there are k^β paths of length β from j to the vertices

$$\{j \pmod{k^{b-1}}, j+1 \pmod{k^{b-1}}, \ldots, j+k^\beta - 1 \pmod{k^{b-1}}\} \tag{6.8}$$

in Column $\beta \pmod{b-1}$. The paths all end in different vertices if $1 \leq \beta < b$, hence there is no bypass of length smaller than b starting from the first column of the graph. Second, if $i > 0$ and $1 \leq \beta \leq b - i$, then from (6.7), there are k^β paths of length β from j to the vertices

$$\begin{aligned}\{&j \pmod{k^{b-1}}, j+k^i \pmod{k^{b-1}}, j+2\cdot k^i \pmod{k^{b-1}}, \ldots, \\ &j+(k^\beta-1)\cdot k^i \pmod{k^{b-1}}\}\end{aligned}$$

in Column $i+\beta \pmod{b-1}$. The paths of length $b-i$ from j in column $i > 0$ end back in Column 0, and from (6.8), increasing the length of the paths from β to $\beta+\gamma$ will result in $k^{\beta+\gamma}$ paths starting from the vertex j in column i and ending in Column γ in the vertices

$$\begin{aligned}\{&\\ &j \pmod{k^{b-1}}, j+1 \pmod{k^{b-1}}, \ldots, j+k^\gamma-1 \pmod{k^{b-1}},\\ &j+k^i \pmod{k^{b-1}}, j+k^i+1 \pmod{k^{b-1}}, \ldots, j+k^i+k^\gamma-1 \pmod{k^{b-1}},\\ &\ldots,\\ &j+(k^\beta-1)\cdot k^i \pmod{k^{b-1}}, j+(k^\beta-1)\cdot k^i+1 \pmod{k^{b-1}}, \ldots,\\ &\quad j+(k^\beta-1)\cdot k^i+k^\gamma-1 \pmod{k^{b-1}}\\ &\}.\end{aligned}$$

The paths all end in different vertices when $\beta+\gamma < b$, hence there is no bypass of length smaller than b in the graph. □

6.5 Tradeoff Between Recovering Synchronization and Correcting Synchronization Errors

So far, I have proved that the Viterbi decoding algorithm can be used to correct insertion, deletion, and duplication errors efficiently, as well as more complex scenarios where

several types of synchronization errors can occur together. In this section, I explain why convolutional codes are are inefficient to correct synchronization errors and show that the (k, b, α)-rectangular graphs presented in the previous section are more promising, but before that I revisit some of the distances presented in Section 6.2.

Convolutional codes and trellis codes in general generate a infinite number of finite codewords of various lengths, but for synchronization errors the length of the codewords is not necessarily preserved by the channels. Furthermore, decoding errors have different consequences whether or not the decoded and original codewords have the same length, so it is important to distinguish between both cases if some insight on the performance of synchronization-correcting codes is to be gained.

Definition 6.21. A *shortcut decoding error* occurs when the final decoded received sequence and the original codeword do not have the same length. A *bypass decoding error* occurs when a received sequence is decoded incorrectly to a codeword whose length is the same as the length of the original codeword. The *minimum bypass deletion distance* of C is defined as

$$d^b_{min} = \min_{\mathbf{u},\mathbf{v}\in C, |\mathbf{u}|=|\mathbf{v}|} d_d(\mathbf{u}, \mathbf{v}),$$

and the *minimum shortcut deletion distance* is defined as

$$d^s_{min} = \min_{\mathbf{u},\mathbf{v}\in C, |\mathbf{u}|\neq|\mathbf{v}|} d_d(\mathbf{u}, \mathbf{v}).$$

The minimum bypass and shortcut distances can be defined similarly for other types of errors, i.e., i^b_{min} and i^s_{min} for insertion errors, t^b_{min} and t^s_{min} for duplication errors, and s^b_{min} and s^s_{min} for synchronization errors.

A very useful distance measure to study the performance of convolutional codes is the minimum free distance, which can be modified for synchronization errors.

Definition 6.22. The minimum free deletion distance of a code is defined as

$$d_{free} = \min(d^s_{min}, d^b_{min}),$$

and the minimum free insertion, duplication, and synchronization distances are defined the same way and denoted by i_{free}, t_{free}, and s_{free}, respectively. The minimum free distances are equivalent to the minimum distances defined in Section 6.2.

The minimum bypass and shortcut distances both affect the minimum free distance of a code, but in a different way: a shortcut decoding error causes a loss of synchronization

between the transmitter and the receiver, whereas a bypass decoding error maintains synchronization but creates a burst of substitution errors. The main result of this chapter is that the robustness of trellis codes against most types of synchronization errors depends on the structure of the codes themselves. Although for clarity purposes the remaining part of the section only considers deletion errors, the analysis and results are also valid for insertion and synchronization errors.

Theorem 6.23. *For a (k, n)-encoder graph whose smallest bypass is of length b_{min} and whose smallest cycle is of length s_{min},*

$$\begin{aligned} d^b_{min} &\leq n \cdot b_{min}; \\ d^s_{min} &\leq n \cdot s_{min}. \end{aligned}$$

Proof. Let $\mathbf{b}$ be the output symbols from a cycle of length s_{min} in the encoder graph. Clearly, there exist two codewords $\mathbf{c} = \mathbf{l} \circ \mathbf{b} \circ \mathbf{r}$ and $\mathbf{c'} = \mathbf{l} \circ \mathbf{r}$ that can be generated by the encoder, and the deletion distance between them is $|\mathbf{c}| = n \cdot s_{min}$. Let $\mathbf{d}$ and $\mathbf{d'}$ be the output symbols for the two paths of a bypass of length b_{min} in the encoder graph. There exists two codewords $\mathbf{c} = \mathbf{l} \circ \mathbf{d} \circ \mathbf{r}$ and $\mathbf{c'} = \mathbf{l} \circ \mathbf{d'} \circ \mathbf{r}$ that can be generated by the encoder, and the deletion distance between them is at most $|\mathbf{d}| = |\mathbf{d'}| = n \cdot b_{min}$. □

The structure of the underlying graphs of (n, k, v) convolutional codes is terrible because they have a cycle of length one and thus $d^s_{min} \leq n$. It means that no matter how much memory they have, there will not be able to correct all the error patterns consisting of n symbol deletions. Interestingly, the graphs of convolutional codes have a minimum bypass distance of $d^s_{min} = n+1$, and as such they reach the bound of Theorem 6.18. This being said, there is no point for the final codeword to have the symbols of the original codeword in the right order if some of them are missing. It is much better to recover synchronization correctly and to correct potential bursts of substitution errors using, for instance, concatenated Reed-Solomon or low-density parity-check codes.

Theorem 6.18 means that when the number of states of an encoder is fixed, there is a tradeoff between its capacity to recover synchronization and its capacity to output the bits of the original codeword in the right order (of course some graphs are bad for both). One extreme case includes the convolutional codes, and the other extreme case includes the graphs consisting of one long cycle, like in Figure 6.9, where codewords of correct length but bearing little resemblance to the original codewords can be returned.

The (k, b, α)-rectangular graphs are promising for two reasons. First, they achieve

the bound of Theorem 6.18 for $s_{min} \geq b_{min} - 1$, which is required to construct good codes for most types of synchronization errors. The second reason why they are so interesting is that all the edges in one column point to vertices in the next column. As a result, their decoding complexity depends on the number of vertices per column instead of the total number of vertices. Furthermore, their minimum shortcut distance increases as the number of columns increases, whereas their minimum bypass distance increases with the number of vertices per column. Consequently, one can use graphs with a large number of small columns for increased robustness against loss of synchronization without increasing the decoding complexity, and use a concatenated code to correct potential bursts of substitution errors.

6.6 A Few Simulation Results

In this section, I present a few very simple trellis encoders and their performance to illustrate the potential of rectangular graphs to correct all types of synchronization errors.

6.6.1 A Trivial Triple-Deletion Synchronization-Recovering Code

The first encoder is a $(2, 2)$-encoder graph over the (2,2,2)-rectangular graph illustrated in Figure 6.12 and analyzed for deletion errors. Its encoding rate is $\frac{1}{2}$, and two output bits are required to end the encoding in state s_0 when used with input sequences of odd length. The minimum bypass deletion distance of the encoder is one, whereas its minimum shortcut deletion distance is three. The code can therefore correct one deletion error and recover synchronization if less than four deletions occur, whether or not the errors occur in burst. This is as good as it gets for $R = \frac{1}{2}$ binary codes over the (2,2,2)-rectangular graph: it has a cycle of length 4, and it is not possible to select the bits in such a way that the minimum bypass deletion distance is greater than 1.

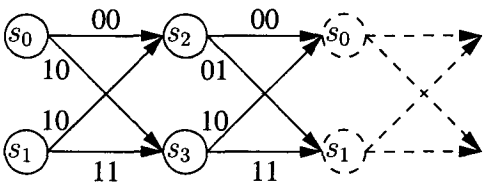

Figure 6.12: A triple-deletion synchronization-recovering code.

Table 6.3: Hamming distance between the original and final codewords for the trellis encoder of Figure 6.12 with two or three deletion errors.

Hamming distance between original and final codewords	2 deletions	3 deletions
0	90.2%	73.6%
1		
2	8.3%	19.5%
3		
4	1.4%	5.3%
5		
6	$< 0.1\%$	1.4%
7		
8		$< 0.2\%$
9		
10		$< 0.01\%$

The encoder was simulated extensively with input sequences of 19 bits, resulting in codewords of 40 bits decoded using the modified Viterbi algorithm for deletion errors presented in this chapter. Table 6.3 shows the Hamming distance between the original and final codewords when two or three deletion errors occur. The result is that if the trellis code is concatenated with a simple outer Reed-Solomon code capable of correcting two substitution errors, the resulting code can correct two deletions with probability 0.985 and three deletions with probability 0.931. This might not seem much, but it is, to the best of my knowledge, the first easily decodable code with a reasonable rate that can correct a burst of three deletion errors. Moreover, the code is based on of the simplest nontrivial rectangular graphs and its decoding complexity is the same as the decoding complexity of a $R = \frac{1}{2}$ convolutional code with a 1-bit register.

6.6.2 A Sample Trellis Code for the Deletion Channel

In most practical systems, loss of synchronization can have catastrophic consequences. I now show that rectangular graphs can overcome this problem. Consider, for instance, the $(2, 2)$-encoder graph over the (2,2,8)-rectangular graph illustrated in Figure 6.12. This code was simulated extensively with random binary input sequences of length 503 and codewords of length 1008. The codewords were transmitted over a deletion channel with bit deletion probability 0.1 and decoded using the Viterbi algorithm for deletion errors. For this set of settings, the probability that the transmitted and decoded codewords do

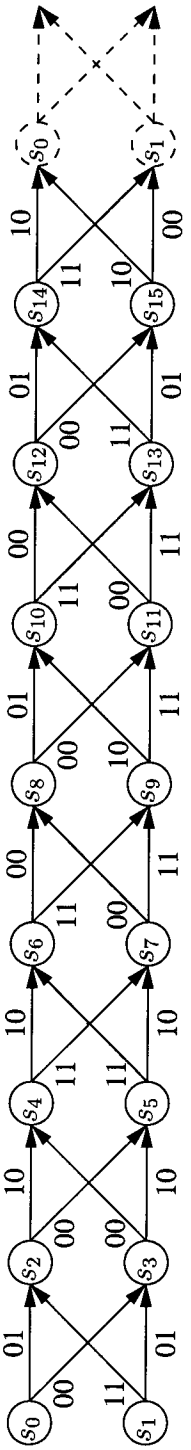

Figure 6.13: A long trellis is very robust against loss of synchronization.

not have the same length is approximately 2×10^{-5}, and the average Hamming distance between the transmitted and received sequences is 67. This means that using this code of rate $\frac{1}{2}$ transforms a channel deleting roughly 10% of the bits into a channel with a 6.7% rate of substitution errors. Again, it must be mentioned that the decoding complexity of the inner code is still the same as that of a $R = \frac{1}{2}$ convolutional code with a 1-bit register. Using a good Reed-Solomon or low-density parity-check outer code, it is possible to correct almost all the substitution errors, thus to obtain a concatenated system with good performance.

On average over all codes, the probability of a shortcut decoding error decreases exponentially with the number of columns in the encoder graph. For instance, over all the binary codes of rate $R = \frac{1}{2}$ using the $(2, 2, 20)$-rectangular graph, the probability of a shortcut decoding error is less than 10^{-12}. This solves one of the most important problems of synchronization, and the only known codes that somewhat achieve this are the watermark codes introduced by Davey and Mackay [20]. The main challenge of trellis codes over rectangular graphs is therefore not to recover synchronization, but instead to find bit configurations minimizing the number of substitution errors between the transmitted and decoded sequences.

6.6.3 Codes for Channels with Segmented Deletion Errors

An interesting problem is to develop codes that can guarantee successful decoding if the synchronization errors or burst of errors are far apart from each other, or if there is a bounded number of errors per segment. Using irregular codes over the $(2, 1, k)$-rectangular graphs and the modified versions of the Viterbi decoding algorithms for deletion or insertion errors, it is possible to obtain codes that significantly outperform the best known codes in this setting. For instance, the almost trivial encoder shown in Figure 6.14 has rate $R = 0.4$ and can correct one deletion error (or one insertion error) every five bits, which is much better than the code with rate $R = 0.25$ from Swart and Ferreira that can correct one deletion error (or one insertion error) every six bits. Moreover, the decoding complexity of my code is equivalent to that of a convolutional code with no memory, whereas, Swart and Ferreira's code is much more complex and much harder to decode.

Liu and Mitzenmacher [64], using another extension of Levenshtein's single-deletion error-correcting codes, designed a code of rate $R = 0.448$ that can correct one deletion error (or insertion error) per byte (segment of eight bits). This is a slightly different

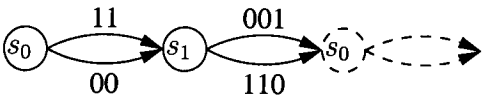

Figure 6.14: A code correcting 1 deletion error every 5 bits.

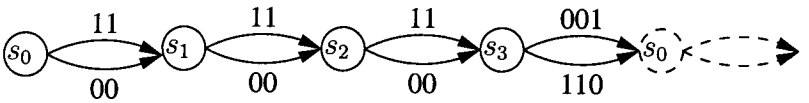

Figure 6.15: A code correcting 1 deletion error every 9 bits.

model, since the code can correct a pair of deletions close to each other if the first deletion occurs near the end of a segment and the second deletion near the beginning of the next segment. The irregular code over the $(2, 1, 4)$-rectangular graph presented in Figure 6.15 has rate $R = \frac{4}{9} \approx 0.444$ and can correct one deletion error every nine bits when the modified Viterbi decoding algorithm for deletion is used. This being said, when the distance between consecutive errors is less than nine, the algorithm will return valid codewords, but sometimes codewords corresponding to illegal error patterns, i.e., patterns with errors too close to each other. This can be solved by allowing the Viterbi algorithm to backtrack when an illegal error pattern is found until a codeword corresponding to a valid error pattern can be returned, which significantly improves the accuracy of the decoded codewords. It should be noted that the complexity of the algorithm increases as the average distance between the errors decreases, since it has to backtrack more often.

The last code of this section is shown in Figure 6.16. Its rate is $R = 0.3$ and it can correct a burst of 2 deletion errors (or 2 insertion errors) every 10 bits if the Viterbi algorithm for deletion or insertion errors is used without backtracking. More precisely, decoding will be successful if any substring of 10 bits in a codeword is corrupted by at most 2 deletions

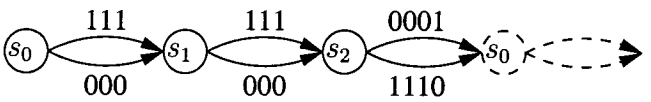

Figure 6.16: A code correcting 2 deletion errors every 10 bits.

6.7 Regarding Maximum likelihood Decoding of Trellis Codes for Deletion Channels

This has already been mentioned several times throughout this thesis, but again one reason explaining why deletion-correcting codes are difficult to derive compared to codes for substitution errors is that the balls associated to the codewords do not have the same size and depend on the structure of the codewords themselves. We saw in Chapter 4 that the problem of finding the number of subsequences of a string when bits are deleted from it is a challenging problem in its own right. This also explains why the modified Viterbi algorithm with the metric used in Section 6.3 does not necessarily find the maximum-likelihood path. Consider for instance that one of the codewords from the codebook $C = \{101, 001, 0001, 00001\}$ is transmitted over a deletion channel with deletion probability p_d and that the received sequence is 01. One might say that 101 and 001 are equally close from 01 since only 1 bit needs to be deleted from both strings to get 01, but it is not true, since to get 01 from 001 the bit can be deleted among the two bits of the first run, whereas from 101 the first bit must be deleted. More precisely, $P(01|001) = 2p_d(1-p_d)^2$ and $P(01|101) = p(1-p_d)^2$. This is quite different from substitution errors, where all the codewords at equal Hamming distance from a received sequence are equally likely.

Even more challenging is the fact that for a deletion channel where each bit is independently deleted with probability p_d and transmitted correctly with probability $1-p_d$, the maximum-likelihood codeword and even its length change depending on the probability of bit deletion p_d, with longer codewords more likely as the probability of bit deletion increases. Using the codebook C defined above, $P(01|0001) = 3p_d^2(1-p_d)^2$ and $P(01|00001) = 4p_d^3(1-p_d)^2$. Hence, if $0 < p_d < \frac{2}{3}$, then 001 is the maximum-likelihood codeword; if $\frac{2}{3} < p_d < \frac{3}{4}$, then 0001 is the maximum-likelihood codeword; finally if $p_d > \frac{3}{4}$, then the maximum-likelihood codeword is 00001. A direct consequence of this phenomenon is that the maximum-likelihood codeword for a received sequence which is already a codeword can be a different codeword: if 001 is received, then $P(001|001) = (1-p_d)^3$, $P(001|0001) = 3p_d(1-p_d)^3$, and $P(001|00001) = 6p_d^2(1-p_d)^3$, thus 001 is not the maximum-likelihood codeword for itself if $p_d > \frac{1}{3}$.

In general, maximum-likelihood decoding of trellis deletion-correcting codes cannot be done using a survivor metric and the Viterbi algorithm in its simplest form. This is formalized in the following theorem.

Theorem 6.24. *For any survivor metric, there exists a trellis such that the modified Viterbi decoding algorithm for deletion errors is not maximum-likelihood decoding.*

Proof. Consider the encoder corresponding to the simple rectangular trellis shown in Figure 6.17. If the deletion probability is small, the maximum likelihood codeword for the received sequence 0100 is 010100, and for 0101 the maximum-likelihood codeword is 011001. Since the two received sequences start with 010, the state of the Viterbi algorithm at time $t = 1$ is the same for both, and whatever branch is kept as the survivor, at least one of the two final survivors will not be the maximum-likelihood codeword for the corresponding received sequence. □

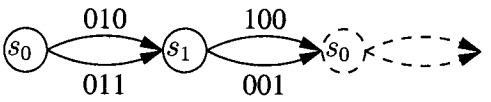

Figure 6.17: A bad trellis for maximum-likelihood decoding.

Under the light of Theorem 6.24, a maximum-likelihood algorithm for trellis codes with deletion errors must have the capability to remember more than one survivor path per state.

6.8 Summary

I this chapter, I have presented a new class of binary codes capable of correcting synchronization errors. The codes use encoders based on graphs, and although convolutional codes do not offer a good performance, I have shown that the structure of rectangular graphs is well suited to be used on channels affected by synchronization errors.

I have formally proved that the Viterbi decoding algorithm can be modified to correct various types of synchronization errors by returning the "closest" codeword from a given received sequence. Correcting one type of synchronization error requires roughly the same complexity as the original Viterbi decoding algorithm, whereas it is not the case when insertions and deletions or synchronization and substitution errors can occur together.

I have demonstrated that codes can recover synchronization with arbitrarily high probability by using encoders based on rectangular graphs with a large number of columns, and this without increasing their encoding and decoding complexities. Finally, I have proved that maximum-likelihood decoding for synchronization errors cannot be achieved

without increasing the number of survivor states in the appropriate versions of the Viterbi decoding algorithm.

Part III

Applications of Synchronization Models

Chapter 7

A Synchronization Approach to Opportunistic Spectrum Access for Cognitive Radio Systems

7.1 Introduction

Cognitive radio is an emerging technology proposed to increase spectrum usage in underutilized licensed spectrum bands. A challenging task for cognitive radio systems is to design efficient opportunistic spectrum access schemes. One way for cognitive users to communicate is to use an interference avoidance approach [49], meaning that the cognitive transmitter and receiver only use the available spectrum in which they don't detect primary user activity. Two difficulties arise in this scenario. First, the availability of the spectrum to the cognitive users depends on the activity of primary users and is therefore highly dynamic. Second, it is possible that the transmitter and the receiver have conflicting information about the available spectrum at any given time due to factors like mobility, distance from the primary users, shadowing, etc.

The channel models developed and described in this chapter are motivated by the two-switch channel model for dynamic and distributed spectrum allocation originally presented by Jafar and Srinivasa [49]. There, the authors model the dynamic spectrum allocation problem as a channel with side-information [93, 16]. The model, illustrated in Figure 7.1, has a switch for each cognitive user which is open if they detect a primary user. The switch requires that the cognitive users avoid using the channel whenever

[1]The work of Section 7.2 was done in collaboration with Chris Nicola.

a primary user is detected. A generalization of the two-switch channel is presented in Figure 7.2, where the available bandwidth B is divided into s subcarriers, each with its own two-switch. Partitioning the bandwidth was proposed after measurements indicated that spectrum bands licensed to primary users were not used uniformly, and the main advantage of this approach is that resources can be allocated to cognitive users without overly interfering with legacy users. In particular, cognitive users must avoid using the subcarrier channels where primary user activity is detected. In this chapter we explore this further by implementing channel models motivated by this scheme. I describe two logical ways in which the cognitive users might approach these channels. In the first approach, the cognitive transmitter always continues with its message, allowing bits to be lost when there are open switches; this can be thought of as an erasure channel. In the second approach, the transmitter stops sending information when it encounters an open switch until the next free (closed switch) subcarrier and then resumes the transmission. This requires that the cognitive receiver attempts to remain synchronized.

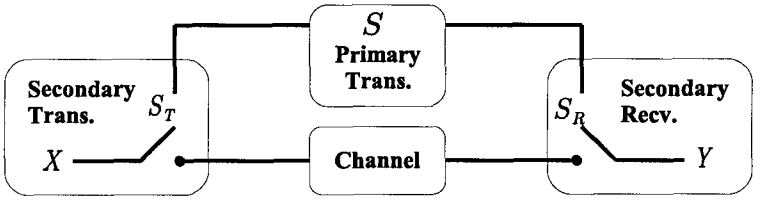

Figure 7.1: Two-switch model from Jafar and Srinivasa.

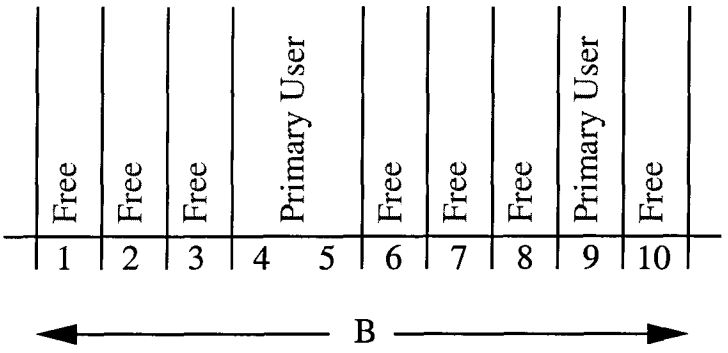

Figure 7.2: Spectrum availability for a cognitive user at time t.

In both models, the detection capabilities of the transmitter and the receiver are not assumed to be ideal, nor do they necessarily match. We define a probability of detection failure and a probability of false alarm for each cognitive user. This gives us some insight

into the effect spectrum sensing quality has on channel capacity and system performance. Another motivation to study these models is to learn about their fundamental limits and to introduce error-correcting coding techniques to solve the dynamic spectrum allocation problem, especially synchronization error-correcting codes. We are aware that some of our assumptions are unrealistic, this being said we believe that our analysis remains valid with assumptions that model practical systems more closely. For instance, although we use coding at the bit level, coding at the packet level is also possible in our setting. Furthermore, we demonstrate that changing erasures into synchronization errors can provide some advantages, which is an interesting result in its own right.

We begin with the noisy erasure model in Section 7.2 and then extend and improve on it to demonstrate that the switch at the receiver is not ideal in this case. Section 7.3 provides the framework for the synchronization approach, and in Section 7.4 we conclude the chapter.

7.2 Opportunistic Spectrum Access via Erasure Models

7.2.1 Two-switch erasure model with binary symmetric noise

Dynamic spectrum allocation by an individual cognitive radio in the presence of a primary user with priority use of the spectrum can be modeled using the two-switch model from Jafar and Srinivasa [49] and seen in Figure 7.1. The two switches model the detection of primary users at the transmitter $\mathcal{T}$, and receiver $\mathcal{R}$, and are represented by the random variables $S_{\mathcal{T}}$ and $S_{\mathcal{R}}$. Each variable is a local estimate of the random variable S which is the actual event that a primary user is operating in the range of $\mathcal{T}$ and/or $\mathcal{R}$. The transmitter is not only required to avoid transmitting in the presence of a primary user, but we also assume that the primary user signal is much more powerful than the channel noise and will result in an near-total loss of information. This is modeled by changing the bit-error probability such that when $S = 0$ (no primary user present), the noise process of the channel is that of the BSC channel with probability of error $p_e^0 = p_{BSC}$, and when $S = 1$, then $p_e^1 = \frac{1}{2}$.

The detection of the primary users by $\mathcal{T}$ and $\mathcal{R}$ is not assumed to be perfect. There is the possibility that they fail to detect an active primary user or that they falsely detect a primary user (false alarm) when none is active. The probabilities of failure at the

transmitter and receiver are represented by:

$$f_T^s = \Pr(S_T \neq s | S = s),$$
$$f_{\mathcal{R}}^s = \Pr(S_{\mathcal{R}} \neq s | S = s).$$

Using this notation, the probabilities of failed detection are represented by f_T^1 and $f_{\mathcal{R}}^1$, and the probabilities of false alarm by f_T^0 and $f_{\mathcal{R}}^0$. These probabilities can be thought of as characteristics of the cognitive radios indicating the ability to detect primary user activity. Furthermore, the probability of failing to detect a primary user is directly related to the probability of interfering with it, so it will very likely be tightly controlled by regulating agencies [45]. It will be seen that these values have a direct effect on the capacity of the dynamic spectrum channel available to the cognitive radios in this model.

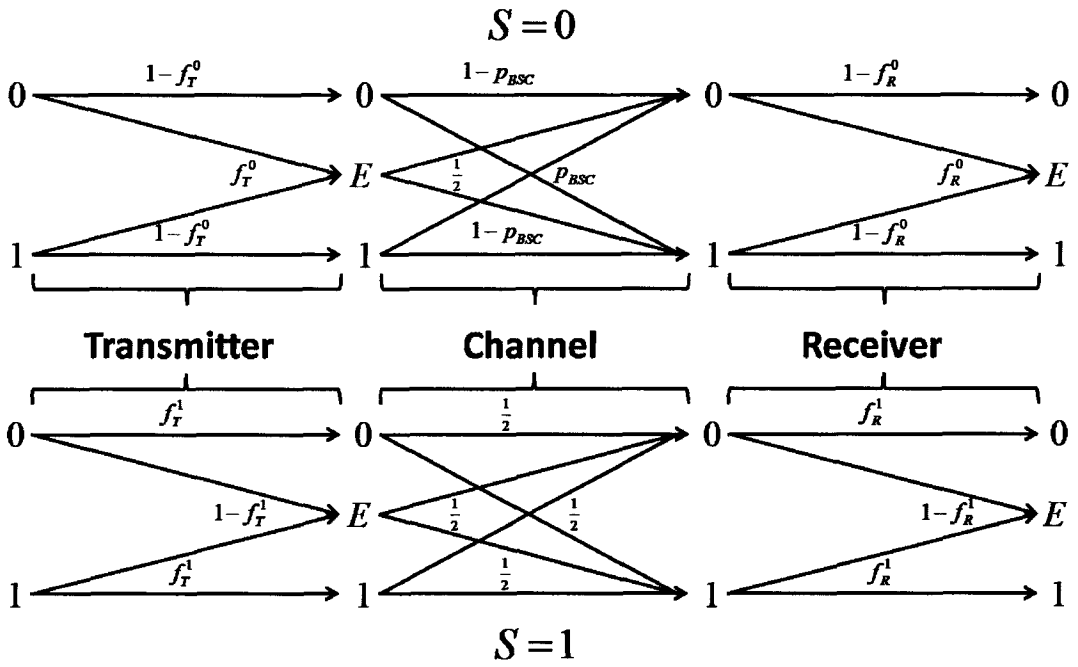

Figure 7.3: Components of the noisy erasure channel

The components of the noisy erasure channel are illustrated in Figure 7.3. While the channel can be considered to have two separate states associated with $S = 0$ and $S = 1$, we can simplify this into the noisy erasure channel shown in Figure 7.4 and express the values ε and α as functions of the probabilities of primary user activity and the detection

properties of $\mathcal{T}$ and $\mathcal{R}$, i.e.,

$$\alpha = (1-p_S)\, f_{\mathcal{R}}^0 + p_S\left(1-f_{\mathcal{R}}^1\right) \text{ and} \tag{7.1}$$

$$\varepsilon = (1-p_S)\left[\frac{1}{2}f_{\mathcal{T}}^0 + \left(1-f_{\mathcal{T}}^0\right)p_{BSC}\right]\left(1-f_{\mathcal{R}}^0\right) + \frac{1}{2}p_S f_{\mathcal{R}}^1, \tag{7.2}$$

where $p_S \triangleq \Pr(S=1)$.

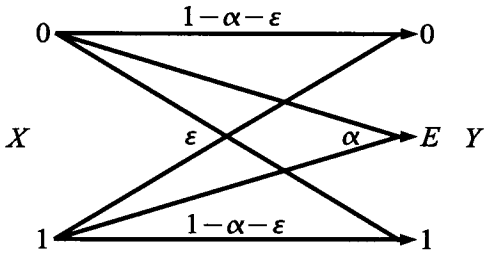

Figure 7.4: Noisy erasure channel.

For this channel the capacity for uniform binary inputs is given by

$$\begin{aligned} C &= 1-\alpha+H([\alpha,1-\alpha]) - H([\varepsilon,\alpha,1-\varepsilon-\alpha]) \\ &= 1-\alpha+h(1-\alpha)-h(\varepsilon)-h(1-\varepsilon-\alpha), \end{aligned} \tag{7.3}$$

where $H(\cdot)$ is the entropy function and $h(x) = -x\log(x)$.

This model, however simple, may not be optimal for this approach to the problem. We show below that it is possible for the receiver to further exploit the knowledge of the imperfect detection properties of $\mathcal{T}$ and $\mathcal{R}$.

7.2.2 One-switch alternative model

The noisy erasure model for the two-switch channel can be considered one of the simplest models for the dynamic spectrum allocation problem in that it is merely a combination of two of the simplest and most well known binary input channels. We note, however, that the receiver may not be exploiting all of the information available from the channel and side-information.

The effect of the switch at the receiver means that it assumes an erasure occurs whenever it detects a primary user. This does not consider the possibility that the primary user may have been detected incorrectly by the cognitive receiver. If the primary user detection is perfect at the receiver then there is no advantage to consider the output

of the channel. However, we have imperfect detection of the primary user, and in this case the receiver can do better if it has an accurate knowledge of the probabilities of failed detection $f_{\mathcal{R}}^{s}$ and $f_{\mathcal{T}}^{s}$.

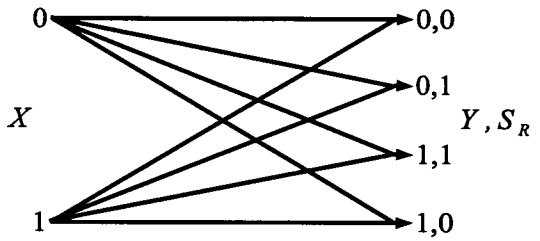

Figure 7.5: One-switch channel transitions.

We consider a modification of the channel model which is illustrated in Figure 7.5, where there are four possible channel outputs corresponding with the two binary output values for the channel Y and the two states of $S_{\mathcal{R}}$. This can be looked at as a one-switch model where there is only a switch at $\mathcal{T}$.

It is possible to compute the channel capacity in this case, and to do so we first define the probability of error conditioned on $S_{\mathcal{R}}$ by $p_e(s_{\mathcal{R}}) \triangleq \Pr(X \neq Y | S_{\mathcal{R}} = s_{\mathcal{R}})$. More precisely,

$$p_e(0) = \frac{1}{2}P_{1|0} + \left[p_{BSC}\left(1 - f_{\mathcal{T}}^{0}\right) + \frac{1}{2}f_{\mathcal{T}}^{0}\right] P_{0|0}, \tag{7.4}$$

$$p_e(1) = \frac{1}{2}P_{1|1} + \left[p_{BSC}\left(1 - f_{\mathcal{T}}^{0}\right) + \frac{1}{2}f_{\mathcal{T}}^{0}\right] P_{0|1}. \tag{7.5}$$

We define $P_{S|S_{\mathcal{R}}} \triangleq \Pr(S = s | S_{\mathcal{R}} = s_{\mathcal{R}})$, and compute

$$P_{0|0} = \frac{(1-p_S)(1-f_{\mathcal{R}}^{0})}{(1-p_S)(1-f_{\mathcal{R}}^{0}) + p_S f_{\mathcal{R}}^{1}},$$

$$P_{0|1} = \frac{(1-p_S) f_{\mathcal{R}}^{0}}{(1-p_S) f_{\mathcal{R}}^{0} + p_S (1-f_{\mathcal{R}}^{1})},$$

with $P_{1|0} = 1 - P_{0|0}$ and $P_{1|1} = 1 - P_{0|1}$. These can be substituted into the equation of the channel mutual information, i.e.,

$$\begin{aligned} I(X; Y, S_{\mathcal{R}}) &= H(X) - H(X|Y, S_{\mathcal{R}}) = 1 - H(X \oplus Y | S_{\mathcal{R}}) \\ &= 1 - \sum_{s_{\mathcal{R}}} \Pr(S_{\mathcal{R}} = s_{\mathcal{R}}) \cdot \left[h(p_e(s_{\mathcal{R}})) + h(1 - p_e(s_{\mathcal{R}}))\right]. \end{aligned}$$

This gives us the capacity of the channel for a uniform input distribution. In the one-switch model the receiver does not assume that no useful information can be received from the channel when a primary user is detected. This is advantageous to $\mathcal{R}$ if it knows that the detection capabilities of both itself and of $\mathcal{T}$ are not ideal. If we assume that $\mathcal{R}$ has an accurate knowledge of the values for $f_{\mathcal{R}}^s$ and $f_{\mathcal{T}}^s$, it is possible for the decoder to exploit this channel further by considering the output value Y. Without that knowledge the receiver may not be able to effectively exploit the information that might have actually been sent in the case of a false detection.

It is worth noting, however, that the capacity gained by considering this model vs. the erasure model is modest unless both the primary user activity is fairly high (i.e., p_S is large) and the probabilities of detection error are large.

7.2.3 Coding for the Erasure Approach

Although in theory coding could be applied to the models described in the previous section in order to compensate for the BSC noise as the primary user activity, the main problem with the erasure approach is that code rates must reflect the probability of primary user activity. In practice, primary user activity will certainly be time-varying and highly bursty, hence it does not seem that coding can be useful in this setting.

7.3 Opportunistic Spectrum Access via Synchronization

In this section, we explore how synchronization error-correcting codes can be used to solve the opportunistic spectrum access problem. As in Section 7.2, we use the same assumptions that allow us to model the problem in an intuitive albeit oversimplistic way: a cognitive transmitter $\mathcal{T}$ wants to convey a message to a cognitive receiver $\mathcal{R}$, the available bandwidth B is divided into s subcarriers with its own two-switch, the cognitive users are not allowed to transmit in the subcarriers used by a primary user, and there is no interference between the subcarriers. The novelty of the synchronization approach over the erasure models is that when an open switch is encountered, the cognitive transmitter keeps the bit to be transmitted until a closed switch is encountered instead of dropping the bit and moving to the next one. Consequently, a discrepancy of the available subcarriers between the secondary transmitter and receiver causes a loss of synchronization, hence

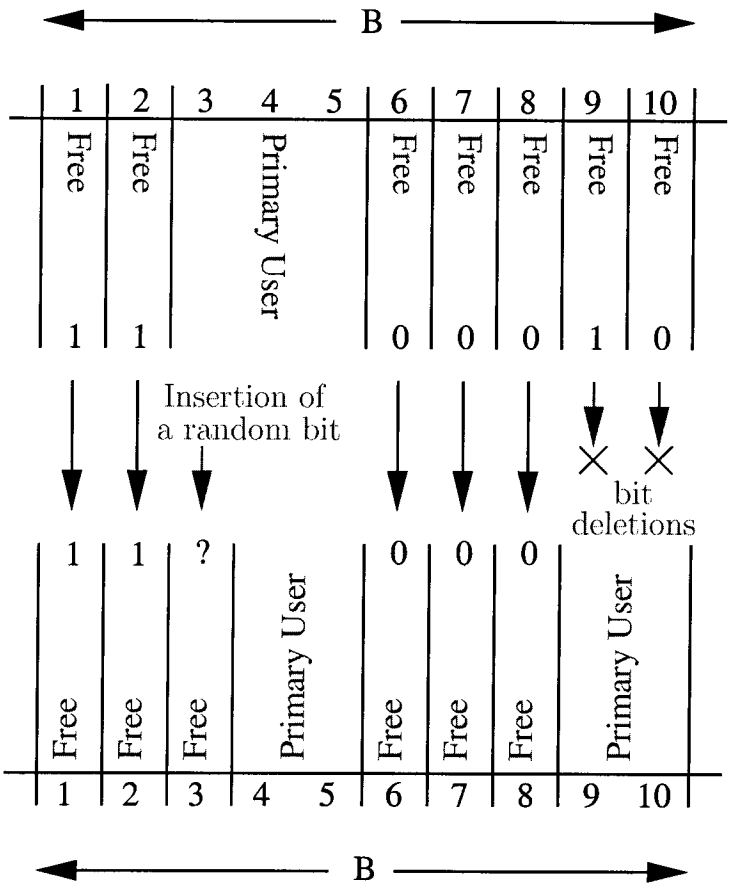

Figure 7.6: Synchronization errors caused by discrepancies of the available spectrum to the cognitive transmitter and receiver.

the opportunistic spectrum access problem can be solved using synchronization error-correcting codes.

In order to illustrate our model, suppose that $\mathcal{T}$ wants to send the string 1100010. It senses the spectrum and realizes for instance that the third, fourth, and fifth subcarriers are being used by primary users, as shown in Figure 7.6. $\mathcal{T}$ then sends a bit on each free subcarrier, and the resulting usage of the bandwidth is 1 1 - - - 0 0 0 1 0. Suppose now that $\mathcal{R}$ senses that the fourth, fifth, ninth, and tenth subcarriers are being used by primary users. If no bit is corrupted by the channel, it follows that the received bit for each subcarrier is 1 1 ? - - 0 0 0 - -, in other words the message 1100010 is received as 11?000. It follows that each primary user detected by $\mathcal{R}$ but not by $\mathcal{T}$ will result in a bit deletion. It has to be noted that $\mathcal{R}$ has side-information about the deletions, since they can only be located in the subcarriers detected as being used by primary users. Furthermore, each primary user only detected by $\mathcal{T}$ generates a random bit insertion at the receiving side, but unlike deletions there is no side-information about their location.

Interestingly, it seems that our study of dynamic spectrum allocation shed new light on channels with synchronization themselves. To argument our statement, consider the case with $f_{\mathcal{T}}^0 = 0$, $f_{\mathcal{T}}^1 = 0$, $f_{\mathcal{R}}^1 = 0$, and $p_{BSC} = 0$. In this scenario, the cognitive users want to communicate over a noise-free channel, and errors only occur when the receiver falsely detects primary user activity on a given subcarrier. From (7.1) and (7.2), $\alpha = (1-p_S)f_{\mathcal{R}}^0 + p_S$ and $\epsilon = 0$, and it follows from (7.3) that the capacity of the resulting channel is

$$\begin{aligned} C &= 1-\alpha \\ &= 1-(1-p_S)f_{\mathcal{R}}^0 - p_S. \end{aligned}$$

At first sight, when p_S is small, this channel seems quite similar to the deletion channel with probability of bit deletion $f_{\mathcal{R}}^0$ presented in Chapter 3. After all, when looking for the value of a missing bit and its position from a string of 10 bits, it does not seem to be a disadvantage to receive a subsequence of nine bits as opposed to an binary array of ten million cells with 9 nonzero entries. This being said, we can divide the previous expression by $(1-p_S)$ to get a capacity expression when the channel is free of primary user activity. This gives the capacity of an erasure channel with erasure probability p_S, i.e.,

$$C' = 1 - f_{\mathcal{R}}^0,$$

which is much higher than the capacity of the corresponding deletion channel. Hence,

by using the synchronization approach, the cognitive users lose the advantage provided by knowing the exact positions of the bits in the codewords and as a result must suffer a loss in data rate proportional to the probability that they disagree whether or not a legacy user is active on a given subcarrier. Furthermore, it goes without saying that the synchronization approach would be much more difficult to implement than the others for all the reasons mentioned throughout this thesis.

Cognitive users in practical settings will have to deal with very dynamic and bursty primary users, and as such the average primary usage is not a very useful measurement. Also, as mentioned previously, their ability to detect primary users will likely be tightly regulated. This means that despite all its drawbacks, the synchronization approach offers a significant advantage over codes for erasure channels or real-time reconciliation of the available spectrum between the cognitive transmitter and receiver: primary activity has no influence whatsoever on the error rate of the channel. Thus, as long as the discrepancy in the spectrum allocation is not larger than the error-correcting capability of the code, $\mathcal{T}$ can transmit on the available subcarriers. If the bandwidth used by primary users changes suddenly, then $\mathcal{T}$ adjusts the subcarriers it uses accordingly and does not need to alert $\mathcal{R}$ of the changes. In most practical systems affected by time noise, synchronization errors have catastrophic consequences and are usually converted into substitution errors or erasures. This is the case, for instance, for packet-switched communication networks, where a preamble is added to each packet to insure that the receiver knows which packets are dropped on their way to destination. Opportunistic access for cognitive radio networks is, to the best of my knowledge, the first problem where converting erasures into synchronization errors can be beneficial.

The models introduced in this chapter are interference-free. We now discuss this assumption and explain why the models remain valid if it is relaxed. First, we assume that there is little interference from the secondary users to the primary users. This is paramount to cognitive radio systems since cognitive users must not interfere with neighboring primary transmissions, probably by using a lower transmit power which has yet to be fixed by regulatory agencies [45]. Bansal, Hossain, and Bhargava [5] studied a closely related model in which the power allocated to the subcarriers decreases as they approach the primary user band. Consequently, one of the techniques proposed for spectrum sensing is energy detection, where the cognitive users measure the energy level in each subcarrier and conclude that those for which the energy level as above a certain threshold are used by primary users. Second, we assume that there is no interference between the subcarriers used by $\mathcal{T}$ when the bandwidth is partitioned; this

can be approached by using orthogonal frequency division multiplexing (OFDM). Third, we assume that there is no interference from the primary users in the subcarriers used by the cognitive users. This is of course unrealistic, especially for cognitive subcarriers located close to the ones used by legacy users. However, the interference to the cognitive users from the primary users can be modeled as additive white Gaussian noise [5]. This is not a problem for synchronization-correcting codes, since and a code capable of correcting insertions and deletions can also correct substitutions of bits (a substitution can be viewed as the deletion of a bit followed by the insertion of the opposite bit at the same position). Good codes capable of correcting a small number of insertion and deletion errors while remaining robust against additive noise could be useful in this setting.

7.4 Summary

In this chapter, we have developed simple models for which we can apply the techniques of error-correcting coding, especially synchronization error-correcting codes, as a means of compensating for primary user activity in dynamic spectrum allocation applications. Using simple and well understood channel models has provided us with some tools and insight to develop approaches to design dynamic spectrum systems. It remains to be seen whether our approach can still be useful when more practical assumptions are used.

Chapter 8

Worst-Case Nonzero-Error Interactive Communication Applied to Distributed Problems with Synchronization Errors

8.1 Introduction

Interactive communication was introduced by Orlitsky [80] and lies at the intersection of information theory and communication complexity. It examines the amount of communication required for one party to convey information to a second party who has correlated information. Two players, an *informant* $P_{\mathcal{X}}$ and a *recipient* $P_{\mathcal{Y}}$, possess private but correlated inputs x and y, respectively. $P_{\mathcal{Y}}$ wants to learn his interlocutor's input without error while minimizing the number of bits that need to be transmitted in the worst case. To do so, $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ alternately exchange data on a noise-free channel following a deterministic protocol they have agreed upon initially. Unlike the original communication complexity model [111] (see the book by Kushilevitz and Nisan [57] for an exhaustive survey), the function to be computed is trivial ($f(x, y) = x$), but the problem is to exploit the correlation between the parties' knowledge for reducing the required amount of communication. The following example illustrates the model.

[1] I started working on nonzero-error interactive communication at the end of my Master of Science. Lemmas 8.9, 8.13, 8.17, and part of Theorem 8.29 were proved in my Master's Thesis [67], but I include them in this chapter to present a complete survey of the subject.

Example 8.1. The League Problem [80]

A sports league has 2^n teams, and the name of each team is a binary string of n bits. $P_{\mathcal{Y}}$ knows the two teams playing in the championship match, but a blackout during the game prevents him from learning who wins. $P_{\mathcal{X}}$, on the other hand, hears the name of the champion team on the radio but has no idea who the runner-up is. $P_{\mathcal{Y}}$ wants to learn the identity of the champion team from $P_{\mathcal{X}}$ with certainty while minimizing the communication required.

If only one round of communication from $P_{\mathcal{X}}$ to $P_{\mathcal{Y}}$ is allowed, then $P_{\mathcal{X}}$ has to transmit the n bits of the winning team. Indeed, if fewer than n bits are transmitted, there are two teams for which $P_{\mathcal{X}}$ sends the same message; if those two teams happen to play in the championship match, then $P_{\mathcal{Y}}$ is not able to learn the winner with certainty. However, a substantial gain can be achieved when interaction is allowed. $P_{\mathcal{Y}}$ sends the position of one of the bits where the names of the two finalists differ, which requires $\lceil \log n \rceil$ bits. It is then sufficient for $P_{\mathcal{X}}$ to send the bit of the winning team at the required position. This protocol requires $\lceil \log n \rceil + 1$ communication bits, an exponential gain compared to the one-way protocol. Orlitsky showed that even if more than two rounds of communication are allowed, no protocol can solve this problem by exchanging a smaller number of bits in the worst case.

In this chapter, I study worst-case nonzero-error interactive communication and compare the results with the original worst-case deterministic model. I allow P_Y to learn P_X's input with a probability of error of at most ϵ and study how it can improve the communication, either by reducing the number of bits that need to be exchanged or by reducing the number of rounds of communication. Four nonzero-error models are presented. The first model, *worst-case private coin randomized interactive communication*, allows $P_{\mathcal{Y}}$ to learn x with a probability of at least $1 - \epsilon$ for of all the possible input pairs (x, y). The players can also use randomized protocols: each player has a private, independent source of randomness whose output can be used to decide which bits should be transmitted. The second model, *worst-case public coin randomized interactive communication*, also allows $P_{\mathcal{Y}}$ to learn $P_{\mathcal{X}}$'s input with a probability of at least $1-\epsilon$ for all of the possible input pairs. It also uses randomized protocols, but instead of private coins, both players can use a public (common) random generator. The third model, *worst-case private coin randomized amortized interactive communication*, allows the players to solve several independent instances of the same problem simultaneously instead of sequentially. The players are again permitted to use private coins, and $P_{\mathcal{Y}}$ can fail to learn x with a probability of at most ϵ for every input pair. The fourth model, *worst-case distribu-*

tional interactive communication, permits only deterministic protocols, but $P_{\mathcal{Y}}$ can learn x incorrectly for a fraction of at most ϵ of all the inputs weighted by their probability distribution.

I prove that the worst-case public coin randomized, private coin randomized amortized and distributional models are equivalent, and that optimal protocols for the three models do not require interaction between the players. The models can be arbitrarily better than the worst-case deterministic model when a single round of communication from $P_{\mathcal{X}}$ to $P_{\mathcal{Y}}$ is allowed.

Interactive communication includes a large class of symmetric problems [83] inherent to several practical applications, including synchronization of mobile data [100], reconciliation of sequences of symbols such as nucleotide sequences in DNA molecules [30], remote data storage [15], list decoding with side information [42], and quantum key distribution [10]. I prove that the deterministic and all of the nonzero-error models are equivalent and efficient for symmetric problems, although optimal protocols requiring only one round of communication can be constructed when errors and randomization are allowed. I also show that all of the models are equivalent and inefficient for Cartesian-product pairs.

The most challenging model to analyze is the worst-case private coin randomized model. I show that the best one-round protocols for this model are at most three times more expensive than the best randomized or deterministic protocols using an unbounded number of rounds of communication. This is a striking difference from the deterministic model, for which Orlitsky [80] showed that the best one-round protocols can require the transmission of exponentially more bits than the optimal protocols. It is also different from the private coin randomized communication complexity of boolean functions, which exhibits the same phenomenon [112]. Finally, I prove that for several classes of problems, the best deterministic and private coin randomized protocols require asymptotically the same number of communication bits. I conjecture that the deterministic and private coin randomized models are equivalent.

The outline of the chapter is as follows. In Section 8.2, I describe the complexity models and present the existing work. The private coin and public coin randomized models are treated in Sections 8.3 and 8.4, respectively. Randomized amortized interactive communication is studied in Section 8.5, and the distributional model in Section 8.6. Finally, the results for balanced and symmetric problems are presented in Section 8.7, and a summary of the chapter in Section 8.8.

8.2 Complexity Models and Known Results

8.2.1 Preliminaries

The framework for studying interactive communication was introduced by Orlitsky in his seminal paper [80]. Let X and Y be finite sets, and let $S \subseteq X \times Y$, the *support set* of (X, Y), define an interactive communication problem. Two players, $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$, possess respectively inputs $x \in X$ and $y \in Y$ such that $(x, y) \in S$, and they want $P_{\mathcal{Y}}$ to learn x while minimizing the communication between them (it is not necessary for $P_{\mathcal{X}}$ to learn y). It is assumed that the communication between the players is binary.

A *k-round protocol* is a protocol such that for every input, there are at most $k-1$ alternations between the data sent by $P_{\mathcal{X}}$ and the data sent by $P_{\mathcal{Y}}$. Due to the asymmetric nature of the interactive communication model, it is assumed that the last round of communication is always from $P_{\mathcal{X}}$ to $P_{\mathcal{Y}}$. A one-round protocol is also called *one-way*, and a protocol requiring more than one round of communication is called *two-way*.

A hypergraph is an ordered pair $G = (V, E)$, where V is the set of vertices and E the set of hyperedges. Each hyperedge is a subset of V. Two distinct vertices v_1 and v_2 of a hypergraph are adjacent if there is an edge $e \in E$ such that $v_1 \in e$ and $v_2 \in e$. A *proper coloring* of a hypergraph is a partition of V in colors such that no adjacent vertices have the same color. The *chromatic number* χ of a hypergraph is the smallest number of colors for which there exists a proper coloring of G. A convenient way to analyze an interactive communication problem $S \subseteq X \times Y$ is to use its *characteristic hypergraph* G_S. The vertices of G_S are the elements of X, and for every $y \in Y$ there is a hyperedge $E(y) \triangleq \{x \mid (x, y) \in S\}$. The number of different hyperedges of G_S is denoted by σ. It should be noted that all of the asymptotic bounds presented in this chapter only make sense for the support sets for which χ is a function of the size of the inputs.

The *ambiguity set* of an input $x \in X$, defined as $a(x) \triangleq \{y \in Y \mid (x, y) \in S\}$, is the set of all possible inputs for $P_{\mathcal{Y}}$ given that $P_{\mathcal{X}}$'s input is x, and the *ambiguity* of x is $|a(x)|$. The *maximum ambiguity* of $P_{\mathcal{X}}$, $\widehat{a_{\mathcal{X}}} \triangleq \max_{x \in X}\{|a(x)|\}$, is the maximum number of possible elements of Y for any element in X. Note that $a(y)$, $|a(y)|$ and $\widehat{a_{\mathcal{Y}}}$ are defined similarly, with the assumption that $\widehat{a_{\mathcal{Y}}} > 1$, since a support set S with $\widehat{a_{\mathcal{Y}}} = 1$ is trivial and does not require communication.

Example 8.2. As an illustration of the league problem presented in Example 8.1, the support set L is defined as $L = \{(t_1, \{t_1, t_2\}) \mid t_1 \neq t_2\}$, where $t_1, t_2 \in \{0,1\}^n$ are teams. The vertices of G_L are the teams of the league, and its hyperedges are the possible

match-ups for the championship game. It follows that the maximum ambiguity of $P_{\mathcal{X}}$ is the number of possible runner-ups given the champion team ($\widehat{a_{\mathcal{X}}} = 2^n - 1$), and the maximum ambiguity of $P_{\mathcal{Y}}$ is the number of possible champion teams given the two finalists ($\widehat{a_{\mathcal{Y}}} = 2$).

A support set $S \subseteq X \times Y$ is a *Cartesian-product* support set if there exists $X' \subseteq X$ and $Y' \subseteq Y$ such that $S = X' \times Y'$. A support set S is *balanced* if $\widehat{a_{\mathcal{X}}} = \widehat{a_{\mathcal{Y}}}$. A *symmetric* support set is a support such that $(x, y) \in S$ if and only if $(y, x) \in S$ (it is clear that symmetric support sets are also balanced). Symmetric support sets arise naturally in all of the problems for which the parties' inputs are bounded by a certain "distance", including all of the applications mentioned in Section 8.1.

8.2.2 Worst-Case Deterministic Model

The worst-case deterministic model was introduced by Orlitsky [80]. It has the following properties:

1. Both players send information according to a deterministic protocol. Each player sends messages based on its input and the messages previously received.
2. When a player sends a message, its interlocutor knows when it ends, and both players know that the transmission ends when the protocol halts.
3. $P_{\mathcal{Y}}$ has to learn x without error for every pair $(x, y) \in S$.

Let the *codeword* of $(x, y) \in S$ be the concatenation of the messages sent by the players on the input pair (x, y). It can be shown that if the three properties are satisfied, then the set of codewords for all the pairs $(x, y) \in S$ is prefix-free.

The *worst-case deterministic complexity* of a support set S, written $\hat{C}_\infty(S)$, is the minimum number of bits the players have to exchange in order for $P_{\mathcal{Y}}$ to learn x without error for every pair $(x, y) \in S$. We write $\hat{C}_k(S)$ when the number of rounds of communication is bounded by k; obviously, $\hat{C}_k(S)$ decreases with k, and $\hat{C}_\infty(S) = \lim_{k \to \infty} \hat{C}_k(S)$. We write $\hat{C}^*(S)$ for the number of bits that need to be transmitted if $P_{\mathcal{X}}$ knows y in advance.

The following results were shown by Orlitsky [80]. A deterministic protocol requires at least $\lceil \log \widehat{a_{\mathcal{Y}}} \rceil$ bits of communication, otherwise if $|a(y)| = \widehat{a_{\mathcal{Y}}}$, then there are different input pairs (x_1, y) and (x_2, y) for which the communication between the players is the same. Clearly, the bound is tight if $P_{\mathcal{X}}$ knows y in advance, and a single round of communication is sufficient.

Result 8.3.

$$\hat{C}_\infty(S) \geq \lceil \log \widehat{a_Y} \rceil = \hat{C}^*(S).$$

The one-way deterministic complexity is the logarithm of the chromatic number of the underlying hypergraph of the problem, and one-way protocols require at most exponentially more bits than protocols allowing interaction.

Result 8.4.

$$\hat{C}_\infty(S) \geq \lceil \log \hat{C}_1(S) \rceil + 1 = \lceil \log \lceil \log \chi \rceil \rceil + 1.$$

A remarkable result from Orlitsky is that two rounds of communication are almost optimal for every problem, i.e.,

$$\hat{C}_2(S) \leq 4\hat{C}_\infty(S) + 3.$$

This is quite different from the original communication complexity model, where for every $k > 0$, there is a function whose best k-round protocol requires an almost exponentially higher number of bits than its best $(k+1)$-round protocol [31].

Orlitsky [81] showed in a subsequent paper that two rounds of communication are not optimal for worst-case deterministic interactive communication, and Zhang and Xia [114] proved that three rounds are not optimal either. Ahlswede, Cai and Zhang [2] conjectured that four rounds are optimal, but the problem remains open, i.e., whether there is a k such that

$$\hat{C}_k(S) \leq \hat{C}_\infty(S) + o(\hat{C}_\infty(S)).$$

Naor, Orlitsky and Shor [77] established an upper bound on the 4-round deterministic complexity.

Result 8.5.

$$\begin{aligned} \hat{C}_4(S) &\leq \log\log\sigma + \log\widehat{a_Y} + 3\log\log\widehat{a_Y} + 7 \\ &\leq \log\log\chi + 2\log\widehat{a_Y} + 3\log\log\widehat{a_Y} + 7 \\ &\leq 3\hat{C}_\infty(S) + o(\hat{C}_\infty(S)). \end{aligned}$$

Balanced and symmetric support sets were studied by Orlitsky [83], who proved that the best one-way protocols require at most two times the amount of communication required by optimal protocols, i.e,

$$\hat{C}_1(S) \leq 2\hat{C}_\infty(S) + 1.$$

Orlitsky also showed that interaction can reduce the communication required, and that three rounds of communication are optimal.

Result 8.6. *Let S be a balanced support set. Then,*

$$\hat{C}_\infty(S) \leq \hat{C}_3(S) \leq \log \widehat{a_Y} + 3 \log\log \widehat{a_Y} + 11 \leq \hat{C}_\infty(S) + o(\hat{C}_\infty(S)).$$

8.2.3 Worst-Case Private Coin Randomized Model

In the first nonzero-error model, $P_\mathcal{Y}$ is allowed to learn x with probability of error ϵ. The players are also allowed to toss coins; $P_\mathcal{X}$ and $P_\mathcal{Y}$ possess independent finite random strings $r_\mathcal{X}$ and $r_\mathcal{Y}$ of arbitrary length, respectively. The communication bits become random variables: the bits sent by $P_\mathcal{X}$ depend on x, $r_\mathcal{X}$ and bits received from $P_\mathcal{Y}$, and the bits sent by $P_\mathcal{Y}$ depend on y, $r_\mathcal{Y}$ and bits received from $P_\mathcal{X}$. It is therefore possible that for a fixed input pair (x, y), a protocol outputs different results for different values of $r_\mathcal{X}$ and $r_\mathcal{Y}$.

Let S be a support set and let $\mathcal{P}$ be a randomized protocol. $\mathcal{P}$ computes S with error ϵ if, for every pair $(x, y) \in S$, the probability that $P_\mathcal{Y}$ answers x on input (x, y) is at least $1 - \epsilon$. The *worst-case communication* of a protocol $\mathcal{P}$ on input (x, y) is the maximum number of bits communicated for any choice of the random strings $r_\mathcal{X}$ and $r_\mathcal{Y}$. The *worst-case cost* of $\mathcal{P}$ is the maximum, for all the inputs (x, y), of the worst-case communication of $\mathcal{P}$ on (x, y).

The *ϵ-error worst-case randomized complexity* of S, written $\hat{R}^\epsilon_\infty(S)$, is the minimum worst-case cost of a randomized protocol computing S with error ϵ, for $0 < \epsilon < 1$. In other words, $\hat{R}^\epsilon_\infty(S)$ is the number of bits transmitted in the worst case by the best protocol which, for every pair $(x, y) \in S$, allows $P_\mathcal{Y}$ to learn x with a probability of at least $1 - \epsilon$. We write $\hat{R}^\epsilon_k(S)$ when the number of rounds is bounded by k. Also, for the rest of this chapter, ϵ is constant and $c(\epsilon)$ is a function of ϵ.

In his first paper on interactive communication, Orlitsky [80] briefly investigated a weaker randomized model, considering the average communication over the choices of $r_\mathcal{X}$ and $r_\mathcal{Y}$ for the worst input pair. He showed that $\mathfrak{R}^\epsilon_1(S)$, the *average-$\epsilon$-error worst-case randomized complexity* of S, is at most four times the worst-case deterministic complexity, i.e.,

$$\mathfrak{R}^\epsilon_1(S) \leq 4\hat{C}_\infty(S) + 2 \log \frac{1}{\epsilon}.$$

8.2.4 Worst-Case Public Coin Randomized Model

In the randomized model previously defined, each player has its own random generator. $P_{\mathcal{X}}$ cannot see $r_{\mathcal{Y}}$ and vice-versa. In the public coin randomized model, both players can access a common "public" random coin. Formally, both players have a common random string r following a probability distribution Π. The communication bits sent by $P_{\mathcal{X}}$ depend on x, r and bits received from $P_{\mathcal{Y}}$, and those sent by $P_{\mathcal{Y}}$ depend on y, r and bits received from $P_{\mathcal{X}}$. A public coin randomized protocol can also be viewed as a probability distribution over a family of worst-case deterministic protocols.

The *ϵ-error worst-case public coin randomized complexity* of a support set S, written $\hat{R}_{\infty}^{\epsilon,pub}(S)$, is the number of bits transmitted in the worst case by the best public coin protocol, which allows $P_{\mathcal{Y}}$ to learn x with an error probability bounded by ϵ for every pair $(x, y) \in S$ and $0 < \epsilon < 1$. We write $\hat{R}_{k}^{\epsilon,pub}(S)$ when the number of rounds is bounded by k.

8.2.5 Worst-Case Amortized Models

For several models of computation, including interactive communication, simultaneous resolution of several independent instances of a problem can be more efficient than sequential resolution of the instances. This phenomenon is known as the *direct-sum problem*, and was introduced by Karchmer, Raz, and Wigderson [51] for communication complexity of relations as a promising approach to separate the complexity classes $\mathcal{NC}^1$ and $\mathcal{NC}^2$ [84].

Let $S \subseteq X \times Y$ define an interactive communication problem, and let $(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)$ be l independent instances of S. P_X knows $(x_1, x_2, \ldots, x_l)$, P_Y knows $(y_1, y_2, \ldots, y_l)$, and the goal is for $P_{\mathcal{Y}}$ to learn all the x_i from $P_{\mathcal{X}}$ while minimizing the worst-case communication. We write $\hat{C}_{\infty}(S^l)$ for the *simultaneous worst-case deterministic complexity* of l instances of a support set S. The *worst-case deterministic amortized complexity* of S, written $\hat{A}_{\infty}(S)$, is a complexity measure representing the average communication per instance, and is given by the expression

$$\hat{A}_{\infty}(S) \triangleq \lim_{l \to \infty} \frac{1}{l} \hat{C}_{\infty}(S^l).$$

We write $\hat{C}_k(S^l)$ and $\hat{A}_k(S)$ when the number of rounds is bounded by k. Clearly, $\hat{C}_{\infty}(S^l) \leq l \cdot \hat{C}_{\infty}(S)$ and $\hat{A}_{\infty}(S) \leq \hat{C}_{\infty}(S)$. Deterministic amortized complexity for interactive communication was studied by Naor, Orlitsky and Shor [77] and Alon and Orlitsky

[3]. In the former paper, it was proven that the deterministic amortized complexity is equal to the complexity when $P_{\mathcal{X}}$ knows y in advance, and that at most four rounds of communication are required to find an optimal protocol. Ahlswede, Cai and Zhang [2] subsequently reduced the number of rounds to three.

Result 8.7.

$$\hat{A}_3(S) = \hat{A}_4(S) = \cdots = \hat{A}_\infty(S) = \hat{C}^*(S) = \log \widehat{a_y}.$$

Example 8.8. We want to solve the league problem for two seasons, assuming that the results are independent. $P_{\mathcal{Y}}$ wants to learn the identity of the two champion teams from $P_{\mathcal{X}}$, who knows the two pairs of finalists. Obviously, if both seasons are solved independently, $2(\lceil \log n \rceil + 1)$ communication bits are required. However, by treating the two seasons as one larger problem, it was shown [33] that $\hat{C}_2(L^2) \leq \lceil \log n \rceil + 6$. When the number of teams in the league is large, solving one or two instances requires roughly the same number of communication bits. Moreover, Result 8.7 implies that the deterministic amortized complexity of the league problem is 1 bit per instance.

This chapter examines the simultaneous resolution of several instances of interactive communication problems using nonzero-error randomized protocols: $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ are allowed to toss private coins, and $P_{\mathcal{X}}$ must learn $(x_1, x_2, \ldots, x_l)$ correctly with a probability of at least $1 - \epsilon$, for $0 < \epsilon < 1$. We write $\hat{R}^\epsilon_\infty(S^l)$ for the *simultaneous ϵ-error worst-case private coin randomized complexity* of l instances of a support set S. The *ϵ-error worst-case private coin randomized amortized complexity* of S is given by the expression

$$\hat{A}^\epsilon_\infty(S) \triangleq \lim_{l \to \infty} \frac{1}{l} \hat{R}^\epsilon_\infty(S^l). \tag{8.1}$$

Again, we write $\hat{R}^\epsilon_k(S^l)$ and $\hat{A}^\epsilon_k(S)$ when the number of rounds is bounded by k.

8.2.6 Worst-Case Distributional Model

In all of the complexity models defined so far, any interactive communication problem is completely described by its support set S. In effect, since I consider the communication in the worst case for all the possible input pairs (x, y), the probability distribution over the inputs is irrelevant. In the final nonzero-error model, deterministic protocols are allowed to fail with probability 1 for some pairs $(x, y) \in S$ provided they are correct for most of the inputs. Let μ be a probability distribution over S (it is assumed that the pairs $(x, y) \notin S$ are not possible) The *worst-case (μ, ϵ)-distributional complexity* of S, written

$\hat{D}^{\mu,\epsilon}_{\infty}(S)$, is the number of bits transmitted in the worst case by the best deterministic protocol that allows $P_{\mathcal{Y}}$ to learn x for a fraction of at least $1-\epsilon$ of the inputs $(x,y) \in S$, weighted by μ, for $0 < \epsilon < 1$. The players are allowed to use an agreed-upon protocol based on μ.

The only model previously studied in the context of interactive communication and considering a probability distribution μ over the inputs is the zero-error average-case deterministic model. It was studied by Orlitsky [82] and Alon and Orlitsky [4]. $P_{\mathcal{Y}}$ wants to learn x with certainty using a deterministic protocol, with the goal of minimizing the expected number of bits that need to be transmitted, weighted by μ. Orlitsky [82] also studied a slightly different distributional model in which $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ want to convey x and y to a third person P with the assumption that $(x,y) \in S$. Several variants were studied including allowing interaction, randomization, and errors. The model has interesting similarities to the Slepian-Wolf problem [96].

8.3 Private Coin Randomized Interactive Communication: One Round of Communication is Nearly Optimal

In this section, I study worst-case private coin randomized interactive communication. I derive two lower bounds and an upper bound and use them to prove that a single round of communication is close to optimal for this model. As mentioned in the introduction, this is quite different from worst-case deterministic interactive communication, and even from the worst-case randomized communication complexity of boolean functions.

Notice first that a randomized protocol can simulate a deterministic protocol by ignoring the output of the random generators, thus

$$\hat{R}^{\epsilon}_{\infty}(S) \leq \hat{C}_{\infty}(S). \tag{8.2}$$

An initial lower bound, Lemma 8.9, shows that for any support set, the difference between private coin randomized complexity and one-way deterministic complexity is at most exponential.

Lemma 8.9.

$$\hat{R}^{\epsilon}_{\infty}(S) \in \Omega(\log \hat{C}_1(S)).$$

Proof. A randomized protocol $\mathcal{P}$ for a support set S can be represented by a binary tree. During the execution of $\mathcal{P}$ on input $(x, y) \in S$, the players travel down the tree from the root to one of the leaves. At each internal node, one of the players uses its input and random string to decide which sibling should be followed, and conveys the answer to its interlocutor by transmitting it a bit. The output of the protocol is one of the values $x' \in a(y)$[1] of the leaf reached at the end of the execution, and the worst-case cost of $\mathcal{P}$ is the height of the tree. The probability of reaching a leaf L is the product of the probabilities over the choice of the random strings encountered alternatively by $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ along the root-to-leaf path leading to L. This probability can be rewritten as $p_L \triangleq p_{\mathcal{X},L} \cdot p_{\mathcal{Y},L}$, where $p_{\mathcal{X},L}$ and $p_{\mathcal{Y},L}$ collect the probabilities pertaining to $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$, respectively.

To prove the lemma, an optimal private coin randomized protocol is derandomized using a well-known technique (see Lemma 3.8 in [57] for the equivalent result for boolean functions). For each leaf L of the protocol tree, $P_{\mathcal{X}}$ sends $p_{\mathcal{X},L}$ to $P_{\mathcal{Y}}$. Each real number is sent using $p = -\log\left(\frac{1}{2} - \epsilon\right) + \hat{R}^{\epsilon}_{\infty}(S)$ bits, which means that the difference between the exact probability and its rounded value is at most 2^{-p}. $P_{\mathcal{Y}}$ then computes $p_L = p_{\mathcal{X},L} \cdot p_{\mathcal{Y},L}$ for each leaf and derives the probability to output x_i for every $x_i \in a(y)$. Suppose that $0 < \epsilon < \frac{1}{2}$. Since $\mathcal{P}$ computes S incorrectly with a probability of at most ϵ, there is a single $x_i = x$ that is output with a probability of at least $1 - \epsilon$. The total rounding error is smaller than

$$2^{\hat{R}^{\epsilon}_{\infty}(S)} \cdot 2^{-p} = 2^{\hat{R}^{\epsilon}_{\infty}(S)} \cdot 2^{-\log\left(\frac{1}{2}-\epsilon\right)+\hat{R}^{\epsilon}_{\infty}(S)} = \frac{1}{2} - \epsilon,$$

because the tree has at most $2^{\hat{R}^{\epsilon}_{\infty}(S)}$ leaves and only the information sent by $P_{\mathcal{X}}$ has to be rounded. Hence, the only $x_i \in a(y)$ with probability greater than $\frac{1}{2}$ is x. The communication complexity of the derandomized protocol is $\hat{C}_1(S) \leq 2^{\hat{R}^{\epsilon}_{\infty}(S)} \cdot p$. Thus,

$$\hat{R}^{\epsilon}_{\infty}(S) \geq \log \hat{C}_1(S) - \log \hat{R}^{\epsilon}_{\infty}(S) - c(\epsilon). \tag{8.3}$$

The result of this lemma follows.

□

Lemma 8.9 can be used to show that the deterministic and private coin randomized models are equivalent when the difference between the one-way deterministic complexity and the two-way deterministic complexity is exponential.

[1] Recall that $a(y)$ is the ambiguity set of y.

Corollary 8.10. *If* $\hat{C}_\infty(S) \sim {}^2 \log \hat{C}_1(S)$, *then*

$$\hat{C}_\infty(S) \sim \hat{R}^\epsilon_\infty(S).$$

If $\hat{C}_\infty(S) \in \Theta(\log \hat{C}_1(S))$, *then*

$$\hat{C}_\infty(S) \in \Theta(\hat{R}^\epsilon_\infty(S)).$$

Proof. Combining (8.2) and (8.3), we get

$$\log \hat{C}_1(S) - \log \hat{C}_\infty(S) - c(\epsilon) \leq \hat{R}^\epsilon_\infty(S) \leq \hat{C}_\infty(S).$$

If $\hat{C}_\infty(S) \sim \log \hat{C}_1(S)$, then asymptotically,

$$\hat{C}_\infty(S) - \log \hat{C}_\infty(S) - c(\epsilon) \leq \hat{R}^\epsilon_\infty(S) \leq \hat{C}_\infty(S),$$

thus $\hat{C}_\infty(S) \sim \hat{R}^\epsilon_\infty(S)$. The second case is proved in a similar way. □

Example 8.11. Recall that for the league problem presented in Example 8.1, $\hat{C}_\infty(L) = \lceil \log n \rceil + 1$ and $\hat{C}_1(L) = n$. Using Corollary 8.10, it follows that $\hat{R}^\epsilon_\infty(L) \sim \log n$, hence the worst-case deterministic and private coin randomized models are equivalent for this problem.

Lemma 8.9 can also be used to show that the worst-case deterministic and private coin randomized models are equivalent when $P_\mathcal{Y}$'s ambiguity is constant.

Corollary 8.12. *If* $\widehat{a_\mathcal{Y}} \in O(1)$, *then*

$$\hat{C}_\infty(S) \sim \hat{R}^\epsilon_\infty(S).$$

Proof. From (8.2) and (8.3), it follows that

$$\log \hat{C}_1(S) - \log \hat{R}^\epsilon_\infty(S) - c(\epsilon) \leq \hat{R}^\epsilon_\infty(S) \leq \hat{C}_\infty(S).$$

Moreover,

$$\hat{C}_\infty(S) \leq \log \log \chi + O(1) \leq \log \hat{C}_1(S) + O(1)$$

[2] $f(n) \sim g(n)$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

from the assumption and Results 8.4 and 8.5, thus

$$\hat{C}_\infty(S) - \log \hat{R}^\epsilon_\infty(S) - O(1) \le \hat{R}^\epsilon_\infty(S) \le \hat{C}_\infty(S).$$

□

The next lemma tightens the bound from Lemma 8.9 for problems with a small discrepancy between $\hat{C}_1(S)$ and $\hat{C}_\infty(S)$.

Lemma 8.13.

$$\hat{R}^\epsilon_\infty(S) \ge \log \widehat{ay} - \log \frac{1}{1-\epsilon}.$$

Proof. Suppose that $\hat{R}^\epsilon_\infty(S) < \log \widehat{ay} - \log \frac{1}{1-\epsilon}$. By definition, there exists a protocol $\mathcal{P}$ for S requiring fewer than $\log \widehat{ay} - \log \frac{1}{1-\epsilon}$ bits of communication and such that for every pair $(x, y) \in S$, the probability that $P_\mathcal{Y}$ does not learn x correctly is at most ϵ.

Let $y \in Y$ be an input such that $|a(y)| = \widehat{ay}$. For every choice of the random strings $r_\mathcal{X}$ and $r_\mathcal{Y}$, fewer than $(1-\epsilon) \cdot \widehat{ay}$ distinct messages can be transmitted; it follows that the protocol makes an error for more than $\epsilon \cdot \widehat{ay}$ of the $x_i \in a(y)$. By a simple counting argument, there is at least one element $x' \in a(y)$ such that the error probability of $\mathcal{P}$ on the input pair (x', y) is more than ϵ, which is a contradiction. □

The previous bound is very weak for problems for which the difference between the one-way deterministic complexity and two-way deterministic complexity is large. For the league problem, $\widehat{ay} = 2$, and Lemma 8.13 gives $\hat{R}^\epsilon_\infty(L) \ge 1$. On the other hand, the bound is tight for balanced and symmetric pairs, as it will be shown in Section 8.7. The next lemma gives a strong upper bound on the private coin randomized complexity.

Lemma 8.14.

$$\hat{R}^\epsilon_1(S) \le 2 \left\lceil \log\log\chi + \log \widehat{ay} + \log\left(\frac{1}{\epsilon} - 1\right) + 1 \right\rceil.$$

Proof. Let G_S be the characteristic hypergraph of S, and let x be $P_\mathcal{X}$'s input. Recall that $P_\mathcal{Y}$ has an input y defining the ambiguity set (hyperedge) $a(y) = \{x_1, \ldots, x_l\}$ and wants to learn which of the x_i is x. Let $k \triangleq \lceil \log\chi \rceil - 1$. Players agree on a proper coloring Ψ of G_S with χ colors. We write the color of the vertices s of G_S in binary form: $\Psi_s \triangleq s_0 s_1 \ldots s_k$, where $s_i \in \{0, 1\}$. We consider these strings as polynomials in $\mathbb{Z}_p$, where

p is a prime number such that

$$\left\lfloor \frac{1-\epsilon}{\epsilon} \cdot k \cdot (\widehat{a_{\mathcal{Y}}} - 1) \right\rfloor < p \leq 2 \left\lfloor \frac{1-\epsilon}{\epsilon} \cdot k \cdot (\widehat{a_{\mathcal{Y}}} - 1) \right\rfloor.$$

Bertrand's postulate [43] guarantees the existence of such a p. In other words, for a vertex s of G_S whose color is Ψ_s, $\Psi_s(t) \equiv (s_0 + s_1 t + \cdots + s_k t^k) (\mathrm{mod}\ p)$.

$P_{\mathcal{X}}$ randomly chooses $m \in \mathbb{Z}_p$ and sends m and $\Psi_x(m)$ to $P_{\mathcal{Y}}$. $P_{\mathcal{Y}}$ then constructs the set

$$E_m \triangleq \{x_i \mid x_i \in a(y) \wedge \Psi_{x_i}(m) = \Psi_x(m)\},$$

randomly chooses an element of E_m and concludes that it is $P_{\mathcal{X}}$'s input.

In order to show that the protocol works, one has to prove that the probability that $P_{\mathcal{Y}}$ answers x is at least $1 - \epsilon$. First, observe that $x \in E_m$, therefore $|E_m| \geq 1$ and $P_{\mathcal{Y}}$ can only answer incorrectly when $|E_m| \geq 2$. Thus, we get

$$\begin{aligned}
\Pr[P_{\mathcal{Y}} \text{ answers } x] &= \sum_{j \in \mathbb{Z}_p} \Pr[P_{\mathcal{X}} \text{ randomly chooses } j \in \mathbb{Z}_p \wedge P_{\mathcal{Y}} \text{ guesses correctly}] \\
&= \frac{1}{p} \sum_{j=0}^{p-1} \Pr[P_{\mathcal{Y}} \text{ answers } x \mid j \text{ is chosen}] \\
&= \frac{1}{p} \sum_{j=0}^{p-1} \frac{1}{|E_j|}.
\end{aligned}$$

Furthermore, since Ψ is a proper coloring of G_S, all of the vertices of the hyperedge $a(y)$ have a different color, thus the corresponding polynomials $\Psi_s(t)$ are different. This difference implies that two such polynomials agree on at most k field elements because their difference, a nonzero polynomial of degree at most k, has at most k roots. Hence, we can deduce that

$$\sum_{j=0}^{p-1} |E_j| \leq k \cdot (\widehat{a_{\mathcal{Y}}} - 1) + p,$$

which implies that

$$\sum_{j=0}^{p-1} \frac{1}{|E_j|} \geq \frac{p}{\left(\frac{k \cdot (\widehat{a_{\mathcal{Y}}} - 1) + p}{p}\right)},$$

and thus

$$\begin{aligned}
\Pr[P_{\mathcal{Y}} \text{ answers } x] &\geq \frac{1}{\frac{k\cdot(\widehat{a_{\mathcal{Y}}}-1)}{p}+1} \\
&\geq \frac{1}{\frac{k\cdot(\widehat{a_{\mathcal{Y}}}-1)}{k\cdot(\widehat{a_{\mathcal{Y}}}-1)\cdot\frac{1-\epsilon}{\epsilon}}+1} \\
&= \frac{1}{\frac{\epsilon}{1-\epsilon}+1} \\
&= \frac{1}{\left(\frac{1}{1-\epsilon}\right)} \\
&= 1-\epsilon.
\end{aligned}$$

Finally, the complexity of the protocol is

$$\begin{aligned}
\hat{R}_1^{\epsilon}(S) &\leq 2\lceil \log p \rceil \\
&\leq 2\left\lceil 1+\log\left(\frac{1}{\epsilon}-1\right)+\log\log\chi+\log\widehat{a_{\mathcal{Y}}}\right\rceil.
\end{aligned}$$

□

Using Lemma 8.14 with the league problem, we get $\hat{R}_1^{\epsilon}(L) \leq 2\lceil \log n \rceil + c(\epsilon)$. More generally, the lemma can be used to prove that the one-way private coin randomized complexity is at most four times that of the worst-case deterministic complexity.

Corollary 8.15.

$$\hat{R}_1^{\epsilon}(S) \leq 4\hat{C}_{\infty}(S) + \left\lceil 2\log\frac{1}{\epsilon}\right\rceil.$$

Proof. Using Lemma 8.14 and Results 8.3 and 8.4, we get

$$\begin{aligned}
\hat{R}_1^{\epsilon}(S) &\leq 2\left\lceil 1+\log\left(\frac{1}{\epsilon}-1\right)+\log\hat{C}_1(S)+\hat{C}_{\infty}(S)\right\rceil \\
&\leq 2\left\lceil 1+\log\left(\frac{1}{\epsilon}-1\right)+\hat{C}_{\infty}(S)-1+\hat{C}_{\infty}(S)\right\rceil \\
&\leq 4\hat{C}_{\infty}(S)+2\left\lceil\log\frac{1}{\epsilon}\right\rceil.
\end{aligned}$$

□

Combining Lemma 8.14 with the two lower bounds proved in this section, we can conclude that one round of communication is nearly optimal for the worst-case private

coin randomized model.

Corollary 8.16.

$$\hat{R}_1^\epsilon(S) \leq 4\hat{R}_\infty^\epsilon(S) + o(\hat{R}_\infty^\epsilon(S)).$$

Proof. From Result 8.4 and Lemmas 8.9, 8.13, and 8.14, it follows that

$$\begin{aligned} \hat{R}_1^\epsilon(S) &\leq 2\log \hat{C}_1(S) + 2\log \widehat{a_Y} + c_1(\epsilon) \\ &\leq 4\hat{R}_\infty^\epsilon(S) + 2\log \hat{R}_\infty^\epsilon(S) + c_2(\epsilon) \\ &\leq 4\hat{R}_\infty^\epsilon(S) + o(\hat{R}_\infty^\epsilon(S)). \end{aligned}$$

□

8.4 Public Coin Randomized Interactive Communication

In this section, I characterize worst-case public coin randomized interactive communication. First, I prove that one round of communication is optimal for this model; the complexity corresponds to the amortized complexity and to the communication required when P_Y knows y in advance. Second, I derive an upper bound on the difference between the private coin and public coin randomized models, and use it to improve upon the upper bound for the one-way private coin complexity derived in the previous section. One should notice that the players can use the concatenation of the private strings r_X and r_Y as a public random string, thus the public coin randomized model is at least as powerful as the private coin randomized model, i.e.,

$$\hat{R}_k^{\epsilon,pub}(S) \leq \hat{R}_k^\epsilon(S).$$

8.4.1 One Round of Communication is Optimal

Lemma 8.17.

$$\log \widehat{a_Y} - \log \frac{1}{1-\epsilon} \leq \hat{R}_\infty^{\epsilon,pub}(S) \leq \hat{R}_1^{\epsilon,pub}(S) \leq \lceil \log(\widehat{a_Y} - 1) \rceil + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil.$$

Proof. Lemma 8.13 can be applied with a public random generator for the lower bound. For the upper bound, recall that P_X has an input x and P_Y has an input y defining the

ambiguity set $a(y) = \{x_1, \ldots, x_l\} = \{x \mid (x, y) \in S\}$. It is assumed that the elements of X are coded using $\lceil \log |X| \rceil$ bits. $P_{\mathcal{X}}$ chooses k random subsets of the bits of his input x and, for each of these subsets, sends the parity of the bits to $P_{\mathcal{Y}}$. $P_{\mathcal{Y}}$ then computes the k corresponding parities for each of the inputs $x_i \in a(y)$. Each time a parity differs from the corresponding result sent by $P_{\mathcal{X}}$, $P_{\mathcal{Y}}$ deduces that $x_i \neq x$ and discards it. When $P_{\mathcal{Y}}$ has performed all of the comparisons, it randomly chooses an input among those not discarded and concludes that it is $P_{\mathcal{X}}$'s input.

The probability of not discarding a vertex x_i is 1 if $x_i = x$ and $\frac{1}{2^k}$ otherwise. Let Z be a random variable representing the number of inputs not discarded after k iterations. Since there are $a(y) - 1$ inputs to discard, it follows that

$$E(Z) = 1 + \frac{1}{2^k}(a(y) - 1) \leq 1 + \frac{1}{2^k}(\widehat{a_{\mathcal{Y}}} - 1).$$

The probability that $P_{\mathcal{Y}}$ learns $P_{\mathcal{X}}$'s input correctly is $E\left[\frac{1}{Z}\right]$, and since $\frac{1}{Z}$ is convex for $Z > 0$, we use Jensen's inequality to get

$$\Pr[P_{\mathcal{Y}} \text{ answers } x] \geq \frac{1}{E[Z]} = \frac{1}{1 + \frac{1}{2^k}(\widehat{a_{\mathcal{Y}}} - 1)}.$$

By letting $k = \lceil \log(\widehat{a_{\mathcal{Y}}} - 1) \rceil + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil$, it follows that

$$\begin{aligned} \Pr[\text{success}] &\geq \frac{1}{1 + \frac{1}{2^{\lceil \log(\widehat{a_{\mathcal{Y}}} - 1) \rceil + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil}} \cdot (\widehat{a_{\mathcal{Y}}} - 1)} \\ &\geq \frac{1}{1 + \frac{1}{\frac{1-\epsilon}{\epsilon}}} \\ &= 1 - \epsilon, \end{aligned}$$

from which we conclude that

$$\hat{R}_1^{\epsilon, pub}(L) \leq k = \lceil \log(\widehat{a_{\mathcal{Y}}} - 1) \rceil + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil.$$

□

Example 8.18. If we apply Lemma 8.17 to the league problem, we get $\hat{R}_1^{\epsilon, pub}(L) \leq \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil \in \Theta(1)$, which shows that one-way public coin randomized complexity can be arbitrarily better than one-way deterministic complexity.

8.4.2 Difference Between the Private Coin and Public Coin Models

The private coin randomized complexity cannot be much worse than the public coin randomized complexity: every public randomized protocol can be transformed into a private randomized protocol whose error probability is slightly larger, and which uses a few more communication bits.

Theorem 8.19. *For all $\delta > 0$ and for all $\epsilon > 0$ such that $\epsilon + \delta < 1$,*

$$\hat{R}_1^{\epsilon+\delta}(S) \leq \hat{R}_\infty^{\epsilon,pub}(S) + \log\log|S| + \log\frac{1}{\delta^2} + 1.$$

Proof. The proof is inspired by a similar result for boolean functions shown by Newman [78]. Let $\mathcal{P}$ be a public coin randomized protocol for S whose error is bounded by ϵ and requiring $\hat{R}_\infty^{\epsilon,pub}(S)$ communication bits. We suppose that the random generator r follows a probability distribution μ. Let $Z(x,y,r)$ be a random variable equal to 1 if the answer given by $P_\mathcal{Y}$ following the execution of $\mathcal{P}$ on input (x,y) is incorrect (different from x), and equal to 0 if correct. Since $\mathcal{P}$ solves S with error at most ϵ, it follows that $\underset{r\in\mu}{E}[Z(x,y,r)] \leq \epsilon$ for every pair $(x,y) \in S$.

A new public protocol for S using fewer random bits is designed. Let t be a parameter to be fixed later, and let $r_1, r_2, \ldots, r_t$ be binary strings. The protocol $\mathcal{P}_{r_1,r_2,\ldots,r_t}$ is defined as follows: $P_\mathcal{X}$ and $P_\mathcal{Y}$ randomly choose i between 1 and t using their public coin and run protocol $\mathcal{P}$ with the common random string r_i.

We now show that strings $r_1, r_2, \ldots, r_t$ exist such that $\underset{i}{E}[Z(x,y,r_i)] \leq \epsilon + \delta$ for every pair $(x,y) \in S$. We choose the t strings $r_1, r_2, \ldots, r_t$ randomly following the probability distribution μ, consider an arbitrary pair $(x,y) \in S$ and compute the probability that $\underset{i}{E}[Z(x,y,r_i)] > \epsilon + \delta$ (where i is uniformly distributed). The latter is equivalent to the probability that $\frac{1}{t}\sum_{i=1}^{t} Z(x,y,r_i) > \epsilon + \delta$. Since $\underset{r\in\mu}{E}[Z(x,y,r)] \leq \epsilon$, Chernoff bound yields

$$\Pr_{r_1,\ldots,r_t}\left[\frac{1}{t}\sum_{i=1}^{t} Z(x,y,r_i) - \epsilon > \delta\right] \leq 2e^{-2\delta^2 t}.$$

By choosing $t = \left\lceil \frac{\log |S|}{\delta^2} \right\rceil$, it follows that

$$\begin{aligned} 2e^{-2\delta^2 t} &= 2e^{-2\delta^2 \left\lceil \frac{\log |S|}{\delta^2} \right\rceil} \\ &\leq 2e^{-2\log |S|} \\ &= 2 \cdot 2^{-2 \log e \log |S|} \\ &= 2|S|^{-2\log e} \\ &< \frac{1}{|S|} \text{ when } |S| > 1. \end{aligned}$$

Thus, for a random choice of $r_1, \ldots, r_t$, the probability that there exists at least one pair $(x, y) \in S$ (there are $|S|$ such pairs) such that $\underset{i}{E}[Z(x, y, r_i)] > \epsilon + \delta$ is smaller than $|S| \cdot \frac{1}{|S|} = 1$. Consequently, there exists a choice of $r_1, \ldots, r_t$ such that for every pair $(x, y) \in S$, the error of protocol $\mathcal{P}_{r_1, r_2, \ldots, r_t}$ is at most $\epsilon + \delta$. The number of random bits used by $\mathcal{P}_{r_1, r_2, \ldots, r_t}$ is $\lceil \log t \rceil$, and in order to transform the public protocol into a private protocol, $P_{\mathcal{X}}$ has to randomly choose i between 1 and t and to send it to $P_{\mathcal{Y}}$. Moreover, from Lemma 8.17, an optimal one-round public coin randomized protocol exists for S ensuring that $\mathcal{P}_{r_1, r_2, \ldots, r_t}$ is also a one-way protocol. Hence,

$$\begin{aligned} \hat{R}_1^{\epsilon+\delta}(S) &\leq \hat{R}_1^{\epsilon, pub}(S) + \lceil \log t \rceil \\ &\leq \hat{R}_1^{\epsilon, pub}(S) + \left\lceil \log \left\lceil \frac{\log |S|}{\delta^2} \right\rceil \right\rceil \\ &\leq \hat{R}_1^{\epsilon, pub}(S) + \log \log |S| + \log \frac{1}{\delta^2} + 1. \end{aligned}$$

□

Example 8.20. The previous theorem is applied to the league problem, for which $|S| = (2^n) \cdot (2^n - 1)$, giving

$$\begin{aligned} \hat{R}_1^{\epsilon'+\delta}(L) &\leq \hat{R}_\infty^{\epsilon, pub}(S) + \log \log |S| + \log \frac{1}{\delta^2} + 1 \\ &\leq \log \log |2^n (2^n - 1)| + c(\epsilon', \delta), \end{aligned}$$

thus $\hat{R}_1^{\epsilon}(L) \leq \log n + c(\epsilon)$. An optimal one-way private coin randomized protocol is thereby yielded for this problem; it improves on the bound given by Lemma 8.14, which is not optimal, and the result from Example 8.11, which does not limit interaction between the players. This example also proves that the bound given by Theorem 8.19 can be reached.

Theorem 8.19 can also be used to tighten the upper bounds on the one-way private coin randomized complexity presented in Section 8.3.

Corollary 8.21. *For all $\delta > 0$ and for all $\epsilon > 0$ such that $\epsilon + \delta < 1$,*

$$\hat{R}_1^{\epsilon+\delta}(S) \leq \lceil \log(\widehat{a_Y} - 1) \rceil + \log\log|S| + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil + \log \frac{1}{\delta^2} + 1.$$

Corollary 8.22.

$$\hat{R}_1^{\epsilon}(S) \leq \log\log\sigma + \log \widehat{a_Y} + \log\log \widehat{a_Y} + c(\epsilon).$$

Proof. Recall that σ is the number of hyperedges of the characteristic hypergraph G_S. Suppose that the hyperedges for all the $y \in Y$ are different; if this is not the case, the players agree on an equivalent support set $S' \subseteq X' \times Y'$, $S' \subset S$, such that there is a distinct hyperedge for each $y \in Y'$. It follows that $|S'| \leq |\mathcal{Y}'| \cdot \widehat{a_Y}' \leq \sigma \cdot \widehat{a_Y}$, and the result follows from Theorem 8.19 and Lemma 8.17. □

If $X = Y = \{0,1\}^n$, the difference between the private and public coin randomized models is at most an additive term of $\log n + O(1)$ bits. Using Result 8.5, the same thing can be said about the difference between the deterministic and public coin randomized models. In fact, the similarity between Result 8.5 and Corollary 8.22 is striking, considering how different the models and the proofs are.

Corollary 8.23.

$$\hat{R}_1^{\epsilon}(S) \leq 3\hat{R}_\infty^{\epsilon}(S) + o(\hat{R}_\infty^{\epsilon}(S)).$$

Proof. It is well known that a χ-colorable u-uniform hypergraph has at most $\binom{\chi}{u}$ hyperedges, thus S can be represented by a (not necessarily uniform) characteristic hypergraph having at most $\sum_{i=1}^{\widehat{a_Y}} \binom{\chi}{i} \leq \chi^{\widehat{a_Y}}$ hyperedges. The result follows from Corollary 8.22 and Lemmas 8.9 and 8.13. □

Corollary 8.24.

$$\hat{R}_1^{\epsilon}(S) \leq 3\hat{C}_\infty(S) + o(\hat{C}_\infty(S)).$$

8.5 Private Coin Randomized Amortized Interactive Communication: One Round of Communication is Optimal

In this section, I show that one round of communication is optimal for the private coin randomized amortized model; the complexity is equal to the number of bits that need to be transmitted when $P_{\mathcal{X}}$ knows y in advance. This is an improvement over the deterministic amortized model, for which interaction is required in order to minimize the communication.

Lemma 8.25.

$$\hat{A}_1^\epsilon(S) = \hat{A}_2^\epsilon(S) = \cdots = \hat{A}_\infty^\epsilon(S) = \log \widehat{a_{\mathcal{Y}}}.$$

Proof. As mentioned in Section 8.2, l independent instances of a support set $S \subseteq X \times Y$ can be treated as a larger support set S^l. Let G_S^l be the characteristic hypergraph of S^l. It is not hard to show that the vertices of G_S^l are the elements of X^l, and that for each l-tuple $(e_1, e_2, \ldots, e_l)$ of hyperedges of G_S, $e_1 \times e_2 \times \cdots \times e_l$ is a hyperedge of G_S^l. Clearly, the maximum ambiguity of $P_{\mathcal{Y}}$ for the support set S^l is $\widehat{a_{\mathcal{Y}}}^l$, and the number of different hyperedges of G_S^l is σ^l. It follows from Lemma 8.13 that

$$\hat{R}_\infty^\epsilon(S^l) \geq l \cdot \log \widehat{a_{\mathcal{Y}}} - \log \frac{1}{1-\epsilon},$$

and Corollary 8.22 gives

$$\hat{R}_1^\epsilon(S^l) \leq \log\log\sigma + l \cdot \log \widehat{a_{\mathcal{Y}}} + \log\log \widehat{a_{\mathcal{Y}}} + 2\log l + c(\epsilon).$$

The result follows from the definitions of $\hat{A}_1^\epsilon(S)$ and $\hat{A}_\infty^\epsilon(S)$ given by (8.1).

□

Lemma 8.25 shows that when several instances are solved simultaneously instead of sequentially, there is no advantage to using a public coin over private coins since no interaction is required and the amount of communication is the same. Also, comparing Lemma 8.25 and Corollary 8.21 when $|S| \in O(2^{cn})$ (which includes the support sets $S \subseteq \{0,1\}^n \times \{0,1\}^n$), the difference between the private coin randomized complexity and the private coin randomized amortized complexity is at most an additive term of $\log n + O(1)$ bits. This bound is tight for the league problem. The same discrepancy between the

deterministic complexity and the deterministic amortized complexity is implicit in the work of Naor, Orlitsky and Shor [77], using Results 8.5 and 8.7.

8.6 Distributional Interactive Communication: One Round of Communication is Optimal

In this section, I study the worst-case distributional model, starting with yet another example using the league problem.

Example 8.26. Suppose that the support set L follows a uniform distribution, and without loss of generality, that n, the length of the team names, is an even integer. $P_{\mathcal{X}}$ sends two bits to P_Y: the parity of the first $\frac{n}{2}$ bits of the champion team's name, followed by the parity of the last $\frac{n}{2}$ bits. $P_{\mathcal{Y}}$ computes the equivalent parities for the two teams playing for the championship and compares the results with the bits received from P_X. The protocol will err for the pairs of finalist teams with identical parity bits, and there are $|L| \cdot \left(\frac{1}{4} - \frac{1}{2^n}\right)$ such pairs. Hence, $\hat{D}_1^{uniform,\frac{1}{4}}(L) \leq 2$.

It should be noted that for every interactive communication problem and for all $\epsilon > 0$, a probability distribution over the inputs exists for which the distributional complexity is 0, for example, if there is an input pair occurring with probability at least $1 - \epsilon$. Consequently, it is interesting to consider probability distributions maximizing the number of bits that need to be transmitted; even in this case, communication can be more efficient than when no error is allowed. This section established that the distributional complexity with the worst possible probability distribution over the inputs is equal to the public coin randomized complexity, which, again, is equivalent to the complexity when $P_{\mathcal{X}}$ knows y in advance. This is hardly a surprise, since the equivalence between the public coin randomized and distributional models was first established by Yao [110] for computational complexity using von-Neumann's Minimax Theorem of game theory [109]. The equality is also verified for communication complexity of boolean functions (see, for example, [57]), and the proof can be applied without modification to partial domains $S \subseteq X \times Y$. A simpler proof tailored for interactive communication problems in presented; it improves the original proof in two ways. First, a family of asymptotically "worst" probability distributions over the inputs is described. The distributions can be used with any interactive communication problem and probability of error. Second, an optimal protocol requiring a single round of communication is constructed.

Lemma 8.27.

$$\log \widehat{a_Y} - \log \frac{1}{1-2\epsilon} \leq \max_{\mu} \hat{D}_{\infty}^{\mu,\epsilon}(S) \leq \max_{\mu} \hat{D}_{1}^{\mu,\epsilon}(S) \leq \lceil \log(\widehat{a_Y} - 1) \rceil + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil .$$

Proof. To prove the lower bound, the following probability distribution μ' over S is used: there is a $y' \in Y$ with $|a(y')| = \widehat{a_Y}$ such that $\Pr[(x,y) = (x_i, y')] = \frac{1}{2\widehat{a_Y}}$ for every $x_i \in a(y)$. The probability of all the other input pairs $(x,y) \in S$ can be any nonzero value. Suppose that $\hat{D}_{\infty}^{\mu',\epsilon}(S) < \log \widehat{a_Y} - \log \frac{1}{1-2\epsilon}$, and let $\mathcal{P}$ be an optimal protocol for S. For any pair $(x,y) \in S$, less than $\widehat{a_Y} \cdot (1-2\epsilon)$ distinct messages can be exchanged between $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$. It follows that when $P_{\mathcal{Y}}$ has input y', the protocol fails on more than $2\epsilon \cdot \widehat{a_Y}$ input pairs (x_i, y'), each pair occurring with probability $\frac{1}{2\widehat{a_Y}}$. Hence, $\mathcal{P}$ fails with probability greater than ϵ over S, which is a contradiction, and

$$\log \widehat{a_Y} - \log \frac{1}{1-2\epsilon} \leq \hat{D}_{\infty}^{\mu',\epsilon}(S) \leq \max_{\mu} \hat{D}_{\infty}^{\mu,\epsilon}(S).$$

To prove the upper bound, the public coin protocol presented in Lemma 8.17 is used. Recall that by receiving the parity of $k = \lceil \log(\widehat{a_Y} - 1) \rceil + \lceil \log \frac{1-\epsilon}{\epsilon} \rceil$ random subsets of the bits of x from $P_{\mathcal{X}}$, $P_{\mathcal{Y}}$ can learn x with probability at least $1-\epsilon$ for every valid input pair. Since the protocol works for any probability distribution over the inputs, it succeeds with probability at least $1-\epsilon$ when the probability distribution is taken over the inputs and the choice of the subsets. By a simple counting argument, it follows that k subsets of the bits of x exist for which the protocol succeeds for at least a fraction $1-\epsilon$ of the inputs weighted by their probability distribution. Consequently, to derandomize the protocol presented in Lemma 8.17 for a fixed μ and a fixed ϵ, $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ agree on k such subsets and can execute the rest of the protocol as before: $P_{\mathcal{X}}$ sends the parity of the k subsets of x to $P_{\mathcal{Y}}$, who compares the received bits with the k corresponding parities for each of the inputs $x_i \in a(y)$. If more than one of the x_i is not discarded, $P_{\mathcal{Y}}$ chooses the one whose input probability given y is the highest. The complexity of the protocol is

$$\max_{\mu} \hat{D}_{1}^{\mu,\epsilon}(S) \leq \hat{R}_{1}^{\epsilon,pub}(S) \leq k \leq \lceil \log(\widehat{a_Y} - 1) \rceil + \left\lceil \log \frac{1-\epsilon}{\epsilon} \right\rceil .$$

□

The family of "worst" probability distributions presented in the previous proof is not unique. For example, when the ambiguity of every input $y \in Y$ is maximal, i.e., $|a(y)| = \widehat{a_Y}$, it is not hard to show that the uniform distribution also maximizes the

required number of communication bits. The league problem is an example of such a support set: P_Y always picks the winner among two teams.

8.7 Equivalence of All the Models for Balanced and Symmetric Pairs

All of the worst-case complexity models introduced in this chapter require asymptotically at least $\log \widehat{a_Y}$ bits of communication, and with the exception of the deterministic and private coin randomized models, asymptotically $\log \widehat{a_Y}$ bits suffice. For some support sets like the league problem, the difference between $\log \widehat{a_Y}$ and $\hat{C}_\infty(S)$ is large, but for other problems, the amortized or nonzero-error models cannot be used to reduce the number of communication bits. Consider the following modification of the league problem: either P_Y knows the two teams playing for the league championship, or it doesn't know anything. It is obvious that in the worst case, whether it is the worst input or the worst probability distribution over the inputs, P_X has to asymptotically send all of the bits of the champion team's name to P_Y and it can do so in a single round of communication. For the problems mentioned in the next lemma, all of the worst-case models presented in this chapter are asymptotically equivalent and inefficient: randomization, nonzero-error protocols, solving several instances simultaneously, knowing y in advance and even interaction cannot significantly reduce the communication between P_X and P_Y.

Lemma 8.28. *If there is a $y \in Y$ with $|a(y)| = \widehat{a_Y} = |X|$, or if S is a Cartesian-product support set, then*

$$\hat{C}_1(S) = \lceil \log \widehat{a_Y} \rceil .$$

Proof. Result 8.3 gives a lower bound of $\lceil \log \widehat{a_Y} \rceil$. If $\widehat{a_Y} = |X|$, then P_X has to describe x completely; it can do so using $\lceil \log |X| \rceil = \lceil \log \widehat{a_Y} \rceil$ bits. If S is a Cartesian-product support set, then $X' \subseteq X$ and $Y' \subseteq Y$ exist such that $S = X' \times Y'$. Again, $\widehat{a_Y} = |X'|$ and P_X can describe x with $\lceil \log |X'| \rceil = \lceil \log \widehat{a_Y} \rceil$ bits of communication. □

It is also possible to prove that all of the worst-case complexity models are equivalent for balanced and symmetric pairs. Recall that a support set S is balanced when $\widehat{a_X} = \widehat{a_Y}$ and symmetric when $(x, y) \in S$ if and only if $(y, x) \in S$.

Theorem 8.29. *Let S be a balanced support set. The following models are all equivalent: (i) deterministic, (ii) amortized deterministic, (iii) deterministic when P_X knows P_Y's*

input, (iv) private coin randomized, (v) public coin randomized, (vi) distributional, (vii) private coin randomized amortized. More precisely,

$$\hat{C}^*(S) \leq \quad \hat{C}_3(S) \quad \leq \hat{C}^*(S) + o(\hat{C}^*(S)); \tag{8.4}$$

$$\hat{C}^*(S) - 1 \leq \quad \hat{A}_3(S) \quad \leq \hat{C}^*(S); \tag{8.5}$$

$$\hat{C}^*(S) - O(1) \leq \quad \hat{R}_3^{\epsilon}(S) \quad \leq \hat{C}^*(S) + o(\hat{C}^*(S)); \tag{8.6}$$

$$\hat{C}^*(S) - O(1) \leq \quad \hat{R}_1^{\epsilon,pub}(S) \quad \leq \hat{C}^*(S) + O(1); \tag{8.7}$$

$$\hat{C}^*(S) - O(1) \leq \quad \hat{D}_1^{\mu,\epsilon}(S) \quad \leq \hat{C}^*(S) + O(1); \tag{8.8}$$

$$\hat{C}^*(S) - 1 \leq \quad \hat{A}_1^{\epsilon}(S) \quad \leq \hat{C}^*(S). \tag{8.9}$$

Proof. Inequality (8.4) can be deduced from Results 8.3 and 8.6; Inequality (8.5) from Results 8.3 and 8.7; Inequality (8.6) from Results 8.3 and 8.6, Inequality (8.2) and Lemma 8.13; Inequality (8.7) from Result 8.3 and Lemma 8.17; Inequality (8.8) from Result 8.3 and Lemma 8.27; Inequality (8.9) from Result 8.3 and Lemma 8.25. □

Unfortunately, Theorem 8.29 means that nonzero-error algorithms cannot significantly reduce the communication for all of the practical applications mentioned in the introduction. This is a somewhat "negative" result, but only because the best deterministic protocols are already very efficient. However, nonzero-error models, even the private coin randomized model, allow efficient one-way protocols. For most practical applications of symmetric pairs, the ambiguity of the players increases at least polynomially with the size of the inputs; in this case, a private coin randomized protocol exists that uses a single round of communication and whose communication complexity is nearly optimal. If the ambiguity of the players increases superpolynomially with the size of the inputs, then the best one-way private coin randomized protocol is optimal.

Lemma 8.30. *Let S be a support set with $X = \{0,1\}^n$, and let $k \geq 1$. If $\widehat{a_Y} \in \Theta(n^k)$, then*

$$\hat{R}_1^{\epsilon}(S) \leq \left(1 + \frac{1}{k}\right) \hat{R}_{\infty}^{\epsilon}(S) + o(\hat{R}_{\infty}^{\epsilon}(S)).$$

If $k \in \omega(n^k)$ for all $k \geq 1$, then

$$\hat{R}_1^{\epsilon}(S) \leq \hat{R}_{\infty}^{\epsilon}(S) + o(\hat{R}_{\infty}^{\epsilon}(S)).$$

Proof. We prove the case $\widehat{a_{\mathcal{Y}}} \in \Theta(n^k)$. Corollary 8.21 and the assumption give

$$\begin{aligned}
\hat{R}_1^{\epsilon}(S) &\leq \log \widehat{a_{\mathcal{Y}}} + \log\log|S| + c(\epsilon) \\
&\leq \log \widehat{a_{\mathcal{Y}}} + \log\log(|X| \cdot \widehat{a_{\mathcal{Y}}}) + O(1) \\
&\leq \log \widehat{a_{\mathcal{Y}}} + \log\log(2^n \cdot \widehat{a_{\mathcal{Y}}}) + O(1) \\
&\leq \log \widehat{a_{\mathcal{Y}}} + \frac{1}{k}\log n^k + \log\log \widehat{a_{\mathcal{Y}}} + O(1) \\
&\leq \left(1 + \frac{1}{k}\right) \cdot \log \widehat{a_{\mathcal{Y}}} + \log\log \widehat{a_{\mathcal{Y}}} + O(1),
\end{aligned}$$

and the result follows from Lemma 8.13. The case $\widehat{a_{\mathcal{Y}}} \in \omega(n^k)$ is proved similarly. □

Example 8.31. A large database D stored on a PC has to be synchronized with an updated version D' of the database stored on a PDA. The problem can be viewed as a *set reconciliation* problem [74], and for this example it is assumed that D and D' are sets of integers in $[1, 10^6]$, that D' has to be conveyed to the PC, and that D and D' differ in at most 1000 entries, i.e., $|D \setminus D'| + |D' \setminus D| \leq 1000$. Translated in the interactive communication framework, D and D' can be expressed as binary strings x and y such that $x_i = 1$ ($y_i = 1$) if and only if $i \in D$ ($i \in D'$). The correlation between D and D' gives an upper bound on the Hamming distance between x and y, thus the support set is symmetric and $\widehat{a_{\mathcal{X}}} = \widehat{a_{\mathcal{Y}}} = \sum_{i=0}^{1000} \binom{1000000}{i}$.

From Result 8.6, we know that a 3-round deterministic algorithm requiring 11452 bits of communication exists. Nonzero-error algorithms cannot do much better: from Lemma 8.13, even with a public random string, at least 11401 bits need to be exchanged in order for $P_{\mathcal{Y}}$ to learn x with error at most ϵ, for $0 < \epsilon < \frac{1}{2}$. From Lemma 8.17, we know that there is a one-way public coin randomized protocol transmitting 11452 bits with a 2^{-50} probability of error, and from Corollary 8.21 that there is a one-way private coin randomized protocol transmitting 11576 bits with a 2^{-50} probability of error .

This chapter assumes that $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ have unbounded memory and computing power. For most of the practical problems modeled by symmetric support sets, extensive research was conducted to design algorithms with reasonable trade-offs between the number of rounds, communication complexity, computational complexity, and space complexity. Of course, the requirements depend on the application and the size of the maximum ambiguity of the players.

8.8 Summary

In this chapter, I have studied worst-case nonzero-error interactive communication. I have shown that if the players are allowed to use public coins, to answer correctly on a fraction of the inputs or to solve a large number of instances simultaneously, then interaction is not necessary and in some cases the communication required can be significantly reduced. I have also proved that one round of communication is almost optimal if the players are allowed to use private coins, and also that all the models are equivalent for the practical data reconciliation problems that can be modeled by symmetric support sets.

Chapter 9

Conclusions, Open Problems, and Future Work

In this thesis, various problems related to communication over channels with symbol synchronization errors were presented. First, I have studied the capacity of the binary deletion channel and considered the problem of computing the number of subsequences when symbols are deleted from a string. In the second part of the thesis, I have presented new classes of error-correcting codes robust against synchronization errors. Finally, I have studied two problems using a synchronization approach, namely opportunistic spectrum access in cognitive radio systems and worst-case nonzero-error interactive communication. In this chapter, I present the main interesting open problems related to the topics presented in my thesis as well as suggestions for further research.

9.1 Capacity Bounds for Channels with Synchronization Errors

It would be interesting to extend the techniques presented in Chapter 3 to derive numerical bounds for the capacity of more general channels admitting insertions and deletions of symbols, and for which only very fragmentary results have been published. Channels allowing deletions and duplications are especially of interest due to their numerous practical applications, and this certainly is an extremely challenging problem. One of the main difficulties to overcome is that an arbitrarily large number of bits can be inserted between two consecutive input bits, resulting in output sequences that can be arbitrarily longer than the input sequences.

It would also be interesting to find closed-form expressions and an efficient algorithm to calculate the number of sequences that can be obtained when inserting and deleting symbols from a string by modifying the techniques presented in Chapter 4.

9.2 Nonlinear Trellis Codes for Channels with Synchronization Errors

The study of nonlinear trellis synchronization-correcting codes presented in Chapter 6 is a very rich problem. This being said, there are several interesting questions to answer and many difficulties to overcome in order to obtain codes that can be implemented in practical systems.

9.2.1 Code Construction

We now have promising graph structures for codes against synchronization errors, as well as efficient decoding algorithms. One of the main remaining difficulties is to allocate the output symbols to the edges in the graphs to obtain codes with good performance. Although it is possible to survey the entire space of codes for simple encoder graphs, it is computationally prohibitive to do so for larger structures. Work on small rectangular graphs showed that there is a significant difference between the average performance over all the symbol configurations and the performance of the best ones, and as such approaches based on randomly assigning symbols to graph edges are bound to fail. Two techniques could be applied. The first one is to find good symbol allocations for small trellises and to assemble them to construct larger ones. Preliminary work indicates that this approach seems a good way to optimize the trellis encoders as the number of columns increases. For trellises with large columns metaheuristic optimization techniques like hybrid evolutionary algorithms [36] could be implemented.

Another question to investigate further is the potential propagation of the synchronization errors for an arbitrarily long time. For instance, the code presented in Figure 6.12 contains a cycle of four 0s, and as a result synchronization is lost if four bits are deleted in a large run of zeros, no matter how far apart the deletions are. More precisely, this ripple effect can potentially occur when the trellis encoder can generate several shifted versions of a substring. It would be interesting to study if codes robust against ripple error propagation have a better performance than codes without this property over channels affected by realistic time noise.

9.2.2 Timing Errors

Synchronization errors come in several flavors. For instance, one can imagine a channel where symbols can be inserted and substituted, but not deleted, or a channel with a lot of deletion errors but very few random symbol insertions. Other examples are channels affected by timing errors and intersymbol interference, like in magnetic recording, where peakshift-errors are common. The techniques presented in this chapter can be extended to correct most types of channels affected by synchronization errors, but one particularly stands out. On the majority of the applications, synchronization errors are caused by varying sampling rates of the system devices, resulting in deletion and duplication errors (no random insertions). Consequently, the most practical problem of this chapter is to construct efficient codes for such channels. Two difficulties need to be overcome. First, the Viterbi decoding algorithm has to be modified, this time to correct deletion and duplication errors. I am confident that this can be done using a blending of the algorithms for duplication and synchronization errors, but it remains to be seen if the computational complexity of resulting algorithm can be reduced without significantly affecting its performance. Second, efficient bit configurations need to be found. Again, preliminary work makes me think that it will not be a problem, and that efficient configurations for deletion errors will also offer a good performance when used for channels affected by deletion and duplication errors.

9.2.3 Soft-Information Algorithms

Another very interesting question is how soft-information can be used to improve the performance of the codes. There is no research on the subject, and to the best of my knowledge this is due to the fact that besides the codes of Davey and Mackay [20], the codes presented in this chapter are the only synchronization-correcting codes that can easily take advantage of soft-information from the channels. This is one more point in their favor, since this could probably lead to interesting coding gains.

A simple yet insightful scenario worth studying is to use one-dimensional amplitude modulation, like 2-AM, with inaccurate clocks between the receiver and transmitter. A slower receiver clock creates deletion errors, a faster receiver clock creates duplication errors, and an irregular clock a mixture of deletion and duplication errors. This could allow to compare the bit error-rates with the same codes used with hard decisions. Another application that could take advantage of soft-information is the opportunistic spectrum access problem presented in Chapter 7.

Table 9.1: Multiplicity of the subsequence $\mathbf{r} = 0010$ in the string $\mathbf{v} = 010$.

	ϵ	$v_1 = 0$	$v_2 = 1$	$v_3 = 0$
ϵ	1	1	1	1
$r_1 = 0$	0	1	1	2
$r_2 = 1$	0	0	1	1
$r_3 = 0$	0	0	0	1
$r_4 = 0$	0	0	0	0

Table 9.2: Multiplicity of the subsequence $\mathbf{r} = 0010$ in the string $\mathbf{v} = 011$.

	ϵ	$v_1 = 0$	$v_2 = 1$	$v_3 = 1$
ϵ	1	1	1	1
$r_1 = 0$	0	1	1	1
$r_2 = 1$	0	0	1	2
$r_3 = 0$	0	0	0	0
$r_4 = 0$	0	0	0	0

9.2.4 Complexity Issues

I have shown in Section 6.3 that the computational complexity of the Viterbi decoding algorithm sharply increases when insertions and / or deletions and / or substitutions can occur together. However, all kinds of savings are possible. For instance, the algorithm does not always need to calculate all the k-prefix synchronization distances, because some are trivial or useless. The complexity can also be significantly reduced by bounding the maximum synchronization offset the algorithm can tolerate. It would be useful to study how these tradeoffs and others affect the performance of the codes, especially for practical implementation purposes

One of the disadvantages of trellis encoders using rectangular graphs is that the entire state diagram has to be stored in memory. This is in contrast of convolutional codes, which can be defined with linear sequential circuits. An interesting question is to investigate whether it is possible to compress the symbol configurations using simple functions without sacrificing the performance of the codes. The fact that efficient synchronization codes are intrinsically nonlinear and intrinsically as acyclic as possible makes me conjecture that this is a challenging task.

9.2.5 Maximum-Likelihood Decoding

It is not clear how far the final survivor paths for all the variations of the Viterbi algorithm in this chapter are from the maximum-likelihood paths, nor how much additional memory

is required to obtain maximum-likelihood decoding algorithms. For deletion errors, all the information required to find the maximum-likelihood codewords can be obtained using a generalized version of the algorithm to compute the multiplicity of a subsequence in a string presented in Section 4.5 (similar algorithms can easily be derived for insertion, duplication and synchronization errors), more precisely from the multiplicity of the first k symbols of $\mathbf{r}$ in the potential codewords, for $0 \leq k \leq |\mathbf{r}|$. The problem is that although the generalized multiplicity can be calculated recursively on the symbols of the trellis paths, it is not possible to define an order relation for it.

Consider again the example from Figure 6.17 presented in Section 6.7. Tables 9.1 and 9.2 contain the multiplicity of $\mathbf{r}$ in the two branches coming from the all-zero state at time $t = 0$. Comparing the last column of both tables, one can see that sometimes the largest value comes from the branch 010, and other times from the branch 011. Hence, if the algorithm wants to find the maximum-likelihood codeword, it can only discard a path if all its last column entries are smaller than or equal to the corresponding entries in one of the other survivor paths at the same state in the decoding process. An interesting problem is therefore to examine, on average and in the worst-case, the number of survivor paths that need to be kept in order to achieve maximum-likelihood decoding. Another problem of interest is to devise an order relation for the generalized multiplicity of the received sequence in the codewords allowing quasi-maximum-likelihood decoding.

9.3 Opportunistic Spectrum Access for Cognitive Radio Systems

The framework for the spectrum access problem in cognitive radio systems presented in Chapter 7 can be used for studying more complex channel models as well as implementing other types of error-correcting codes. For instance, it would be interesting to extend these channel models to include AWGN channels and more detailed modeling of the primary user detection.

The nature of the error-correcting decoding schemes indicates that an accurate estimation of the design parameters of the cognitive radios, especially the accuracy of their primary user detection, will be valuable to the decoder in effectively exploiting the capacity of the channels. It would certainly be useful to understand through simulations how much this knowledge (or lack thereof) affects decoding.

Spectrum sensing using energy detection is well suited for the use of soft information;

it would be interesting to study how much it can improve the overall performance of the codes, especially for the synchronization approach. It would also be interesting to study possible tradeoffs between the erasure and the synchronization approaches, i.e., coding techniques that can use the side-information provided by the potential positions of the deletion errors to increase transmission rates while remaining reasonably robust against variation in primary user activity.

9.4 Worst-Case Private Coin Randomized Interactive Communication

I have not been able to completely characterize the private coin randomized model presented in Chapter 8, and in this section I discuss two open problems.

9.4.1 One-Way Private Coin Randomized Complexity

I have proved that one round of communication is almost optimal for private coin randomized interactive communication, but is one round optimal? If not, is there a k such that an optimal k-round protocol always exists, i.e., a k such that

$$\hat{R}_k^\epsilon(S) \leq \hat{R}_\infty^\epsilon(S) + o(\hat{R}_\infty^\epsilon(S))?$$

The lower bounds for the private coin randomized complexity presented in this chapter do not restrict the number of rounds between $P_\mathcal{X}$ and $P_\mathcal{Y}$. Furthermore, it is not clear if existing techniques for proving lower bounds for other models of computation like communication complexity of nonboolean functions can be used to derive stronger lower bounds in the interactive communication setting. As for upper bounds, although it is not obvious how randomized protocols can use interaction to reduce the number of bits that need to be transmitted, the best upper bound, Theorem 8.19, is not always tight.

9.4.2 Deterministic Model Versus Private Coin Randomized Model

The private coin randomized model allows nearly optimal one-way protocols, but it does not seem that it can be more efficient than determinism when interaction is allowed. In

fact, I conjecture that the worst-case deterministic and worst-case private coin randomized models are equivalent, i.e.,

$$\hat{R}^{\epsilon}_{\infty}(S) \sim \hat{C}_{\infty}(S).$$

The first thing to remark is that the two lower bounds for the private coin randomized complexity presented in Section 8.3 are asymptotically equal to Results 8.3 and 8.4 for the deterministic model. I have shown in the same section that the models are equivalent when $P_{\mathcal{Y}}$'s maximum ambiguity is constant, which includes all of the maximally unbalanced pairs such as the league problem, i.e, pairs with $\widehat{a_{\mathcal{Y}}} \in O(1)$ and $\widehat{a_{\mathcal{X}}} \in \Omega(|X|)$. In Section 8.7, I have proved that the models are equivalent for balanced and symmetric support sets. Thus, in order to resolve the conjecture, one needs to study problems for which the private coin randomized complexity is not tightly bounded by Lemmas 8.9 and 8.13, i.e., problems that are neither balanced nor too unbalanced.

Bibliography

[1] Sachin Agarwal, David Starobinski, and Ari Trachtenberg. On the scalability of data synchronization protocols for PDAs and mobile devices. *IEEE Network Magazine*, July/Aug. 2002.

[2] Rudolf Ahlswede, Ning Cai, and Zhen Zhang. On interactive communication. *IEEE Transactions on Information Theory*, 43(1):22–37, 1997.

[3] Noga Alon and Alon Orlitsky. Repeated communication and Ramsey graphs. *IEEE Transactions on Information Theory*, 41(5):1276–1289, 1995.

[4] Noga Alon and Alon Orlitsky. Source coding and graph entropies. *IEEE Transactions on Information Theory*, 42(5):1329–1339, 1996.

[5] Gaurav Bansal, Jahangir Hossain, and Vijay K. Bhargava. Adaptive power loading for OFDM-based cognitive radio systems. In *Proc. 2007 IEEE International Conference on Communications (ICC)*, pages 5137–5142, June 2007.

[6] Gaurav Bansal, Md. Jahangir Hossain, Praveen Kaligineedi, Hugues Mercier, Chris Nicola, Umesh Phuyal, Md. Mamunur Rashid, Kapila C. Wavegedara, Ziaul Hasan, Majid Khabbazian, and Vijay K. Bhargava. Some research issues in cognitive radio networks. In *Proceedings of the 8th IEEE Africon Conference in Africa (AFRICON)*, Windhoek, Namibia, 26-28 September 2007. Invited Paper.

[7] Leonard D. Baumert, Robert J. McEliece, and Henk C. A. van Tilborg. Symbol synchronization in convolutionally coded systems. *IEEE Transactions on Information Theory*, 25(3):362–365, 1979.

[8] P. A. H. Bours. On the construction of perfect deletion-correcting codes using design theory. *Designs, Codes and Cryptography*, 6:5–20, 1995.

[9] Patrick A. H. Bours. Construction of fixed-length insertion/deletion correcting runlength-limited codes. *IEEE Transactions on Information Theory*, 40(6):1841–1856, 1994.

[10] Gilles Brassard and Louis Salvail. Secret-key reconciliation by public discussion. In *Advances in Cryptology: Proceedings of Eurocrypt '93*, volume 765 of *Lectures Notes in Computer Science*, pages 410–423. Springer-Verlag, 1994.

[11] L. Calabi. On the computation of Levenshtein's distances. Report TM-9-0030, Parke Mathematical Laboratories, Inc., Carlisle, MA, 1967.

[12] L. Calabi and W. E. Hartnett. Some general results of coding theory with applications to the study of codes for the correction of synchronization errors. *Information and Control*, 15(3):235–249, Sep. 1969.

[13] Willem A. Clarke and Hendrik C. Ferreira. A new linear, quasi-cyclic multiple insertion/deletion correcting code. In *Proc. 2003 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, volume 2, pages 899–902, 2003.

[14] Willem A. Clarke and Hendrik C. Ferreira. Coding for insertion/deletion error propagation effects associated with fixed length decoding windows. In *Proc. 2004 IEEE International Symposium on Information Theory (ISIT)*, page 228, 2004.

[15] Graham Cormode, Mike Paterson, Süleyman Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Proc. eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 197–206, 2000.

[16] Thomas M. Cover and Mung Chiang. Duality between channel capacity and rate distortion with two-sided state information. *IEEE Transactions on Information Theory*, 48(6):1629–1638, June 2002.

[17] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, second edition, 2006.

[18] F. H. C. Crick, J. S. Griffith, and L. E. Orgel. Codes without commas. In *Proc. National Academy of Sciences in the United States of America*, volume 43, pages 416–421, 1957.

[19] L. J. Cummings. Aspects of synchronizable coding. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1:67–84, 1987.

[20] Matthew C. Davey and David J. C. MacKay. Reliable communication over channels with insertions, deletions and substitutions. *IEEE Transactions on Information Theory*, 47(2):687–698, 2001.

[21] Suhas Diggavi and Matthias Grossglauser. On information transmission over a finite buffer channel. *IEEE Transactions on Information Theory*, 52(3):1226–1237, Mar. 2006.

[22] Suhas N. Diggavi and Matthias Grossglauser. On transmission over deletion channels. In *Proc. 39th Annual Allerton Conference on Communication, Control, and Computing*, 2001.

[23] Suhas N. Diggavi and Matthias Grossglauser. Bounds on the capacity of deletion channels. In *Proc. 2002 IEEE International Symposium on Information Theory (ISIT)*, page 421, 2002.

[24] R. L. Dobrushin. Shannon's theorems for channels with synchronization errors. *Problems of Information Transmission*, 3(4):11–26, 1967. Translated from *Problemy Peredachi Informatsii*, 3(4):18-36, 1967 (in Russian).

[25] L. Dolecek and V. Anantharam. A synchronization technique for array-based LDPC codes in channels with varying sampling rate. In *Proc. 2006 IEEE International Symposium on Information Theory (ISIT)*, pages 2057–2061, 2006.

[26] A. S. Dolgopolov. Capacity bounds for a channel with synchronization errors. *Problems of Information Transmission*, 26(2):111–120, 1990. Translated from *Problemy Peredachi Informatsii*, 26(2):27-37, 1990 (in Russian).

[27] Eleni Drinea and Michael Mitzenmacher. A simple lower bound for the capacity of the deletion channel. Available at http://www.eecs.harvard.edu/~michaelm/postscripts/deletiontemp.ps.

[28] Eleni Drinea and Michael Mitzenmacher. Improved lower bounds for i.i.d. deletion channels. In *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, 2004.

[29] Eleni Drinea and Michael Mitzenmacher. On lower bounds for the capacity of deletion channels. In *Proc. 2004 IEEE International Symposium on Information Theory (ISIT)*, page 227, 2004.

[30] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[31] Pavol Duris, Zvi Galil, and Georg Schnitger. Lower bounds on communication complexity. In *Proc. 16th annual ACM Symposium on Theory of Computing (STOC)*, pages 81–91, 1984.

[32] Peter Elias. Coding for noisy channels. In *Proc. IRE Convention Record*, volume 43, pages 37–46, 1955.

[33] Tomàs Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM Journal on Computing*, 24(4):736–750, 1995.

[34] H. C. Ferreira, W. A. Clarke, A. S. J. Helberg, K. A. S. Abdel-Gaffar, and A. J. Han Vinck. Insertion/deletion correction with spectral nulls. *IEEE Transactions on Information Theory*, 43(2):722–732, 1997.

[35] R. Fuji-Hara, Y. Miao, J. Wang, and J. Yin. Directed $B(K, 1; v)$ with $K = \{4, 5\}$ and $\{4, 6\}$ related to deletion/insertion-correcting codes. *Journal of Combinatorial Designs*, 9(2):147–156, 2001.

[36] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[37] Robert G. Gallager. Sequential decoding for binary channels with noise and synchronization errors. Lincoln Lab Group Report 2502, 1961. Available at http://web.mit.edu/gallager/www/pages/pubs.html.

[38] E. N. Gilbert. Synchronization of binary messages. *IRE Transactions on Information Theory*, 6(4):470–477, 1960.

[39] Solomon W. Golomb, J. R. Davey, I. S. Reed, H. L. Van Trees, and Jack J. Stiffler. Synchronization. *IEEE Transactions on Communications*, pages 481–491, 1963.

[40] Solomon W. Golomb and Basil Gordon. Codes with bounded synchronization delay. *Information and Control*, 8:355–372, 1965.

[41] Solomon W. Golomb, Basil Gordon, and L. R. Welch. Comma-free codes. *Canadian Journal of Mathematics*, 10(2):202–209, 1958.

[42] Venkatesan Guruswami. List decoding with side information. In *Proc. 18th IEEE Annual Conference on Computational Complexity (CCC)*, pages 300–309, 2003.

[43] Godfrey H. Hardy and Edward M. Wright. *An introduction to the theory of numbers.* Oxford University Press, 1979.

[44] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 627–636, 1996.

[45] Simon Haykin. Cognitive radio: Brain-empowered wireless communications. *IEEE Journal on Selected Areas in Communications*, 23(2):201–220, Feb. 2005.

[46] Albertus S. J. Helberg and Hendrik C. Ferreira. On multiple insertion/deletion correcting codes. *IEEE Transactions on Information Theory*, 48(1):305–308, 2002.

[47] Daniel S. Hirschberg and Mireille Régnier. Tight bounds on the number of string subsequences. *Journal of Discrete Algorithms*, 1(1):123–132, 2000. Available at http://www.ics.uci.edu/~dan/pubs/JDA.ps.

[48] Kees A. Schouhamer Immink. *Coding Techniques for Digital Recorders.* Prentice Hall Professional Technical Reference, 1991.

[49] Syed Ali Jafar and Sudhir Srinivasa. Capacity limits of cognitive radio with distributed and dynamic spectral activity. *IEEE Journal on Selected Areas in Communications*, 25(3):529–537, Apr. 2007.

[50] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding.* IEEE Press, 1999.

[51] Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5(3/4):191–204, 1995.

[52] Aleksandar Kavcic and Ravi Motwani. Insertion/deletion channels: Reduced-state lower bounds on channel capacities. In *Proc. 2004 IEEE International Symposium on Information Theory (ISIT)*, page 229, 2004.

[53] J. E. Kelley. The cutting-plane method for solving convex programs. *SIAM Journal on Applied Mathematics*, 8:703–712, 1960.

[54] Majid Khabbazian, Hugues Mercier, and Vijay K. Bhargava. Severity analysis and countermeasures for the wormhole attack in wireless ad hoc networks. Submitted, IEEE Transactions on Wireless Communications, May 2007, submission number Paper-TW-May-07-0536. Revised, November 2007. Accepted for publication, January 2008.

[55] Majid Khabbazian, Hugues Mercier, and Vijay K. Bhargava. Wormhole attack in wireless ad hoc networks: Analysis and countermeasure. In *Proceedings of the 2006 IEEE Global Telecommunications Conference (GLOBECOM)*, San Francisco, USA, 27 November - 1 December 2006.

[56] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466, 1952.

[57] Eyan Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[58] C. Y. Lee. Some properties of nonbinary error-correcting codes. *IRE Transactions on Information Theory*, 4(2):77–82, 1958.

[59] Vladimir I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1(1):8–17, 1965. Translated from *Problemy Peredachi Informatsii*, 1(1):12-25, 1965 (in Russian).

[60] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady*, 10(8):707–710, Feb. 1966. Translated from *Doklady Akademii Nauk SSSR*, 163(4):845-848, 1965 (in Russian).

[61] Vladimir I. Levenshtein. On perfect codes in deletion and insertion metric. *Discrete Mathematics and Applications*, 2(3):241–258, 1992. Translated from *Diskretnaya Matematika*, 3(1):3-20, 1991 (in Russian).

[62] Vladimir I. Levenshtein. Efficient reconstruction of sequences. *IEEE Transactions on Information Theory*, 47(1):2–22, Jan. 2001.

[63] Shu Lin and Daniel J. Costello. *Error Control Coding.* Paerson Prentice Hall, second edition, 2004.

[64] Zhenming Liu and Michael Mitzenmacher. Codes for deletion and insertion channels with segmented errors. In *Proc. 2007 IEEE International Symposium on Information Theory (ISIT)*, 2007. To appear.

[65] A. Mahmoodi. Existence of perfect 3-deletion-correcting codes. *Designs, Codes and Cryptography*, 14:81–87, 1998.

[66] Rudolf Mathon and Tran Van Trung. Directed t-packings and directed t-Steiner systems. *Designs, Codes and Cryptography*, 18:187–198, 1999.

[67] Hugues Mercier. Réconciliation et complexité de la communication de données corrélées. Mémoire de maîtrise, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada, 2003. Reconciliation and Communication Complexity of Correlated Data, Master's Thesis, in French.

[68] Hugues Mercier and Vijay K. Bhargava. Nonlinear trellis codes for channels with synchronization errors. In Preparation. To be submitted to the IEEE Transactions on Information Theory.

[69] Hugues Mercier and Vijay K. Bhargava. A note on variable-length multiple-synchronization-correcting codes. In Preparation. To be submitted to the IEEE Transactions on Information Theory.

[70] Hugues Mercier and Vijay K. Bhargava. Improved bounds for the capacity of the binary deletion channel. In *Proceedings of the 2006 International Symposium on Information Theory and its Applications (ISITA)*, pages 423–427, Seoul, Korea, 29 October - 1 November 2006.

[71] Hugues Mercier, Majid Khabbazian, and Vijay K. Bhargava. On the number of subsequences when deleting symbols from a string. *IEEE Transactions on Information Theory.* Submitted, October 2006, submission number 6-751. Revised, March 2007. Accepted for publication, February 2008.

[72] Hugues Mercier, Pierre McKenzie, and Stefan Wolf. Worst-case nonzero-error interactive communication. *IEEE Transactions on Information Theory.* Submitted, May 2006, submission number 6-302. Revised, March 2007. Accepted for publication, October 2007.

[73] Hugues Mercier, Pierre McKenzie, and Stefan Wolf. Worst-case randomized interactive communication. In *Proceedings of the 2005 IEEE International Symposium on Information Theory (ISIT)*, pages 430–434, Adelaide, Australia, 4-9 September 2005.

[74] Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2213–2218, 2003.

[75] Michael Mitzenmacher. Capacity bounds for sticky channels. 2007. Available at http://www.eecs.harvard.edu/~michaelm/postscripts/stickytemp.pdf.

[76] Takuo Mori and Hideki Imai. Viterbi decoding considering synchronization errors. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E79-A(9):1324–1329, 1996.

[77] Moni Naor, Alon Orlitsky, and Peter Shor. Three results on interactive communication. *IEEE Transactions on Information Theory*, 39(5):1608–1615, 1993.

[78] Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, pages 67–71, 1991.

[79] Chris Nicola, Hugues Mercier, and Vijay K. Bhargava. Error-correcting codes for dynamic spectrum allocation in cognitive radio systems. In *Proceedings of the 2007 International Symposium on Signals, Systems & Electronics (URSI ISSSE)*, pages 247–250, Montréal, Canada, 30 July - 2 August 2007. Invited Paper.

[80] Alon Orlitsky. Worst-case interactive communication I: Two messages are almost optimal. *IEEE Transactions on Information Theory*, 36(5):1111–1126, 1990.

[81] Alon Orlitsky. Worst-case interactive communication II: Two messages are not optimal. *IEEE Transactions on Information Theory*, 37(4):995–1005, 1991.

[82] Alon Orlitsky. Average-case interactive communication. *IEEE Transactions on Information Theory*, 38:1534–1547, 1992.

[83] Alon Orlitsky. Interactive communication of balanced distributions and of correlated files. *SIAM Journal on Discrete Mathematics*, 6(4):548–564, 1993.

[84] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[85] Edwin P. D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4(4):356–372, 1988.

[86] Edward A. Ratzer. *Error-correction on non-standard communication channels.* PhD thesis, University of Cambridge, 2003.

[87] Ron M. Roth and Paul H. Siegel. Lee-metric BCH codes and their application to constrained and partial-response channels. *IEEE Transactions on Information Theory*, 40(4):1083–1096, 1994.

[88] Reihaneh Safavi-Naini and Yejing Wang. Traitor tracing for shortened and corrupted fingerprint. In *Proc. 2002 ACM Workshop on Digital Rights Management*, volume 2696 of *Lectures Notes in Computer Science*, pages 81–100. Springer-Verlag, 2003.

[89] Leonard J. Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory*, 45(7):2552–2557, 1999.

[90] F. F. Sellers. Bit loss and gain correction code. *IRE Transactions on Information Theory*, 8(1):35–38, 1962.

[91] Ubolthip Sethakaset and T. Aaron Gulliver. Differential amplitude pulse-position modulation for indoor wireless optical channels. In *Proc. 2004 IEEE Global Telecommunications Conference (GLOBECOM)*, 2004.

[92] Nabil Shalaby, Jianmin Wang, and Jianxing Yin. Existence of perfect 4-deletion-correcting codes with length six. *Designs, Codes and Cryptography*, 27:145–156, 2002.

[93] Claude E. Shannon. Channels with side information at the transmitter. *IBM Journal of Research and Development*, 2(4):289–293, Oct. 1958.

[94] B. S. Sharif, J. Neasham, O. R. Hinton, and A. E. Adams. A computationally efficient doppler compensation system for underwater acoustic communications. *IEEE Journal Of Oceanic Enginering*, 25(1):52–61, 2000.

[95] Amin Shokrollahi. Raptor codes. 2003 IEEE International Symposium on Information Theory (ISIT), 2003. Recent Results Session, available at http://www.inference.phy.cam.ac.uk/mackay/dfountain/RaptorPaper.pdf.

[96] David Slepian and Jack K. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4):471–480, 1973.

[97] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. Published electronically at www.research.att.com/njas/sequences/, 1996-2007.

[98] Neil J. A. Sloane. On single-deletion-correcting codes. In K. T. Arasu and Á. Seress, editors, *Codes and Designs (Ray-Chaudhuri Festschrift)*, volume 10 of *Ohio State University Mathematical Research Institute Publications*, pages 273–291. de Gruyter, 2002.

[99] Neil J. A. Sloane. Challenge problems: Independent sets in graphs. Published electronically at http://www.research.att.com/~njas/doc/graphs.html, 2005.

[100] David Starobinski, Ari Trachtenberg, and Sachin Agarwal. Efficient PDA synchronization. *IEEE Transactions on Mobile Computing*, 2(1):40–51, 2003.

[101] Theo G. Swart and Hendrik C. Ferreira. Insertion/deletion correcting coding schemes based on convolutional coding. *Electronics Letters*, 38(16):871–873, 2002.

[102] Theo G. Swart and Hendrik C. Ferreira. A note on double insertion/deletion correcting codes. *IEEE Transactions on Information Theory*, 49(1):269–273, Jan. 2003.

[103] Theo G. Swart, Hendrik C. Ferreira, and Marco P. F. dos Santos. Using parallel-interconnected Viterbi decoders to correct insertion/deletion errors. In *Proc. 7th IEEE Africon Conference in Africa (AFRICON)*, pages 341–344, 2004.

[104] Jeffrey D. Ullman. On the capabilities of codes to correct synchronization errors. *IEEE Transactions on Information Theory*, 13(1):95–105, 1967.

[105] R. R. Varshamov. An arithmetic function applicable to coding theory. *Soviet Physics-Doklady*, 10:185–187, 1965. Translated from *Doklady Akademii Nauk SSSR*, 161(3):540-543, 1965 (in Russian).

[106] R. R. Varshamov and G. M. Tenengolts. Codes which correct single asymmetric errors. *Automation and Remote Control*, 26(2):286–290, 1965. Translated from *Avtomatika i Telemekhanika*, 26(2):288-292, 1965 (in Russian).

[107] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[108] N. D. Vvedenskaya and R. L. Dobrushin. The computation on a computer of the channel capacity of a line with symbol drop-out. *Problems of Information Transmission*, 4(3):76–79, 1968. Translated from *Problemy Peredachi Informatsii*, 4(3):92-95, 1968 (in Russian).

[109] Michel Willem. *Minimax Theorems*. Progress in Nonlinear Differential Equations and Their Applications. Birkhäuser, 1996.

[110] Andrew Chi-Chih Yao. Probabilistic computations: toward a unified measure of complexity. In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

[111] Andrew Chi-Chih Yao. Some complexity questions related to distributed computing. In *Proc. 11th ACM Symposium on Theory of Computing (STOC)*, pages 209–213, 1979.

[112] Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 420–428, 1983.

[113] Jianxing Yin. A combinatorial construction for perfect deletion-correcting codes. *Designs, Codes and Cryptography*, 23:99–110, 2001.

[114] Zhen Zhang and Xiang-Gen Xia. Three messages are not optimal in worst case interactive communication. *IEEE Transactions on Information Theory*, 40(1):3–10, 1994.