Data Compression and Run-Length-Limited ISI-Mitigation (RLIM) Coding for Molecular Communication

by

Melih Şahin

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical and Electronics Engineering

KOÇ ÜNİVERSİTESİ

September 1, 2025

Data Compression and Run-Length-Limited ISI-Mitigation (RLIM) Coding for Molecular Communication

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Melih Şahin

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Prof. Dr. Özgür B. Akan (Advisor)

Prof. Dr. Fatih Alagöz

Assist. Prof. Dr. Murat Kuşcu

Date: _____

To my grandparents,

Muzaffer Şahin and Hatice Şahin

ABSTRACT

Data Compression and Run-Length-Limited ISI-Mitigation (RLIM) Coding for Molecular Communication

Melih Şahin

Master of Science in Electrical and Electronics Engineering

September 1, 2025

Molecular Communication (MC) enables information transfer at the nanoscale by encoding messages into sequences of released molecules, but its practical deployment is hindered by two fundamental constraints: the high cost of molecule releases and severe inter-symbol interference (ISI) arising from residual molecules in the diffusion channel. This thesis develops coding techniques to tackle each challenge. We first introduce source-coding schemes that reduce both average code length and molecule usage. Building on an MC-adapted Huffman baseline (MoHuffman), we propose Optimized Molecular Prefix Coding (MoPC) to select a prefix codebook with minimal expected length and fewest number of 1-symbols. To push compression further, we derive Molecular Arithmetic Coding (MoAC) using an existing constrained arithmetic coding construction scheme and show its superior efficiency over substitution arithmetic coding (SAC), our different adaptation of arithmetic source coding to MC. Finally, we design Molecular Arithmetic with Prefix Coding (MoAPC) to ensure unique decodability under finite-precision arithmetic. Using two nucleotide alphabets, we then demonstrate that MoAPC has a better compression ratio than MoPC. Through MC simulations, the effectiveness of the proposed methods is also shown. To mitigate ISI at the channel coding level, we develop an infinite family of Run-Length-Limited ISI-Mitigation (RLIM) codes with a corresponding built-in error correction algorithm. We then demonstrate, via binomial and diffusion channel simulations, that RLIM codes reduce bit-error rate compared to prominent coding schemes. Together, these contributions lay a comprehensive foundation for reliable and efficient diffusion-based Molecular Communication.

ÖZETÇE

Moleküler Haberleşme için Veri Sıkıştırma ve Koşu Uzunluğu Sınırlı ISI Azaltım (RLIM) Kodlaması
Melih Şahin
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
1 Eylül 2025

Moleküler Haberleşme (MH), bilgi mesajlarını salınan molekül dizilerine kodlayarak nano ölçekte bilgi aktarımını mümkün kılar; ancak gerçek hayatta uygulanabilirliği, molekül salınımının yüksek maliyeti ve difüzyon kanalındaki zamanla biriken moleküllerden kaynaklanan şiddetli semboller arası girişim (ISI) olmak üzere iki temel kısıtlamayla engellenmektedir. Bu tez, her bir zorluğu ele alan kodlama teknikleri geliştirmektedir. Öncelikle ortalama kod uzunluğunu ve molekül tüketimini düşüren kaynak-kodlama şemaları tanıtıyoruz. MH'ye uyarlanmış Huffman temelli bir yöntem (MoHuffman) üzerinden, beklenen kod uzunluğunu en aza indiren ve "1" sembolü sayısını mümkün olduğunca azaltan Optimize Moleküler Önek Kodlama'yı (MoPC) öneriyoruz. Sıkıştırmayı bir adım daha ileri taşımak için, var olan bir kısıtlı aritmetik kodlama inşa yapısından yararlanarak Moleküler Aritmetik Kodlama'yı (MoAC) geliştiriyor ve bunun, aritmetik kaynak kodlamanın MH'ye farklı bir uyarlamamız olan Yerine Koymalı Aritmetik Kodlama'ya (SAC) göre üstün verim sunduğunu gösteriyoruz. Ayrıca, sonlu hassasiyetli aritmetikte benzersiz çözülebilirliği garanti altına almak amacıyla Önek Kodlamalı Moleküler Aritmetik Kodlama'yı (MoAPC) tasarlıyoruz. İki adet nükleotit alfabesi kullanarak MoAPC'nin MoPC'den daha yüksek sıkıştırma oranı sağladığını gösteriyoruz. Yaptığımız MH simülasyon sonuçları da önerdiğimiz yöntemlerin yararlılığını doğrulamaktadır. ISI'yı kanal kodlama katmanında hafifletmek için, Koşu Uzunluğu Sınırlı ISI-Azaltım (RLIM) sonsuz kod ailesini gömülü hata düzeltme algoritması ile birlikte geliştiriyoruz. Binom ve difüzyon kanalı simülasyonları, RLIM kodlarının öne çıkan diğer kodlama yöntemlerine kıyasla bit-hata oranını düşürdüğünü göstermektedir. Bütün bu katkılar, difüzyona

dayalı Moleküler İletişim için güvenilir ve verimli bir temel oluşturmaktadır.

ACKNOWLEDGMENTS

I would like to express my profound gratitude to Prof. Dr. Özgür Barış Akan, my thesis advisor. His unwavering support and contagious enthusiasm for research have shaped every stage of this work. The breadth of knowledge and perspective I gained under his guidance far exceeds anything I could have imagined when I began this journey.

I am deeply grateful to Prof. Dr. Öznur Özkasap, my undergraduate supervisor, for her invaluable guidance and constant encouragement throughout my undergraduate studies and beyond.

I extend my sincere thanks to Prof. Dr. Engin Erzin for welcoming me into his lab as an undergraduate intern in the summer of 2022; the experience broadened my research perspective and provided skills that have been invaluable in my later work.

I wholeheartedly thank Prof. Dr. Alptekin Küpçü for the insightful discussions and constructive feedback that have strengthened my approach to research.

I thank my committee members, Prof. Dr. Fatih Alagöz and Assist. Prof. Dr. Murat Kuşcu, for the time and insight they invested in reviewing this thesis.

I dedicate my sincere appreciation to my family for standing by me throughout this journey. Their constant support and encouragement have carried me forward. Whatever I have accomplished reflects their belief in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AC	Arithmetic Coding
BCSK	Binary Concentration Shift Keying
BER	Bit Error Rate
EOF	End Of File (terminator symbol)
ISI	Inter-Symbol Interference
MC	Molecular Communication
MoAC	Molecular Arithmetic Coding
MoAPC	Molecular Arithmetic with Prefix Coding
MoPC	Molecular Prefix Coding
RLIM	Run-Length-Limited ISI-Mitigation
SAC	Substitution Arithmetic Coding
SER	Symbol Error Rate

Chapter 1

INTRODUCTION

This chapter and Abstract are based on our paper [52][1.1] and preprint [53]. Molecular communication (MC) is a bio-inspired communication method aiming to transmit information in the characteristics of chemical signals. These signals are released and detected by molecular entities, such as cells or nano-scale devices. MC holds great promise for precision therapeutics, in-body diagnostic nanosensor networks, and environmental biosensing [1]. The simplest form of MC involves diffusion, where each molecule moves pseudo-randomly through space via Brownian Motion[2, 1].

Diffusion-based MC leads to a phenomenon known as inter-symbol-interference (ISI), which occurs when the information molecules transmitted in the communication system overlap or interfere with each other in subsequent signal intervals, causing the communication channel to have a high memory component [3, 4]. This thesis develops novel source-coding (data compression) and channel-coding methods to mitigate ISI in diffusion-based MC channels.

1.1 System Model

We assume a molecular communication via diffusion (MC) channel, where a point transmitter releases a predetermined number of information molecules into the environment at the start of each signal interval with a constant symbol duration t_s. These information molecules move in a pseudo-random manner through the 3-dimensional

[1.1] © 2024 IEEE. Personal use permitted; for any other use, contact IEEE.

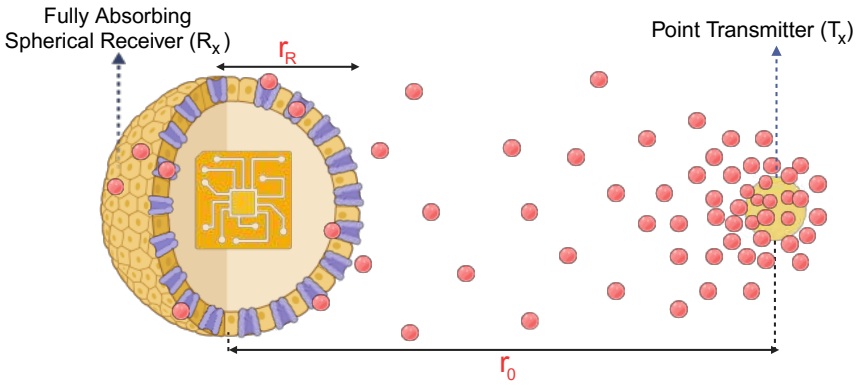


Figure 1.1: MC Channel [1.2]

fluidic environment, following the principles of Brownian motion as described in [2]. The receiver absorbs any information molecule that comes into contact with its surface and keeps track of the count of these molecules for each signal interval.

One of the most wide-spread MC modulation techniques [5] is the binary concentration shift keying (BCSK) [6]. In BCSK, if a current signal interval corresponds to a 1-bit, a certain number of information molecules are released into the environment from the transmitter at the very start of that signal interval. If a current signal interval corresponds to a 0-bit, no information molecules are released. This paper uses BCSK modulation and assumes perfect time synchronization between the transmitter and receiver, both of which are common approaches in most MC coding studies [1].

This process is depicted in Fig. 1.1, where r_R is the radius of the spherical receiver, r_0 is the distance between the center of the spherical receiver and the point transmitter. At each time step, Δt (in seconds), the position of a molecule (x, y, z) is updated along each coordinate axis as follows [7, 8]:

$$\Delta x, \Delta y, \Delta z \sim \mathcal{N}(0, 2 \cdot D \cdot \Delta t), \tag{1.1}$$

where D is the diffusion coefficient of the fluidic environment in $\mu m^2/s$.

[1.2]Created in BioRender. Şahin, M. (2024) https://BioRender.com/f03d790

The analytical function [9] that gives the probability that an information molecule touches (i.e., gets absorbed by) the fully absorbing spherical receiver until time t (in seconds) is

$$F(t) = \frac{r_R}{r_0} \cdot \text{erfc}\left(\frac{r_0 - r_R}{\sqrt{4 \cdot D \cdot t}}\right), \tag{1.2}$$

where r_R denotes the radius of the spherical receiver in µm, r_0 represents the distance between the center of the spherical receiver and the point transmitter in µm, and $\text{erfc}(\cdot)$ denotes the complementary error function. The i^{th} channel coefficient, p_i, is defined to be the probability that a molecule, emitted in the k^{th} signal interval, gets absorbed during the $(k+i-1)^{th}$ signal interval. Channel coefficients [1] are

$$p_i = F(i \cdot t_s) - F((i-1) \cdot t_s), \quad i = 1, 2, \ldots, I, \tag{1.3}$$

where I is the channel memory, and t_s is the signal interval duration. Let b_j represent the bit information transmitted in the $(j-1)^{th}$ previous signal slot (with b_1 indicating the bit from the current signal interval and b_2 indicating the bit from the previous interval). Through basic probability theory [10], the number of expected molecules, N_{exp}, to be detected in the current signal interval can then be modeled by using a summation of binomial distribution expressed as

$$N_{exp} \sim \left(\sum_{j=1}^{I} b_j \cdot \mathcal{B}(M, p_j)\right) + \mathcal{N}(0, \sigma_n^2), \tag{1.4}$$

where M is the number of information molecules emitted at the start of each signal interval for the transmission of a 1-bit, $\mathcal{B}(M, p_j)$ denotes the binomial distribution with M trials and a success probability of p_j, and $\mathcal{N}(0, \sigma_n^2)$ is a Gaussian distribution with mean 0 and variance σ_n^2 which we use to model the counting noise inside the receiver. Using the formula for the mean of a binomial distribution, $(b_i \cdot M \cdot p_i)$ gives the expected number of molecules released in the $(i-1)^{th}$ previous slot and detected in the current interval. For instance, $(b_1 \cdot M \cdot p_1)$ is the expected number molecules that are both released and detected in the current signal slot. For

sufficiently large values of $\mathbb{E}(N_{exp})$, the binomial distribution at (1.4) can well be approximated by the following Gaussian normal distribution [10]:

$$N_{exp} \sim \mathcal{N}\Big(\sum_{j=1}^{I} b_j \cdot M \cdot p_j, \big(\sum_{j=1}^{I} b_j \cdot M \cdot p_j \cdot (1 - p_j) \big) + \sigma_n^2 \Big) \tag{1.5}$$

Since a summation of binomial distributions can be analytically non-trivial, the normal distribution given in (1.5) proves to be highly beneficial in many analytical derivations as we demonstrate in Section 3.2.4.

Chapter 2

MOLECULAR ARITHMETIC CODING (MOAC) AND OPTIMISED MOLECULAR PREFIX CODING (MOPC)

2.1 Background and Motivation

This chapter is based on our paper [52][2.1]. Source coding (data compression) methods reduce the number of bits needed to encode the data [11]. For MC via diffusion, reducing the number of bits required for lossless data transmission can improve channel quality by allowing for longer signal intervals and thus reduce ISI. Conversely, channel coding introduces additional bits for data redundancy, which can shorten signal intervals and increase ISI. However, certain channel coding techniques have error-correcting properties that can outweigh the disadvantage of shorter signal intervals, as demonstrated in [12]. Integrating source coding with channel coding strategies leverages the strengths of both approaches. A recent study supports the benefits of this integrated approach: Simulations in [13] demonstrated that integrating Huffman source coding [14] with ISI-mitigating channel codes [12] results in significant improvements in symbol error rate (SER) compared to using Huffman coding alone.

Arithmetic coding [15] is known to be better than Huffman coding in terms of its data compression rate [16] [17]. Additionally, arithmetic coding is utilised in many of the most effective biological data compression algorithms [11]. Considering MC will mostly find application areas in biological organisms [2], the adaption of

[2.1] Personal use permitted; for any other use, contact IEEE.

arithmetic source coding, of which use is highly prevalent in bio-informatics, into MC is necessary. Accordingly, in this paper, we develop two novel arithmetic source coding methods: SAC and MoAC. The latter outperforms SAC and is built by adapting a general constrained-arithmetic-coding construction [18].

This part of our thesis is organized as follows: Section 2.2 presents our proposed coding schemes: optimized molecular prefix coding (MoPC*), substitution arithmetic coding (SAC), molecular arithmetic coding (MoAC), and molecular arithmetic with prefix coding (MoAPC). Section 2.3 discusses detection and error correction algorithms. Section 2.4 provides a comparative analysis of the proposed source coding methods.

2.2 Arithmetic and Prefix Source Coding for Molecular Communication

2.2.1 Optimized Molecular Prefix Coding (MoPC)

In the context of source coding, assigning each symbol with a code in such a way that none of the codes is a prefix of another code ensures unique decodability. For codebooks under no restriction, one technique to find a length-wise optimal prefix codebook is the Huffman coding [14].

Authors of [13] combine source coding with the error correction properties of one of the most successful MC channel codes [12] by not allowing consecutive 1-bits in each resultant Huffman code through substituting each 1-bit with a 10. In this paper, we abbreviate the MC-adapted Huffman coding [13] as MoHuffman. Though MoHuffman highly improves the symbol and word error rate values compared to Huffman coding, it does not always produce a length-wise optimal codebook. Consider the alphabet (a, b, c, d, e) with respective probabilities $(0.201, 0.201, 0.201, 0.199, 0.198)$. MoHuffman produces a prefix code space $a \rightarrow 00, b \rightarrow 010, c \rightarrow 1010, d \rightarrow 1000, e \rightarrow 10010$, with expected code length 3.595.

However, $a \to 000, b \to 010, c \to 100, d \to 0010, e \to 1010$ is a better prefix codebook with expected code length 3.397.

In literature, a prefix codebook whose codes avoid consecutive 1-bits and end with a 0-bit (or equivalently start with a 0-bit) is known as a $(1, \infty)$ constrained Huffman coding [19, 20]. In this paper, we will refer to it as Molecular Prefix Coding (MoPC), as MoPC also incorporates the ISI-mitigating error correction technique [12] (given in Algorithm 2.2 in Section 2.3) whose usage for MC prefix coding is proposed in MoHuffman [13]. Furthermore, MoPC also has an optimization constraint on transmission of 1-bits as will be clarified in this subsection. Finding a length-wise optimal MoPC is equivalent to the problem of constructing optimal prefix-free codes with unequal letter cost, as the cost of 1-bits and 0-bits can be taken as 2 and 1 respectively. Polynomial algorithms for this problem exist, with time complexity as little as $\mathcal{O}(n^2)$, where n is the symbol alphabet size [21], [22].

However, minimizing the coding length is not the only criterion; reducing the transmission of 1-bits is also important. In this paper, we first select the prefix codebook with the shortest length. If there are multiple length-wise optimal codebooks, we then choose the one that produces the lowest expected number of 1-bits. We abbreviate an optimal MoPC constructed under these conditions as MoPC*. For instance, when given an alphabet (a, b, c) with respective probabilities $(0.4, 0.3, 0.3)$, the length-wise optimal MoPC codebooks $a \to 10, b \to 010, c \to 00$ and $a \to 00, b \to 010, c \to 10$ both produce the same expected length, $0.4 \cdot 2 + 0.3 \cdot 3 + 0.3 \cdot 2 = 2.3$. However, the first codebook has an average per-symbol 1-bit transmission of 0.7, while the second codebook has an average per-symbol 1-bit transmission of 0.6. Therefore, since the second codebook would consume fewer information molecules, it should be preferred over the first one; and it is actually a MoPC*.

To the best of our knowledge, there is not a polynomial algorithm to perform this task of choosing the codebook with the least average number of 1-bits among the length-wise optimal MoPC codebooks. Accordingly, MoPC* codebooks in the

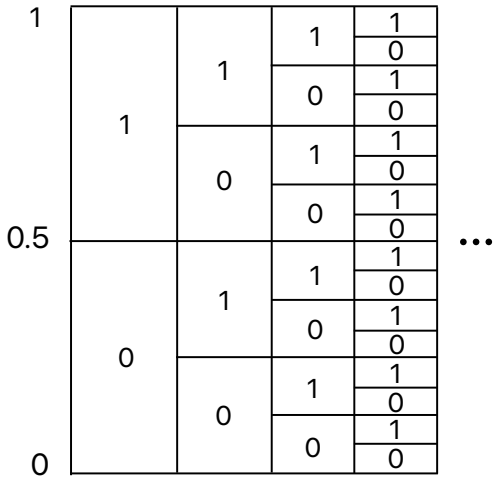


Figure 2.1: Code Space Depiction of AC © 2024 IEEE [2.1]

performance evaluation section of this paper have all been derived by a trivial brute-force algorithm, which searches through the space of all possible MoPC codebooks.

2.2.2 Classical Arithmetic Coding (AC)

This section presents an accessible introduction to classical arithmetic coding (AC), as its definitions and underlying logic will be frequently referenced throughout the subsequent sections. Each AC code (i.e., bit sequence) in the code space has a unique interval associated with it. For instance, '1' is associated with $[0.5, 1)$; and '01' is associated with $[0.25, 0.5)$ as in Fig. 2.1. Let $b_1 b_2 ... b_n$ be a code of length n. Then its corresponding interval, $[k, l)$ is defined as follows [15, 16]:

$$[\sum_{k=1}^{n} b_k \cdot 2^{-k}, \quad 2^{-n} + \sum_{k=1}^{n} b_k \cdot 2^{-k}) \tag{2.1}$$

When given a finite alphabet, together with the probabilities of each symbol, a unique interval $[a, b) \subseteq [0, 1)$ can be associated with each word [16]. Let *word* be a symbol sequence with length n such that $word(i)$ denotes the i^{th} symbol of the *word*, where $1 \leq i \leq n$. Let our *alphabet* set, which includes N symbols be represented by a bijective ordering function $ord : alphabet \rightarrow \{1, ..., N\}$. This way, each symbol of *alphabet* can be associated with a number between 1 and N. Let

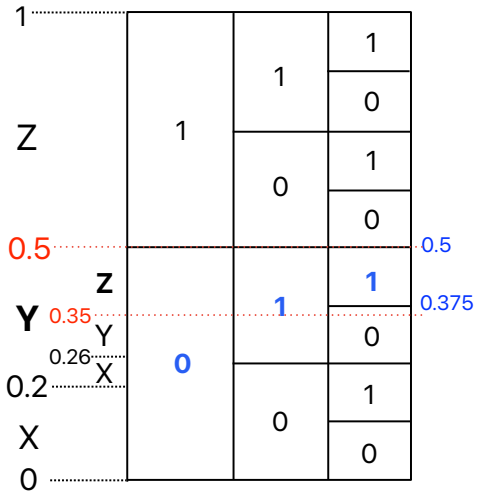


Figure 2.2: AC Encoding of the Exemplary Word YZ © 2024 IEEE [2.1]

$prob : \{1, ..., N\} \rightarrow [0, 1]$ be a function which maps each number k to the probability of the symbol associated with the number k. For $i = 1, \ldots, N$, define:

$$c(i) = \sum_{k=1}^{i-1} \text{prob}(k), \quad d(i) = \sum_{k=1}^{i} \text{prob}(k) \tag{2.2}$$

The interval $[a, b)$ associated with *word* can be recursively obtained using composition of functions, as follows [15].

Let $a_1 = c(ord(word(1)))$ and $b_1 = d(ord(word(1)))$.

For all $2 \leq i \leq n$, let

$$a_i = a_{i-1} + (c(ord(word(i)))) \cdot (b_{i-1} - a_{i-1}) \tag{2.3}$$

$$b_i = a_{i-1} + (d(ord(word(i)))) \cdot (b_{i-1} - a_{i-1}) .$$

Define $[a, b) = [a_n, b_n)$.

Through using this unique interval $[a, b)$, a bit sequence can be associated with the given *word*. This bit sequence is the shortest code, having the corresponding interval, $[k, l)$, computed using (2.1), conforming $[k, l) \subseteq [a, b)$. If the information of the length of the bit sequence is provided to the decoder, an end-of-file (EOF) symbol at the end of each word is not needed [16].

This encoding procedure will now be illustrated through a simple example. Consider an exemplary EOF-included alphabet (X, Y, Z) with respective probabilities $(0.2, 0.3, 0.5)$, where Z serves as the EOF symbol. Define the ordering function $ord : \{X, Y, Z\} \rightarrow \{1, 2, 3\}$ as being $ord(X)$=1, $ord(Y) = 2$, and $ord(Z) = 3$. Then, using (2.2) and (2.3), for the exemplary word YZ, the corresponding interval is calculated to be $[0.35, 0.5)$. The shortest bit sequence, whose corresponding interval is a subset of $[0.35, 0.5)$, is 011 as shown in Fig. 2.2. The interval of 011 is $[0.375, 0.5)$ from (2.1), which satisfies $[0.375, 0.5) \subseteq [0.35, 0.5)$ as it should.

For EOF-included decoding, assume a bit sequence $\mathbf{b} = b_1 b_2 \dots b_n b_{n+1} b_{n+2} \dots b_{n+h}$, which comprises appended encodings of ensuing words, is given. Let the interval of $\mathbf{b}$ be $[k_1, l_1)$ computed based on the definition at (2.1). Let the encoding of the initial world be $b_1 b_2 ... b_n$ with its interval being $[k_2, l_2)$, computed using (2.1). The decoding of the initial word of this whole bit sequence, $\mathbf{b}$, is the longest word in which the only EOF character is at its end, and whose corresponding interval $[a, b)$ satisfies $[k_1, l_1) \subseteq [a, b)$. The decoder does not know the position of the initial word's final bit, b_n. Since an EOF character is available, this is not an issue. Because the interval of $b_1 b_2 ... b_n b_{n+1} b_{n+2} ... b_{n+h}$ is a subset of the interval of $b_1 b_2 ... b_n$ (i.e $[k_1, l_1) \subseteq [k_2, l_2)$). Therefore, any decoding of the bit sequence $b_1 b_2 ... b_n \cup S$, where S represents all possible bit sequences of varying lengths, and $\cup$ is the sequence appending operator, would all be decoded as the initial word whose encoding is $b_1 b_2 ... b_n$.

As an instance, for the exemplary alphabet given in the paragraph two before, the codes $011 \cup S$, where S is any bit sequence, would all be decoded as YZ. For EOF-excluded decoding, the decoder must have the knowledge of where the initial word's encoding ends in the given whole bit sequence. Then the initial words is decoded as being the longest word whose encoding identically matches the given initial portion of the whole encoding.

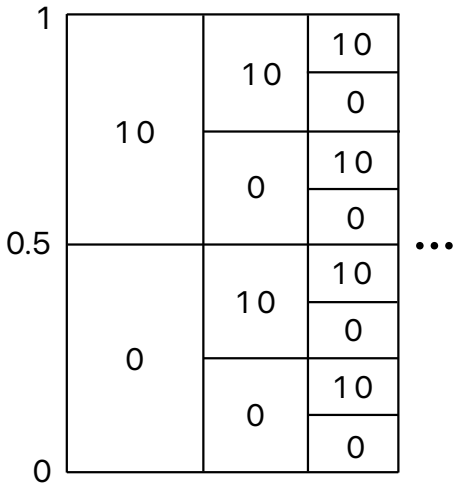


Figure 2.3: Code Space Depiction of SAC © 2024 IEEE [2.1]

2.2.3 Substitution Arithmetic Coding (SAC)

Our purpose is to create an arithmetic coding method that ensures each 1-bit is followed by at least one 0-bit. This property is needed to achieve the error-correction property in an MC channel as proposed in [12]. One simple but inefficient solution, as we propose, is to substitute each 1-bit in the AC with a 10. For instance the word YZ, using the encoding algorithm of the AC, is first encoded to be 011. Then, in this new scheme, it would be converted to 01010. Then to decode 01010, we would substitute each 10 with a 1-bit, converting it back to its original form, 011. Then, using the decoder algorithm for the AC, the corresponding word YZ would be found. We name this new scheme as the Substitution Arithmetic Coding (SAC). This adaption technique is very similar to the approach followed in MoHuffman [13]. The code space depiction of SAC is given in Fig. 2.3.

2.2.4 Molecular Arithmetic Coding (MoAC)

In this section, building on the general scheme outlined in [18], we construct an arithmetic source coding method, which we name MoAC, with error correction capabilities achieved by avoiding consecutive 1-bits. We will demonstrate both theo-

retically and practically that MoAC offers substantial advantages in coding length and power consumption over SAC. The ISI-mitigating codes defined in [12], without the existence of a 1-bit condition in each code, are equivalent to run-length-limited (RLL), also known as constrained, codes of order $(1, \infty)$. More generally, an RLL code of order (d, k) enforces that a 1-bit must be followed by at least d 0-bits, and no more than k consecutive 0-bits can appear [23]. The scheme in [18] provides a method for constructing arithmetic codes for any RLL code of order (d, k), specifically for channel coding. Additionally, in [19], this approach is extended to data compression for binary source alphabets. The proposed MoAC data compression scheme is designed to accommodate source alphabets of any finite size, extending beyond the binary case.

We will follow the general scheme in [18], applying it to $(1, \infty)$ RLL codes, with some modifications for MoAC. Specifically, instead of adhering to the strict design where a 0-bit occupies the first column, MoAC removes this bit and requires each encoding to terminate with a 0-bit. This modification aligns MoAC with the structure of MoPC, proving useful for later defining MoAPC. Let $C(n)$ denote all $(1, \infty)$ RLL codes of length n that do not require the existence of a 0-bit at the start of each code. First two terms of $C(n)$, which is known as the 1-limited sequence [24], are as follows:

$$C(1) = \{0, 1\}, \quad C(2) = \{00, 01, 10\} \tag{2.4}$$

To construct an arithmetic encoder for $C(n)$, growth rate W of $C(n)$ needs to be derived [18]. Let $|C(n)|$ denote the total number of bit sequences (i.e., codes) inside $C(n)$. From (2.4), $|C(1)| = 2$ and $|C(2)| = 3$. Note that $|C(n)|$ conforms to the recursive relation $|C(n)| = |C(n-1)| + |C(n-2)|$ [24]. This is because, all codes of $C(n)$ can be formed by inserting 0 to the start of all $C(n-1)$, and by inserting 10 to the start of all $C(n-2)$. Thus $|C(n)| = Fibonacci[n+2]$. Then, W for $C(n)$, using properties of $Fibonacci[n]$, is given by:

$$W = \lim_{n\to\infty} \frac{|C(n)|}{|C(n-1)|} = \phi = \frac{1+\sqrt{5}}{2} = 1.618\ldots \tag{2.5}$$

Note that, $(1, \infty)$ RLL happens to have to the same W constant, ϕ [20]. Using ϕ, the code space structure of MoAC, as shown in Fig. 2.4, will now be defined. For $C(n)$ codes, a 1-bit must be followed by a 0-bit, while a 0-bit can be followed by either a 0-bit or 1-bit. Furthermore, the general scheme in [18] enforces a 1-bit to have an interval height of $(1/W)^{n+1}$, where n is the column index. Please note that since we have not enforced the code space to begin with a 0-bit, we shifted the original formula of $(1/W)^n$ by 1. Based on these rules, we recursively construct the MoAC code space as follows:

At the 1^{st} column, the 0-bit is assigned the interval $[0, 1/\phi)$, and the 1-bit interval is assigned the interval $[1/\phi, 1)$ as shown in Fig 2.4. For any 1-bit in any n^{th} column with interval assignment $[x, y)$, there is a 0-bit in the $(n+1)^{\text{th}}$ column with interval assignment $[x, y)$. For any 0-bit in any n^{th} column with interval assignment $[x, y)$, there is a 0-bit in the $(n+1)^{\text{th}}$ column with interval assignment $[x, (x+(y\cdot\phi))/(1+\phi))$, and a 1-bit in the $(n+1)^{\text{th}}$ column with interval assignment $[(x+(y\cdot\phi))/(1+\phi), y)$. Let $b_1 b_2 ... b_n$ be a code of length n, which do not contain consecutive 1-bits. Then, we define its corresponding MoAC interval, $[k, l)$, as follows.

$$[\sum_{i=1}^{n} b_i \cdot (1/\phi)^i, \quad (1/\phi)^{(n+b_n)} + \sum_{i=1}^{n}(b_i \cdot (1/\phi)^i)) \tag{2.6}$$

Using (2.3), each word of a given alphabet can be associated with a unique MoAC interval, $[a, b)$. Then we define the MoAC encoding of a given word to be the shortest bit sequence that ends with a 0-bit and whose MoAC interval $[k, l)$, computed using (2.6), satisfies $[k, l) \subseteq [a, b)$. To illustrate the working mechanism of MoAC, we will use the same example given for AC. Let our exemplary EOF-included alphabet be (X, Y, Z) with respective probabilities $(0.2, 0.3, 0.5)$. Let our exemplary word be YZ as previously. Using (2.2) and (2.3), the interval of YZ is computed to be $[0.35, 0.5)$.

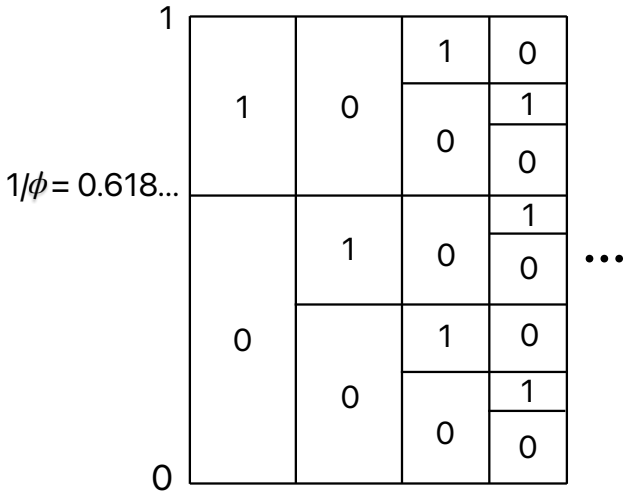


Figure 2.4: Code Space Depiction of MoAC © 2024 IEEE [2.1]

Then the shortest MoAC sequence whose interval $[k, l)$ is a subset of $[0.35, 0.5)$ is found to be 01000 as illustrated in Fig. 2.5. The MoAC interval of 01000, using (2.6), is $[(1/\phi)^2, (1/\phi)^2 + (1/\phi)^5) \approx [0.381, 0.472) \subseteq [0.35, 0.5)$. To decode 01000, we identify the shortest word where the EOF character (in this case, Z) exclusively appears at its end, and whose interval is a subset of 01000. This word is YZ.

Now it will be shown that MoAC produces shorter codes than SAC with an approximate ratio of 1 to 1.0413. In deriving this ratio, the following lemma will be used. Note that, if a code has an associated interval $[k, l)$, its interval height is defined to be $l - k$.

Lemma: The interval height values of MoAC codes having the same length n and ending with a 0-bit are all equal. Similarly, all MoAC codes of length n ending with a 1-bit have the same interval height value. For instance, MoAC codes 100, 010, and 000 have identical interval height values.

Proof: We proceed by induction. Initial induction statement for $n = 1$ clearly holds, as at the first column of Fig. 2.4, there is only one 1-bit and 0-bit. As the inductive argument, assume that what is stated at the lemma above holds for an i^{th} column, i.e. for all codes of length i. In the i^{th} column, let the interval height of each 1-bit be a, and accordingly let the interval height of each 0-bit be $a \cdot \phi$. In the

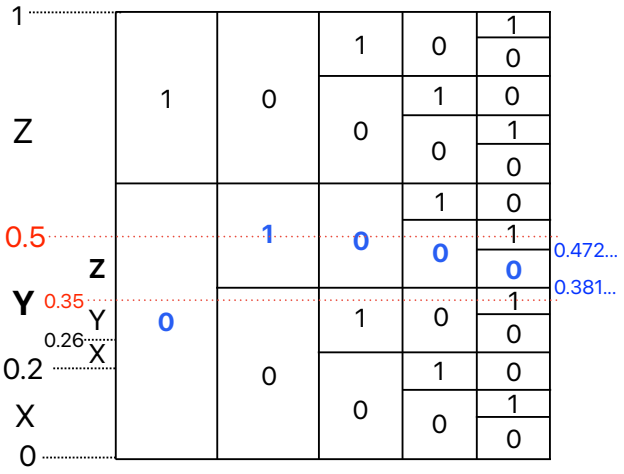


Figure 2.5: MoAC Encoding of the Exemplary Word YZ © 2024 IEEE [2.1]

$(i+1)^{\text{th}}$ column, a 0-bit can come either after a 0-bit or a 1-bit. If it comes after a 1-bit, its length is same as the 1-bit, i.e. a. If it comes after a 0-bit, its length is $(a \cdot \phi) \cdot (\phi/(1+\phi)) = a \cdot ((\phi \cdot \phi)/(1+\phi)) = a \cdot 1 = a$. And since all 1-bits in the $(i+1)^{\text{th}}$ column comes after a 0-bit, they all naturally have the same length. This concludes the inductive argument. □

Assume for any given *word* of any alphabet, the interval $[a, b)$ is assigned to it using (2.3). Let x be the height of the interval assigned to the *word* (i.e., $x = b - a$). Then for an AC code to be assigned to *word*, at least, the interval height of this code must not be bigger than x. Thus the shortest AC code that can be assigned to *word* has the length $\lceil \log_{\frac{1}{2}} x \rceil$. Since in an AC code, appearance of 1-bits and 0-bits are equally likely, on average, an AC code of length n is transformed into a SAC code of length $(1/2) \cdot n + 2 \cdot (1/2) \cdot n = (3/2) \cdot n$. Thus, for the given *word*, its shortest expected SAC encoding length is $\lfloor (3/2) \cdot \lceil \log_{\frac{1}{2}} x \rceil \rfloor$.

The MoAC encoding of *word* must end with a 0-bit; and from the Lemma and (2.6), all the MoAC codes of length n that ends with a 0-bit has an assigned interval height of $(1/\phi)^n$. Due to how MoAC code space is defined, if the equation $(1/\phi)^n \leq x/2$ holds, then it guarantees that there exist a MoAC code of length n whose interval $[k, l)$ is a subset of the interval $[a, b)$ associated with *word*. This equation

then implies that the upper bound on the length of MoAC encoding of *word* is $\lceil (log_{\frac{1}{\phi}} x/2) \rceil \leq \lceil (log_{\frac{1}{\phi}} x) \rceil + 2$. In MoAC, a 0-bit is appended to an encoding that ends with a 1-bit. Thus the final upper bound becomes $\lceil (log_{\frac{1}{\phi}} x) \rceil + 3$.

Note that, for any finite alphabet that consists of more than one symbol, the following fact can easily be proven: For any infinitesimally small positive real number ϵ, there exist a natural number N, such that the height values of the intervals, assigned to all words lengthier than N, are smaller than ϵ. Thus, for an alphabet containing more than one symbol, and for all of its sufficiently lengthy words, (2.7) gives the ratio of expected encoding length of SAC to that of MoAC:

$$\lim_{x \to 0^+} \frac{\lfloor (3/2) \cdot \lceil \log_{\frac{1}{2}} x \rceil \rfloor}{\lceil (log_{\frac{1}{\phi}} x) \rceil + 3} = (3/2) \cdot \log_2 \phi \approx 1.0413... \tag{2.7}$$

To compare the average number of 1-bits produced by SAC and MoAC, we first calculate the appearance probability of 1-bits in a MoAC code by counting the 1-bits in $C(n)$. Let $one[n]$ denote the total number of 1-bits in $C(n)$. From (2.4), $one[1] = 1$, and $one[2] = 2$. We remarked that $C(n)$ can be obtained by inserting 0 to the start of all $C(n-1)$ and by inserting 10 to the start of all $C(n-2)$. Therefore we have $one[n] = one[n-1] + one[n-2] + |C(n-2)| = one[n-1] + one[n-2] + Fibonacci[n]$. The sequence $one[n-1]$ is known as the self-convolution of the Fibonacci numbers [25].

We remind that in a MoAC code of length n, all codes must end with a 0-bit. Hence, for a MoAC code of length n, all possible codes are the elements of $C(n-1)$, each appended with a 0-bit. Thus, using the Lemma, all MoAC codes of length n appear with equal probabilities (i.e., they have the same interval heights). Consequently, one$[n-1]$ gives the expected number of all 1-bits in all MoAC codes of length n. Note that the total number of bits in all MoAC codes of length n is given by $n \cdot |C(n-1)| = n \cdot Fibonacci[n+1]$. Accordingly, the following limit, l, which we have computed using numerical methods, gives the expected ratio of 1-bits in a MoAC code:

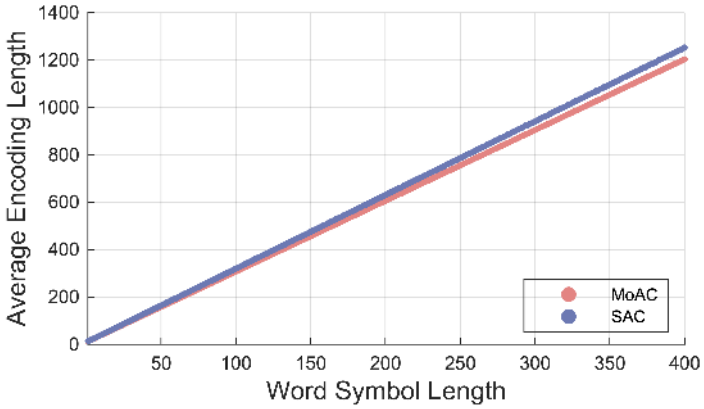


Figure 2.6: Average encoding length of MoAC and SAC © 2024 IEEE [2.1]

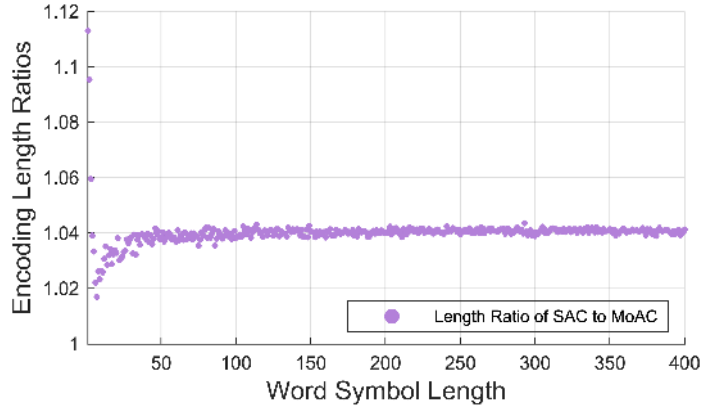


Figure 2.7: Ratio of SAC to MoAC average encoding lengths © 2024 IEEE [2.1]

$$l = \lim_{n\to\infty} (one[n-1]/(n \cdot Fibonacci[n+1])) \approx 0.276... \tag{2.8}$$

Recall that in SAC, each 1-bit produced by AC is replaced with a 10. Since in AC, the distribution of 1-bits and 0-bits are equally likely, the appearance probability of 1-bit in SAC is 1/3, while that of a 0-bit is 2/3. The expected number of 1-bits in MoAC for a word, of which encoding length is n, is $l \cdot n \approx 0.276 \cdot n$. For the same word, the number of expected 1-bits in its SAC encoding, from (2.7), is $(3/2) \cdot (\log_2 \phi) \cdot n \cdot (1/3)$. Dividing these two numbers we get $((3/2) \cdot (\log_2 \phi) \cdot n \cdot (1/3))/(0.276 \cdot n) \approx 1.257$. This shows that SAC (and AC), in average, uses 25.7

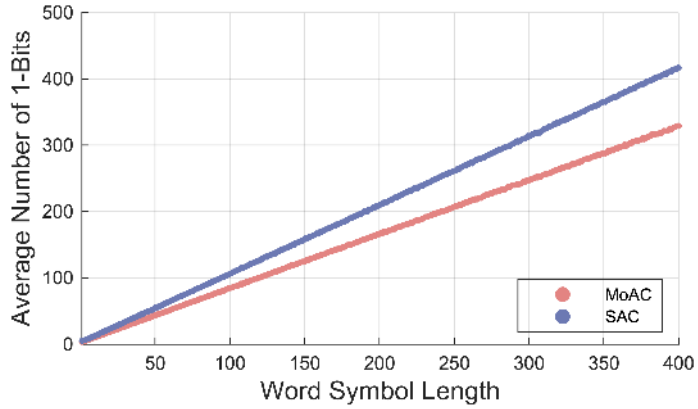


Figure 2.8: Average Number of 1-bits of MoAC and SAC © 2024 IEEE [2.1]

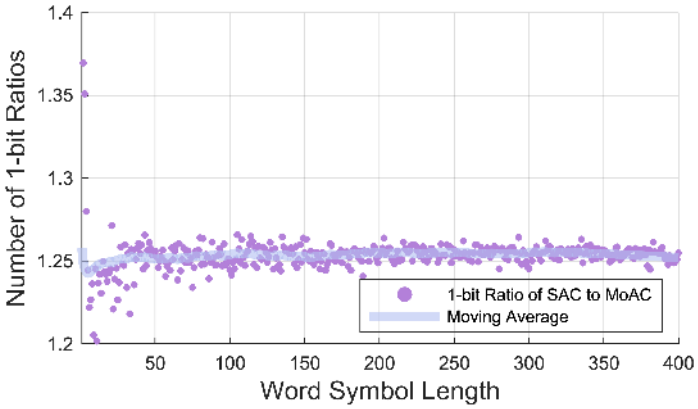


Figure 2.9: The ratios of the Average Number of 1-bits of SAC to those of MoAC © 2024 IEEE [2.1]

percent more 1-bits than MoAC does.

To reinforce these theoretical results, finite precision versions of MoAC and SAC will now be compared. Let our exemplary alphabet be (A, B, C, EOF), with corresponding respective probabilities $(0.33, 0.33, 0.33, 0.01)$. The average encoding length and number of 1-bits comparisons are given in Figs. 2.6 and 2.8 respectively. For each word length, we chose 400 random words using the symbol distributions of the exemplary alphabet. And a bit precision of 20 was designated. Fig. 2.7 and 2.9 empirically verify the theoretical length and 1-bit ratios of 1.0413, and 1.257 respectively.

Finite Precision Zero-Order MoAC

Due to the unsymmetrical nature of MoAC as can be seen in Fig. 2.4, the implementation of finite precision MoAC is non-arbitrarily different, and more complex than the finite-precision implementation of AC given in [15], [16]. The link for a GitHub repository that includes the zero-order Python implementations and pseudo-codes of MoAC and AC (both with and without EOF versions) is provided in Section 2.5.

Finite Precision Higher-Order MoAC

Once an algorithm for the zero-order MoAC is available, implementation of the higher order MoAC is trivial. We just allocate intervals for symbols according to their conditional probabilities, based on the preceding symbols. Corresponding changes in MoAC decoder can also be easily implemented. Other than this, there is no need for change in any other part of the proposed MoAC algorithm. For a better understanding, interested readers may look into the Higher-Order Modeling chapter of the book [16].

2.2.5 Molecular Arithmetic with Prefix Coding (MoAPC)

Since the number of precision bits has to be finite, unique decodability of MoAC is not guaranteed. In the Python implementation of MoAC, we have created an underflow expansion process which is non-arbitrarily different than that of the AC. Although this measure increases the accurate decodability rate of MoAC, there can still be cases where decoding errors do occur. Hence, the encoder is required to decode the MoAC encoding of the to-be-transmitted word to verify a perfect matching between the original and the resultant words. The implementation of this checking mechanism can be found in the GitHub repository, whose link is provided in the Code Availability section. If resultant words do not match, the word is encoded through MoPC*. We name this as the Molecular Arithmetic with Molecular Prefix Coding (MoAPC).

So, there needs to be a mechanism to inform the decoder if the incoming message was encoded through MoAC or MoPC*. For this purpose, the header mechanism shown in Fig. 2.10 is proposed to inform the decoder which encoding scheme was opted for. For MoAC, encoding a word, and decoding its encoding have almost the same computational cost. However, since the decoder just repeats the steps of the encoder in our implementation of MoAC, if the transmitter has a strong memory component, the decoding inside it can be computationally more efficient. In our proposed header mechanism, a 0-bit is inserted to the start of a word if it was encoded by MoAC, and the bit sequence 10 is inserted to the start of a word if it was encoded by MoPC*.

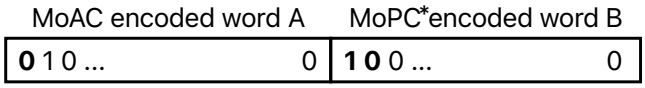


Figure 2.10: MoAPC: MoAC / MoPC* Distinguisher © 2024 IEEE [2.1]

2.3 Detection

2.3.1 Algorithms for Detection and Error Correction

We follow an almost-identical detection approach to the one introduced in [12]. Please note that the only original contributions in this subsection are the introduction of the *min* constant, redefinition of r_i^{min} [2.2], the definition of the last chunk of the bit-sequence, $\mathbf{b}^{\lfloor n/spacing \rfloor}$, and the optimization of the *spacing* constant, that was previously assigned a fixed value based on the coding block length used.

Let $\mathbf{r^i} = (r_1^i, r_2^i, ..., r_{spacing}^i)$ represent the count of the detected information molecules for corresponding signal intervals for the incoming message $\mathbf{b^i} = (b_{spacing}$.

[2.2]For ISI-Mitigating Codes [12], the parameter r_{min}^i is defined as $min(\mathbf{r^i} \setminus \{r_1^i\})$. By redefining r_{min}^i as being $non\_zero\_min(\mathbf{r^i})$, we have generalized its detection mechanism to be applicable across various coding schemes. In the Performance Evaluation section, to ensure a fair comparison, we tested ISI-Mitigating Codes [12] using both the redefined and original approaches. As shown in Figs. 2.20–2.23, both methods yield comparable error rates.

Algorithm 2.1 Detection Algorithm © 2024 IEEE [2.1]

Require: the *molecule_count_sequence* with size n, where *molecule_count_sequence*[i] denotes the number of molecules detected at the i^{th} signal interval, the scaling coefficient a, the spacing constant *spacing*, and the minimum constant *min*

let *detected_bit_sequence* be a sequence of $0s$ of size n

for $k \leftarrow 1$ **to** n **do**

$i = \lfloor (k-1)/spacing \rfloor + 1$

if $molecule\_count\_sequence[k] \geq \tau^i$ **then**

if $molecule\_count\_sequence[k] \geq min$ **then**

$detected\_bit\_sequence[k] = 1$

end if

end if

end for

return *detected_bit_sequence*

$_{(i-1)+1}, \ldots, b_{spacing\cdot(i-1)+spacing})$, where b_j denotes the j^{th} bit of the whole encoded message, and *spacing* is an integer constant. If the number of bits of the whole encoding is n, and if *spacing* does not divide n, $\mathbf{b}^{\lfloor n/spacing \rfloor}$ is defined to be the last ($spacing + n \ mod \ spacing$) bits of the whole encoding. Then, we can similarly define $\mathbf{r}^{\lfloor n/spacing \rfloor}$. The integer constant *spacing* can take the value that results in the least symbol error rate value.

Define $r^i_{max} = max(\mathbf{r^i})$, and $r^i_{min} = non\_zero\_min(\mathbf{r^i})$. If $\mathbf{r^i} = (0, 0, ..., 0)$, take r^i_{min} to be ∞. Then the optimal threshold, τ^i, of the i^{th} code-word, $\mathbf{b^i}$, can be found as

$$\tau^i = a \cdot r^i_{min} + (1-a) \cdot r^i_{max}, \tag{2.9}$$

where a is the scaling coefficient [12]. Note that $0 \leq a \leq 1$. Most importantly, this scheme assumes that each $\mathbf{b^i}$ contains at least one 1-bit. But in source coding this may not always be the case. We solve this problem by introducing another

Algorithm 2.2 Error Correction Algorithm [12] © 2024 IEEE [2.1]

Require: *detected_bit_sequence* with size n

1: **for** $j \leftarrow 1$ **to** n **do**
2: **if** $detected\_bit\_sequence[j] == 1$ and not $(j == n)$ **then**
3: $detected\_bit\_sequence[j+1] = 0$
4: **end if**
5: **end for**

channel-specific constant min which denotes the least number of molecules that a receiver could detect in the signal interval of a 1-bit. In the proposed Algorithm 2.1, If the number of molecules in a signal-interval falls below min, that signal interval is always detected to be a 0-bit.

For determining a, we adopt the pilot-signal approach given in [12], where, at the start of the communication, predetermined ensuing words are sent. Then, starting from 0 and continuing to 1, with a step size of 0.004, the decoder can determine the value of a that results in the most accurate decoding of predetermined pilot symbols in terms of symbol error rate. After the incoming message is detected using the threshold method given in Algorithm 2.1, detected bit sequence is processed through an ISI-mitigating error correction algorithm defined in [12], and given in Algorithm 2.2. This algorithm, taking the advantage of the fact that the proposed coding does not contain consecutive 1-bits, mitigates the ISI that may be caused by a preceding 1-bit.

2.3.2 Word Differentiation for EOF-Excluded Words

If two types of information molecules are available at the transmitter, words belonging to an EOF-excluding alphabet can be distinctively transmitted, irrespective of the encoding scheme used: To differentiate ensuing words, a 1-bit is transmitted at the beginning of each word's transmission using type-2 (coloured in blue) molecules,

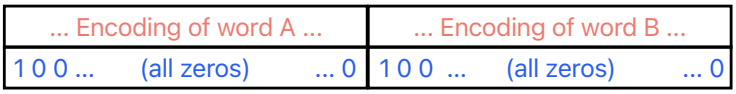


Figure 2.11: 2 Molecule Types Word Distinguisher © 2024 IEEE [2.1]

while type-1 (coloured in red) molecules are exclusively used for transmitting the encoding of each word, as shown in Fig. 2.11.

2.4 Performance Evaluation

2.4.1 Encoding Length and Power Consumption Comparisons

In Alphabets 1 and 2 (Tables 2.1 and 2.2), we represent exemplary nucleotide probability distributions of a single-strand DNA. For the Alphabet 1, we have chosen the probability values to create a non-uniform (i.e., a lower entropy) symbol distribution. In contrast, for Alphabet 2, we have selected an alphabet with a more uniform distribution (i.e., a higher entropy). This selection allows us to more thoroughly assess the performance of our proposed methods as the performance of coding schemes can vary between nearly-uniform and non-uniform symbol alphabets [16]. Alphabet 1 does not contain an EOF symbol while Alphabet 2 contains an EOF symbol. Testing the performance of MoAC (thus that of MoAPC) for both EOF-included and EOF-excluded cases are important, as the finite-precision implementation of MoAC, which is accessible in the Code Availability section, is different between EOF-included and EOF-excluded versions.

In order to minimize the expected power consumption of ISI-Mitigating, Uncoded, and Huffman coding methods, we have assigned codes that contain the fewest number of 1-bits to the symbols with the highest probabilities. For each compared method, average encoding length and average number of 1-bits comparisons for words of length from 1 to 400 are given in Figs. 2.12, 2.13, 2.16, and 2.17. For each word length, we randomly chose 400 words using the symbol distributions of

Table 2.1: Exemplary Alphabet 1 © 2024 IEEE [2.1]

Symbol	A	T	C	G
Probability	0.50	0.25	0.23	0.02
Uncoded	00	01	10	11
ISI-Mitigating [12]	0001	0010	0100	0101
Huffman	0	10	110	111
MoHuffman [13]	0	100	10100	101010
MoPC*	0	100	10100	101010

Table 2.2: Exemplary Alphabet 2 © 2024 IEEE [2.1]

Symbol	A	T	C	G	EOF
Probability	0.25	0.24	0.23	0.23	0.05
Uncoded	000	001	010	100	011
ISI-Mitigating [12]	00001	00010	00100	01000	00101
Huffman	00	10	01	110	111
MoHuffman [13]	00	100	010	10100	101010
MoPC*	000	100	010	0010	1010

the corresponding alphabet. For MoAC, AC and SAC, bits precision of 20 is designated. In both alphabets, availability of a single type of an information molecule is assumed; and the mechanism shown in Fig. 2.10 is adopted for MoAPC.

In Figs. 2.14 and 2.18, we compare the average encoding lengths of all error-correcting compression methods to the average encoding length of MoAPC. As the figures show, MoAPC has a shorter average encoding length than all these methods. During additional comparisons with other symbol alphabets (which are not shown here for the sake of brevity), MoAPC consistently had a shorter average encoding length compared to MoPC*, MoHuffman [13], SAC, and ISI-Mitigating codes [12] for words longer than an alphabet-dependent number, which is usually less than 50. In Fig. 2.18, MoPC* is better than MoHuffman for word lengths less than 81; however, for word lengths greater than 81, it is mostly outperformed by MoHuffman. The

reason for this is that the probability of the EOF symbol for Alphabet 2 is set at 0.05. As the word length increases beyond 20, the probability of the EOF symbol decreases, leading to a change in the actual probability distribution of the alphabet. This causes the MoPC* to perform in an alphabet distribution for which it was not optimized, leading to a slight reduction in its performance.

For each alphabet, in Figs. 2.15 and 2.19, accurate decoding ratio of arithmetic coding methods are given. As can be observed in these figures, each encoding that AC (and thus SAC) produces, can almost always be correctly decoded. However this is not the case for MoAC, whose accurate decoding ratio decreases as the word symbol length increases, due to its irrational nature. This phenomena further justifies the use of the uniquely decodable MoAPC. In terms of the power consumption, for the Alphabet 1, MoAPC outperforms all given methods, including its arithmetic coding competitors AC and SAC, except the Uncoded one in Fig. 2.13. In Alphabet 2, in addition to the Uncoded one, MoAPC is also outperformed by MoPC*, as shown in Fig. 2.17. This power performance variance between MoPC* and MoAPC on the exemplary alphabets indicates the alphabet-dependent nature of source coding.

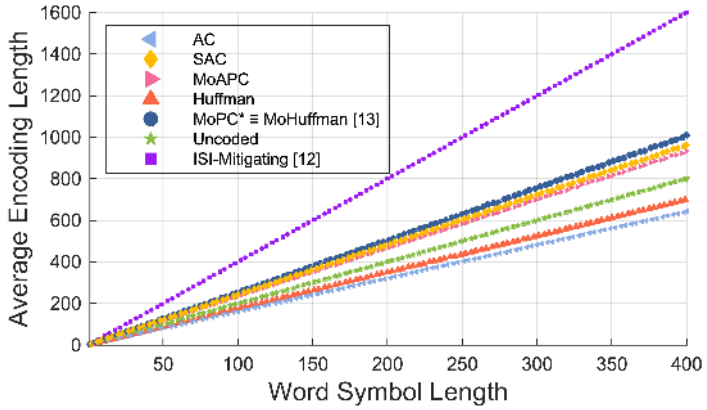


Figure 2.12: Encoding Length Comparison for Alphabet 1 © 2024 IEEE [2.1]

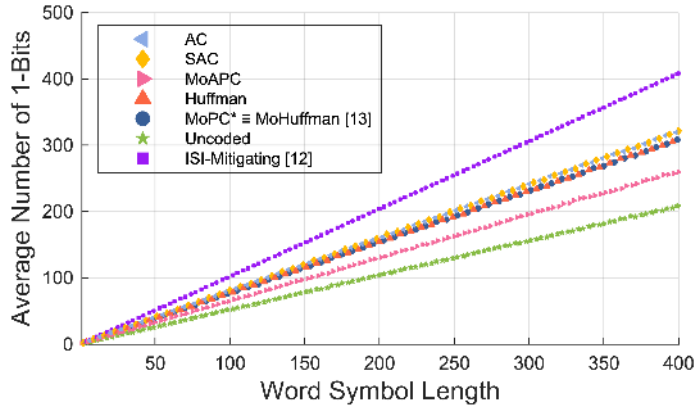


Figure 2.13: Power Consumption Comparison for Alphabet 1 © 2024 IEEE [2.1]

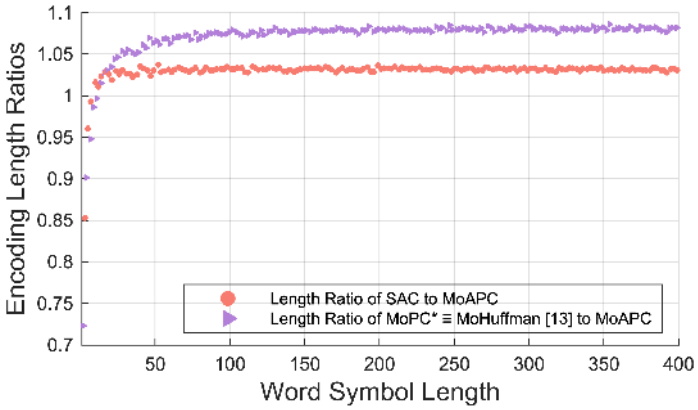


Figure 2.14: Encoding Length Ratios for Alphabet 1 © 2024 IEEE [2.1]

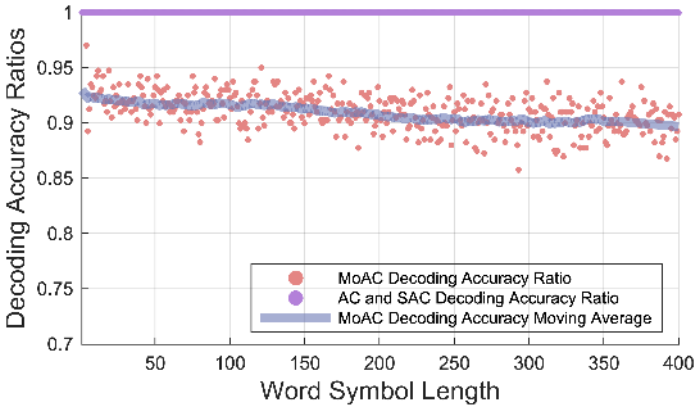


Figure 2.15: Arithmetic Accuracy Ratios for Alphabet 1 © 2024 IEEE [2.1]

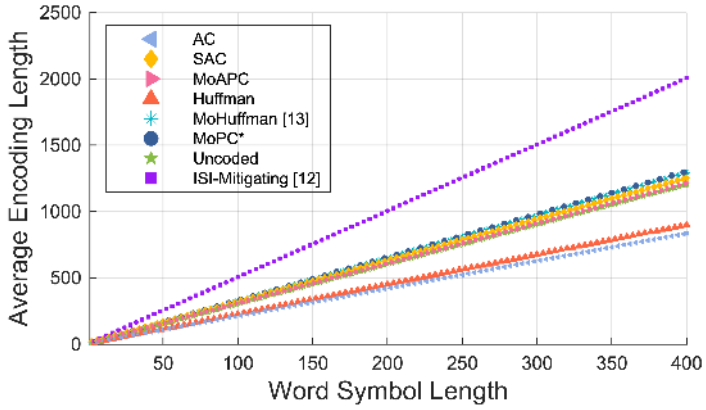


Figure 2.16: Encoding Length Comparison for Alphabet 2 © 2024 IEEE [2.1]

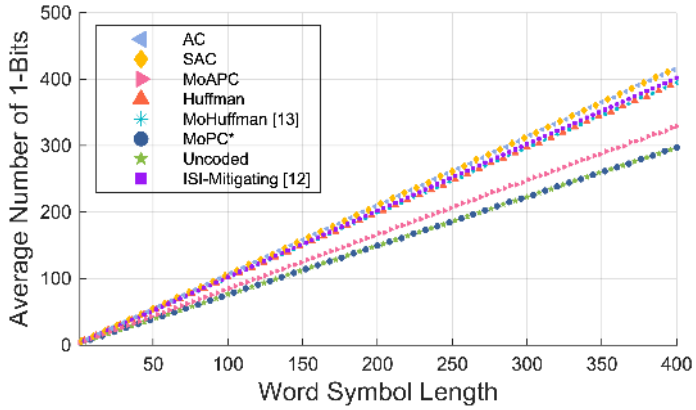


Figure 2.17: Power Consumption Comparison for Alphabet 2 © 2024 IEEE [2.1]

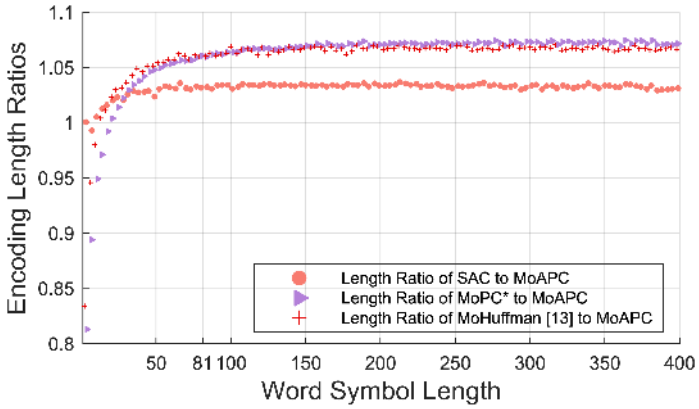


Figure 2.18: Encoding Length Ratios for Alphabet 2 © 2024 IEEE [2.1]

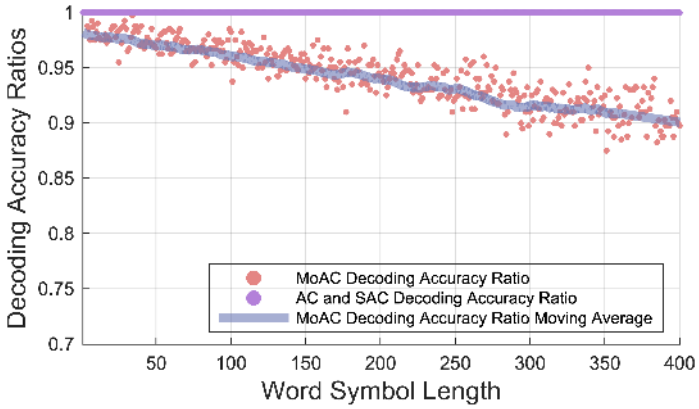


Figure 2.19: Arithmetic Accuracy Ratios for Alphabet 2 © 2024 IEEE [2.1]

2.4.2 MC Channel Simulation Results

In comparing different coding strategies for MC, normalizing the signal durations and signal powers is essential. This ensures that equal amounts of information are transmitted through various coding schemes within the same time duration while using an equal number of information molecules.

The normalization is done in the following way, as briefly outlined in [26]: Let I be the set of all information (i.e., the words) available for transmission. In a deterministic approach, each element of I appears with a probability of $1/|I|$. In a probabilistic approach, each element of I may appear with different probabilities, which sum to 1. Assume a coding scheme C_1 encodes a randomly chosen element of I, using S_1 expected number of bits, and M_1 expected number of 1-bits. Also assume that a coding scheme C_2 encodes a randomly chosen element of I, using S_2 expected number of bits, and M_2 expected number of 1-bits. Then the signal interval value of the coding scheme C_2 should be S_1/S_2 times the signal interval value of coding scheme C_1. Similarly, the molecule count per transmission of a 1-bit value for the coding scheme C_2 should be M_1/M_2 times that of the coding scheme C_1.

Table 2.3: Average Encoding Length and Power Consumption for Alphabet 1 with Word Length 20 © 2024 IEEE [2.1]

Coding Method	**Encoding Length**	**Power Consumption (Number of 1-bits)**
Uncoded	40	10.4
ISI-Mitigating [12]	80	20.4
AC	33.32009	16.53598
SAC	49.85607	16.53598
MoAPC	48.63674	13.44275
Huffman	35	15.4
MoPC* ≡ MoHuffman [13]	50.4	15.4

Table 2.4: Signal Interval and Molecule Count Normalizations for Alphabet 1 with Word Length 20 © 2024 IEEE [2.1]

Coding Method	**Signal Interval**	**Molecule Count** ($M = 100, 200, \dots$)
Uncoded	200 ms	$1 \cdot M$
ISI-Mitigating [12]	100 ms	$\lfloor 0.5098 \cdot M \rceil$
AC	240 ms	$\lfloor 0.6289 \cdot M \rceil$
SAC	160 ms	$\lfloor 0.6289 \cdot M \rceil$
MoAPC	164 ms	$\lfloor 0.7736 \cdot M \rceil$
Huffman	229 ms	$\lfloor 0.6753 \cdot M \rceil$
MoPC* ≡ MoHuffman [13]	159 ms	$\lfloor 0.6753 \cdot M \rceil$

Average encoding length and 1-bit counts of all compared coding methods are given in Tables 2.3 and 2.5 respectively for the Alphabets 1 and 2. The word length is chosen to be 20. For Alphabet 2, EOF symbol is not counted in the word

Table 2.5: Average Encoding Length and Power Consumption for Alphabet 2 with Word Length 20 © 2024 IEEE [2.1]

Coding Method	**Encoding Length**	**Power Consumption (Number of 1-bits)**
Uncoded	63	16.73684
ISI-Mitigating [12]	105	22
AC	46.83747	23.15221
SAC	69.98969	23.15221
MoAPC	68.72786	18.61808
Huffman	47.84210	22.57894
MoHuffman [13]	70.42105	22.57894
MoPC*	68.84210	16.73684

Table 2.6: Signal Interval and Molecule Count Normalizations for Alphabet 2 with Word Length 20 © 2024 IEEE [2.1]

Coding Method	**Signal Interval**	**Molecule Count** ($M = 100, 200, \dots$)
Uncoded	200 ms	$1 \cdot M$
ISI-Mitigating [12]	120 ms	$\lfloor 0.7607 \cdot M \rceil$
AC	269 ms	$\lfloor 0.7229 \cdot M \rceil$
SAC	180 ms	$\lfloor 0.7229 \cdot M \rceil$
MoAPC	183 ms	$\lfloor 0.8989 \cdot M \rceil$
Huffman	263 ms	$\lfloor 0.7412 \cdot M \rceil$
MoHuffman [13]	179 ms	$\lfloor 0.7412 \cdot M \rceil$
MoPC*	183 ms	$1 \cdot M$

length. That is, each word of Alphabet 2 comprises of 20 non-EOF symbols followed

Table 2.7: Parameters Used in the Simulation © 2024 IEEE [2.1]

Parameter	**Value**
Diffusion coefficient (D)	79.4 $\mu m^2/s$
Distances between Tx and Rx (r_0)	10 µm
Receiver radius (r_R)	5 µm
Gaussian Counting Noise Variance (σ_n^2)	0
Uncoded Signal Interval (t_s)	200 ms
Particle-Tracking Simulator Step Size (Δt)	1 ms

by the EOF symbol. For MoAPC, AC, and SAC, the encoding length and power consumption values have been computed by randomly choosing 10^6 words from each corresponding alphabet. For other methods, these values have been computed probabilistically.

In the simulation, the signal interval and molecule count values are normalized based on the values of the Uncoded method. Accordingly, the normalized signal interval and molecule count per transmission of a 1-bit values for all the compared methods are given in the Tables 2.4 and 2.6 for Alphabet 1 and 2, respectively. Note that the function, $\lfloor x \rceil$, rounds the given real number x to the nearest integer.

We implemented our particle-tracking MC simulator based on the design of the simulator given in [27], which uses the distribution at (1.1). For simulation parameters, we have used the values given in Table 2.7[2.3]. To use in the detection Algorithm 2.1, for each exemplary alphabet, we initially, on a set of 1024 random words of length 20, computed the optimal *spacing*, the optimal a, and the min[2.4] val-

[2.3]Parameters in Table 2.7 are commonly used in MC literature, representing an MC channel where human insulin hormone is an information molecule [4].

[2.4]For ISI-Mitigating codes, as the existence of a 1-bit is guaranteed at each block [12], the min value is not computed. For others, to estimate the smallest possible min value (calculated by taking the minimum value among all the number of absorbed molecules during each signal interval that corresponds to a 1-bit in the pilot signals), we scaled each calculated min by $\frac{5}{6}$.

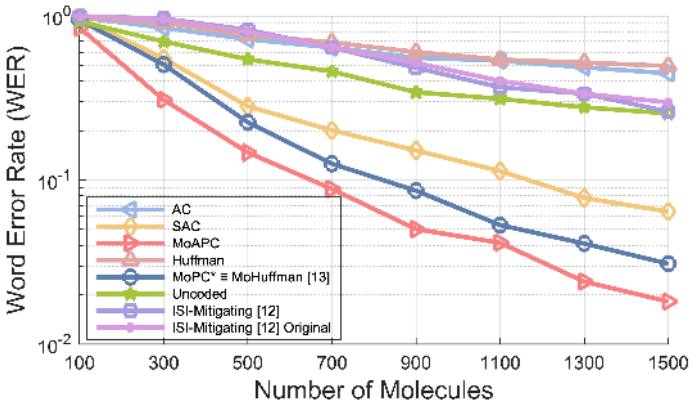


Figure 2.20: Word Error Rates of Exemplary Alphabet 1 © 2024 IEEE [2.1]

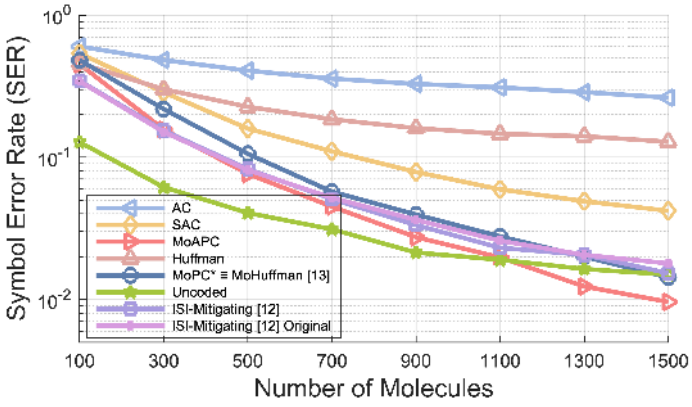


Figure 2.21: Symbol Error Rates of Exemplary Alphabet 1 © 2024 IEEE [2.1]

ues for each respective method, at each different molecule count. In the simulation, using these pre-determined values of coefficients a, *spacing* and *min* in Algorithm 2.1, for each method at each different molecule count, we sent 5120 randomly chosen words of length 20.

Word error rate (WER) is defined as the ratio of the number of decoded words that do not perfectly match their corresponding original words to the total number of transmitted words. The simulation results are shown in Figs. 2.20, 2.21, 2.22, and 2.23, giving the respective WER and SER values. For a fair comparison among EOF-excluded methods, it is assumed that the receiver knows where the encodings

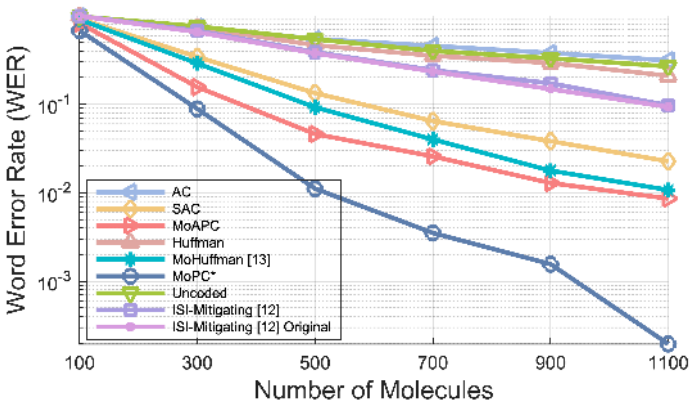


Figure 2.22: Word Error Rates of Exemplary Alphabet 2 © 2024 IEEE [2.1]

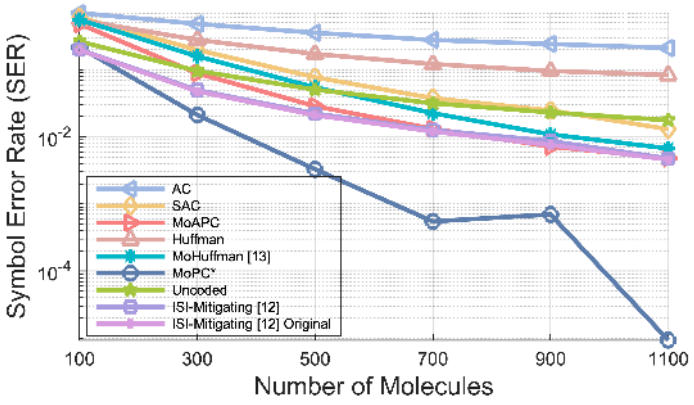


Figure 2.23: Symbol Error Rates of Exemplary Alphabet 2 © 2024 IEEE [2.1]

end for all the transmissions of Alphabet 1. The simulation results show that the proposed MoAPC consistently outperformed SAC, which in turn always outperformed AC. These findings demonstrate that we have successfully adapted arithmetic source coding to molecular communication with significantly better channel reliability. For Alphabet 1, the proposed MoAPC achieved the best WER performance, and asymptotically outperformed all others in terms of SER. For Alphabet 2, MoPC* surpassed all other methods, including its main competitors MoHuffman and Huffman, in terms of both SER and WER.

2.4.3 Self-Synchronisation Property and Future Work on MoAC

While arithmetic coding methods offer superior compression performance compared to prefix coding techniques, they lack the crucial property of self-synchronization. Self-synchronization allows a decoder to recover from bit errors after a certain number of symbols, ensuring more reliable decoding. In prefix coding, most codebooks possess this property [28]. Conversely, a single symbol error in arithmetic coding causes all subsequent symbols to be detected randomly based on the symbol distribution of the alphabet.

Unless MoAPC has a significantly better compression and power consumption advantage over MoPC*, as in Alphabet 1, it can be expected to perform inferiorly than MoPC* in highly stochastic MC channels, due to its lack of self-synchronization property. Several techniques can integrate self-synchronization into arithmetic coding. Soft decoding for error resilience is discussed in [29]. Marker methods for error detection are presented in [30, 31, 32], and error correction techniques are detailed in [33, 34]. Future research in MC source coding should prioritize the integration of these techniques into MoAC, equipping it with self-synchronization capabilities.

2.4.4 Computational Considerations for MC Source Coding

Since MC is designed for nano-scale environments, circuit designs should remain relatively simple and efficient. Although finding a MoPC* prefix codebook is currently an exponential task, once found, it requires fewer computational resources than MoAC (and thus MoAPC) during encoding and decoding. However, for higher-order source coding, the number of prefix codebooks required increases exponentially with the data compression order [16]. As shown in Figs. 2.15 and 2.19, most of the encoding in MoAPC relies on MoAC. Therefore, since MoAC offers a superior compression performance compared to MoPC*, the MoPC* component of MoAPC can be fixed at the zeroth order while still benefiting from the higher-order data compression advantages of MoAC. For higher order MC source coding, this approach

could possibly make MoAPC a more effective choice than MoPC* by reducing the exponential space requirements associated with higher-order prefix coding.

2.5 Code Availability

At https://github.com/MelihSahinEdu/MCArithmetic.git, pseudocode for MoAC, as well as Python implementations of MoAC and AC (with and without EOF versions) can be found.

Chapter 3

RUN-LENGTH-LIMITED ISI-MITIGATION (RLIM) CODING

3.1 Background and Motivation

This chapter is based on our preprint [53]. MC causes a high degree of inter-symbol interference (ISI) [1, 35], a phenomenon where the gradual accumulation of information molecules, not yet absorbed by the receiver, impairs the receiver's ability to correctly decode the intended signal [3]. Therefore, developing coding schemes that directly address the detrimental effects of ISI remains one of the central challenges in MC channel coding research.

A number of coding schemes have been used or proposed in the MC literature, including ISI-mitigating codes [12], ISI-free codes [36], Uncoded BCSK [6], and Hamming [37, 38] codes. In terms of bit error rate (BER), ISI-mitigating codes have been shown to outperform these techniques [12]. Through extensive MC simulations across diverse scenarios, we demonstrate that our novel infinite family of coding schemes, $\text{RLIM}_i(n, k)$ for $i \in \{2, 3, 4\}$, offers a significant BER advantage over prominent methods, including ISI-mitigating codes. An $\text{RLIM}_i(n, k)$ code has blocklength n and information length k (so the codebook has size 2^k), and enforces the constraint that every transmitted 1 is followed by i consecutive zeros. This run-length constraint spaces out 1s, reducing intersymbol interference and improving detection robustness in diffusive channels with memory.

The structure of this part of the thesis is as follows: Section 3.2 gives the codebook constructions for the proposed $\text{RLIM}_i(n, k)$. Detection algorithms are given

in Section 3.2.1. Sections 3.2.2 and 3.2.3 present our error-correction algorithm and compare it to Viterbi decoding baselines; in particular, we show equivalence to the last-wins variant of Viterbi decoding. An analytical estimation of static threshold is available in Section 3.2.4. In Section 3.3, MC channel simulation results are provided.

3.2 RLIM Coding and Codebook Derivation

We propose a new infinite family of codebooks, denoted by $\text{RLIM}_i(n, k)$, where $i, n, k \in \mathbb{N}$. Here, RLIM stands for run-length-limited ISI-mitigation. We first define the properties of $\text{RLIM}_i(n)$ as follows.

1. Each 1-bit is followed by i 0-bits, provided i or more available positions exist following the 1-bit. If fewer than i positions are available after the 1-bit, all subsequent bits are 0-bits.
2. Each code starts with i 0-bits.
3. Each code must contain at least one 1-bit.

The 1^{st} and 2^{nd} restrictions are needed to ensure that, after an accurate detection of a single 1-bit of any code, none of the following i bits can be a 1-bit. So that, if one of these i bits are erroneously detected to be a 1-bit, it can be error corrected to a 0-bit. The 3^{rd} condition has to do with the adaptive-threshold detection technique as used in [12]; and why it is needed will be illustrated in Section 3.2.1. If the 3^{rd} condition were dropped, $\text{RLIM}_i(n)$ would be equal to run-length-limited codes of order (i, ∞) of length n with leading merging zeros [23]. Note that, when the order i of the RLIM scheme is set to 1, resultant codes correspond to the ISI-mitigating codes [12].

Let $C_i(n)$ denote the subset of all codes belonging to $\{0, 1\}^n$, with the 1^{st} condition in place, but the 2^{nd} and 3^{rd} conditions dropped. In coding literature, $C_i(n)$ is known as a d-limited sequence (where d substitutes i) [39]. Then, through basic

Combinatorics, and from [39], $C_i(n)$ conforms to the following relation (3.1), where each row represents a distinct code, and $|.|$ is the cardinality function. Note that 0_n^m denotes a matrix of 0s with m columns and n rows; and similarly 1_n^m denotes a matrix of $1s$ with m columns and n rows.

$$\mathbf{C}_i(n) = \left[\begin{array}{c:c} \mathbf{0}^{\mathbf{1}}_{|C_i(n-1)|} & \mathbf{C}_i(n-1) \\ \hdashline \mathbf{1}^{\mathbf{1}}_{|C_i(n-1-i)|}\,\mathbf{0}^{\boldsymbol{i}}_{|C_i(n-1-i)|} & \mathbf{C}_i(n-1-i) \end{array}\right] \tag{3.1}$$

This recursive relation holds, because the elements of $C_i(n)$ can be categorized into two groups: those that start with a 0-bit, and those that start with a 1-bit. If a code starts with a 0-bit, the remaining part of it can be any element of $C_i(n-1)$. Similarly, If a code starts with a 1-bit, this 1-bit must be followed by i 0-bits, then the remaining part can be any element of $C_i(n-1-i)$. To obtain the $\mathrm{RLIM}_i(n)$, first we need to enforce the 2^{nd} condition, which was dropped in the derivation of $C_i(n)$. That is, i 0-bits are concatenated to the start of each code of $C_i(n-i)$. Then, to enforce the 3^{rd} condition, we need to remove the code that consist entirely of 0-bits from the matrix. These operations can be done via the matrix given below. Note that the superscript (*) indicates the removal of the code (i.e., the row) consisting completely of 0-bits.

$$\mathbf{RLIM}_{\boldsymbol{i}}(\boldsymbol{n}) = \left[\begin{array}{c:c} \mathbf{0}^{\mathbf{i}}_{|\mathbf{C}^*_i(\mathbf{n-i})|} & \mathbf{C}^*_{\boldsymbol{i}}(\boldsymbol{n-i}) \end{array}\right] \tag{3.2}$$

To better illustrate how these matrices can be utilised, we will now obtain the code space of $\mathrm{RLIM}_2(6)$ using the matrices given above. We first need the base-cases. Let us give $C_2(1)$, $C_2(2)$, and $C_2(3)$.

$$C_2(1) = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ C_2(2) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix},\ C_2(3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.3}$$

Now we apply the recursion matrix given at (3.1) to obtain the $C_2(4)$ as follows.

$$C_2(4) = \left[\begin{array}{c:c} 0_4^1 & C_2(3) \\ \hdashline 1_2^1\ 0_2^2 & C_2(1) \end{array}\right] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

To obtain $\text{RLIM}_2(6)$ we need to apply the matrix given at (3.2) to the $C_2(4)$. This procedure is as follows.

$$\text{RLIM}_2(6) = \left[\begin{array}{c:c} 0_5^2 & C_2^*(4) \end{array}\right] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Any $\text{RLIM}_i(n)$ can be obtained using the recursive procedure defined in this section. For a given value of i, only the base cases, $C_i(1), \ldots, C_i(j), \ldots, C_i(i+1)$, where $1 \leq j \leq i+1$, should be pre-computed as in (3.3). This is a trivial task: Each base-case $C_i(j)$ has, in total, $j+1$ codes (i.e., rows). The first code of each $C_i(j)$ is the 0 vector, and the remaining j codes have the following property: The k^{th} code (i.e. row) of $C_i(j)$, where $2 \leq k \leq j+1$, has a 1-bit in the $(j+2-k)^{th}$ position, with remaining places all assigned with 0-bits.

Finally, to perform encoding and decoding, after deriving the code space $\text{RLIM}_i(n)$, a bijective function needs to be created between $\{0,1\}^{\lfloor \log_2 |RLIM_i(n)| \rfloor}$ and a subset S of $\text{RLIM}_i(n)$ with size $2^{\lfloor \log_2 |RLIM_i(n)| \rfloor}$, where $\lfloor . \rfloor$ is the floor function. The subset S should include codes of $\text{RLIM}_i(n)$ with the fewest number of 1-bits. This criterion

is important because it ensures that the coding scheme can utilize a greater number of information molecules for transmitting each 1-bit [40], thereby reducing the bit error rate.

Let us denote such a subset S of $\text{RLIM}_i(n)$, by $\text{RLIM}_i(n,k)$ provided that $|RLIM_i(n)| \geq 2^k$. That is, S a minimal subset with size 2^k of $\text{RLIM}_i(n)$ in term of the total number of 1-bits it contains. Since there can be multiple such minimal subsets, we choose the one that contains lexicographically (in binary order) the smallest codes. After creating the subset S, we order it by the binary values of its codes to facilitate the binary search algorithm during decoding. Specifically, when converting each code in S back to its associated element in $\{0,1\}^k$, the binary search enables retrieval in $\mathcal{O}(k)$ time. The Python implementations of codebook generation, encoding, and decoding algorithms, for any $\text{RLIM}_i(n,k)$, are given in Section 3.4.

Classical (d,k) RLL coding is well established [23]. Weight-aware variants also exist where the codebook is defined by fixing the Hamming weight or bounding the running digital sum, and then indexing those sets enumeratively [41, 42]. Since the effect of the 3^{rd} restriction is minuscule, what actually sets apart RLIM codes from conventional RLL codes is the minimisation of the total number of 1-bits in the code space. Our proposed RLIM forms a fixed-rate, minimum-weight subset drawn across multiple weight classes, rather than taking a single constant-weight (or bounded-weight) class. To the best of our knowledge, this fixed-size minimum-weight selection principle for (i,∞) constraints has not been explicitly formulated in prior work [23, 41, 43]. As we will demonstrate in the performance evaluation section, this selection will prove advantageous over the classical RLL codes, whose 2^k codewords are typically chosen lexicographically (as in our simulation comparisons), without any 1-bit minimisation.

3.2.1 Detection

Assume that an $\text{RLIM}_i(n)$ coding scheme is used. We will adopt the adaptive threshold detection technique. Let $m = (m_1, \ldots, m_n)$ be the sequence giving the absorbed number of molecules at each signal interval of a code $c \in \text{RLIM}_i(n)$. Define $m_{max} = max\{m_{i+1}, ..., m_n\}$[3.1] and $m_{min} = min\{m_{i+1}, ..., m_n\}$[3.2]. Then the adaptive threshold, τ^m_{adapt}, pertaining to m is found as

$$\tau^m_{adapt} = a \cdot m_{min} + (1 - a) \cdot m_{max}, \ \text{[12]} \tag{3.6}$$

where a is the channel-specific scaling constant [12]. Note that $0 \leq a \leq 1$, and thus $m_{min} \leq \tau^m_{adapt} \leq m_{max}$. If the number of absorbed molecules falls below τ^m_{adapt}, the corresponding signal interval is detected as a 0-bit, otherwise it is detected as a 1-bit. To find the optimal value of a, ensuing pilot signals, whose content is pre-known by the receiver, are sent [12]; and the value of a that results in the least BER value is chosen. The receiver can do this selection by increasing a from 0 to 1 with a pre-determined step size of s, whose value, in this chapter, is taken to be 0.005. Through this approach, an element of m that has the value m_{max} is always detected to be a 1-bit. This is why $\text{RLIM}_i(n)$ needs at least one 1-bit in each code.

Additionally, we provide the implementation of RLIM-specific static-threshold detection technique in Algorithm 3.1, as we will evaluate both the static and dynamic approaches in Section 3.3. Algorithm 3.1 strengthens the classical static threshold algorithm [6] by providing robustness against edge cases through its Lines 7–10 [3.3]. It ensures that each detected code contains at least one 1-bit, which holds true for all $\text{RLIM}_i(n)$ codes. Using pilot signals, the optimal static threshold for a given channel

[3.1] For ISI-mitigating codes, m_{max} is defined as the maximum of $m - \{m_1\}$ (i.e., the sequence m excluding its first element). By redefining $m_{max} = max\{m_{i+1}, ..., m_n\}$, we generalize this approach beyond ISI-mitigating codes.

[3.2] For ISI-mitigating codes, m_{min} is defined as the minimum of $m - \{m_1\}$. By redefining $m_{min} = min\{m_{i+1}, ..., m_n\}$, we generalize this approach beyond ISI-mitigating codes.

[3.3] This holds when the detected molecule-count sequence contains a non-zero value. In our implementation, whenever the detected integer sequence is the 0 vector, we return a code whose only 1-bit is located in its $(i + 1)$th index (where the indices start from 1).

Algorithm 3.1 Proposed Detection with Static Threshold

Require: Let m be the sequence of absorbed-molecule counts of length n, τ_{static} the static threshold, and i the order of $\text{RLIM}_i(n, k)$.

Let *detected_bit_sequence* $\leftarrow \mathbf{0}_{1\times n}$
for $j \leftarrow 1$ **to** n **do**
    if $m[j] \geq \tau_{\text{static}}$ **then**
        detected_bit_sequence$[j] \leftarrow 1$
    end if
end for
if *detected_bit_sequence*$[i+1:n]$ is the 0-vector **then**
    $l \leftarrow \arg \max_{h=i+1,\ldots,n} m[h]$
    detected_bit_sequence$[l] \leftarrow 1$
end if
return *detected_bit_sequence*

can be determined similarly to the optimal scaling constant derivation: By testing thresholds from 1 to M, the threshold that yields the lowest BER is identified.

3.2.2 Error Correction

After detection, a resultant binary sequence may not be an element of the coding scheme used. Thus, error correction is needed. By observing that in any code of $\text{RLIM}_i(n)$, the first i bits are 0 and no 1-bit can be followed by another 1-bit within the next i positions, we propose Algorithm 3.2 for error correction. Note that, when the order i is set to 1 in Algorithm 3.2 the resulting algorithm coincides with the one for ISI-mitigating codes presented in [12]. To the best of our knowledge, we are not aware of any prior $O(n)$ time decoder that, for any $y \in \{0,1\}^n$, returns a minimum-Hamming-distance sequence satisfying the general (i, ∞)–RLL constraint (with leading i zeros); Algorithm 3.2 provides such a procedure. After applying the error correction algorithm to a detected code, the resultant code c may not be an

Algorithm 3.2 Proposed Error Correction for RLIM

Require: *detected_code* with size n, order i of $\mathrm{RLIM}_i(n,k)$

```
j = 1; skip = i
while j ≤ n do
    if skip > 0 then
        detected_code[j] = 0
        skip = skip − 1
    else
        if detected_code[j] == 1 then
            skip = i
        end if
    end if
    j = j + 1
end while
```

element of $\mathrm{RLIM}_i(n,k)$, in which case we iteratively substitute the right-most 1-bit of the code c with a 0-bit, while searching it inside $\mathrm{RLIM}_i(n,k)$ at each substitution. We now state the following theorem, which establishes the optimality of our RLIM error-correction algorithm.

Theorem: In terms of Hamming distance (the number of differing bits between two binary words of equal length), for (i,∞)-RLL codes, Algorithm 3.2 is optimal. That is,

$$\forall\, y \in \{0,1\}^n : \qquad d_H\Big(F_i(y),\, y\Big) \;=\; \min\Big\{\, d_H(x,y) \;:\; x \in RLL_i(n)\Big\}, \tag{3.7}$$

where $F_i(y)$ denotes the output of Algorithm 3.2, d_H is the Hamming distance, and $RLL_i(n)$ is the set of all RLL codes of length n and order i with leading 0-bits.

Proof: Let $y \in \{0,1\}^n$. If $x \in RLL_i(n)$ and for some t we have $y_t = 0$ and $x_t = 1$, define x' by setting $x'_t = 0$ and $x'_u = x_u$ for $u \neq t$. Then $x' \in RLL_i(n)$ (replacing a 1 by 0 cannot violate the RLL constraint) and $d_H(x',y) = d_H(x,y) - 1$. Therefore if $x^* \in RLL_i(n)$ has the smallest $d_H(x^*,y)$ (i.e, x^* is a minimizer code), then the

indices of 1-bits of x^* (denoted as $ones(x^*)$) are a subset of $\{\, t \in \{i+1, \ldots, n\} : y_t = 1 \,\}$. We map each 1-bit of y at index $t \geq i{+}1$ to the half-open interval $[t,\, t+i+1)$. We are then trying to choose the maximum number of 1-bits with non-overlapping intervals to minimize the Hamming distance. Selecting 1-positions that satisfy the (i, ∞) constraint is exactly selecting a maximum-size set of non-overlapping intervals, for which the earliest-feasible greedy rule is optimal ([44], Chapter 16.1). Algorithm 3.2 implements this rule by scanning left-to-right, keeping the earliest feasible 1 and skipping the next i positions; hence it maximizes $|ones(x)|$, thus minimizing $d_H(x, y)$. □

Note that, $RLIM_i(n) \subset RLL_i(n)$. Since our detection methods guarantee the existence of at least one 1-bit in positions $i+1$ through n of a detected binary sequence c of length n, $F_i(c)$ is a $\mathrm{RLIM}_i(n)$ code by the construction of the Algorithm 3.2. Therefore, the theorem above also applies for the RLIM codebooks, resulting in the following corollary:

Corollary: In terms of Hamming distance, for $RLIM_i(n)$ codes, Algorithm 3.2 is optimal. That is,

$$\forall\, y \in \{0,1\}^n,\ \Big[\exists\, t \in \{i{+}1, \ldots, n\} :\ y_t = 1\Big] \implies d_H\Big(F_i(y),\, y\Big) = \min_{x \in RLIM_i(n)} d_H(x, y). \tag{3.8}$$

3.2.3 Equivalence with Viterbi Decoding and Complexity

We now present the generic, standard trellis-based error-correction procedure for (i, ∞)–RLL codes, known as Viterbi hard-decoding [45, 46], which selects the codeword of minimum Hamming distance to the detected word. Let $y \in \{0,1\}^n$ be the detected word and let $x \in \{0,1\}^n$ be a candidate codeword. We decode on the standard (i, ∞)–RLL constraint as a labeled digraph with vertices $\{0, \ldots, i\}$ [23]: from each state (i.e., vertex) s, there is exactly one outgoing edge labeled 0-bit to $\min\{s+1, i\}$, and only from state i there is an outgoing edge labeled 1-bit to the state 0. A length-n path from start state $s_0 = 0$ (implementing the leading merging 0-bits, i.e., the first i outputs are 0-bits) yields x by reading edge labels; termination

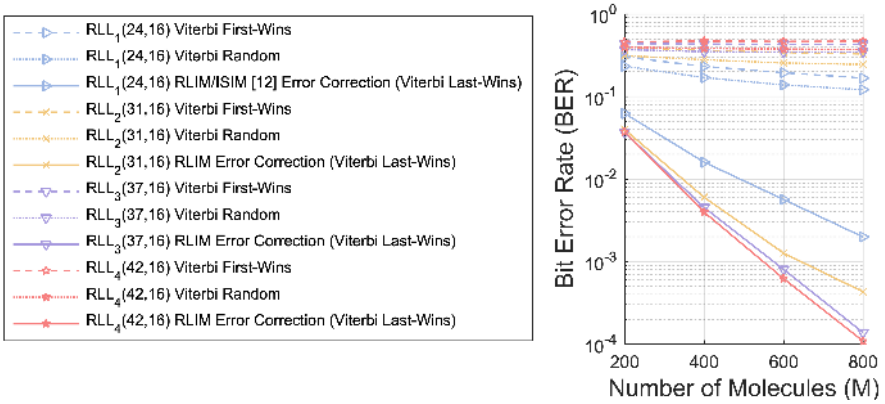


Figure 3.1: Error Correction Comparisons for RLL Codes with $t_s = 200\ ms$, $\sigma_n^2 = 0$, and $r_0 = 10$ µm

can happen at any state.

With Hamming branch cost $\mathbf{1}[x_t \neq y_t]$, the add–compare–select (Viterbi) recursion returns a minimum-Hamming-distance codeword x to y. Standard Viterbi expositions leave equal-metric ties unspecified ('break ties arbitrarily') [45], and real implementations resolve equality with a fixed comparator policy. In hardware/software, this typically amounts to using a strict comparator (always pick the first candidate when equal), a non-strict comparator (pick the later candidate when equal), or a randomized tie-break (select uniformly among equal-cost candidates at each state/time and among equal-cost terminal states) [47].

With the deterministic Viterbi last-wins policy (i.e., in case of ties at state i, prefer the self-loop $i \xrightarrow{0} i$ over $(i{-}1) \xrightarrow{0} i$, and at termination choose the tied state with the largest index), RLIM Algorithm 3.2 and Viterbi yield identical outputs; hence they are equivalent. RLIM runs in $O(n)$ time with $O(1)$ auxiliary space. Viterbi runs in $O((i{+}1)n)$ time with $O((i{+}1)n)$ space using full back-pointers (or $O(i{+}1)$ auxiliary space with checkpointed or divide-and-conquer traceback). Consequently, from a computational standpoint, RLIM is a better choice than its Viterbi-equivalent.

We therefore compare three decoders for (i, ∞)-RLL: generic Viterbi (first-wins, random tie-breaking) and our RLIM error correction (equivalent to Viterbi last-

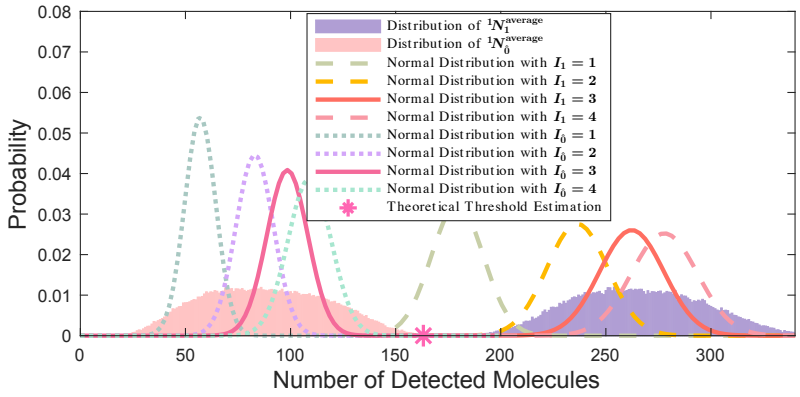
wins). Simulation parameters, other than those listed in the caption of Figure 3.1, are listed in Tables 3.1–3.3, and Section 3.3 details the simulation rationale and procedure. We use static-threshold detection, which yields the best BER under the non-drift channel settings of Section 3.3. For each comparison, we first transmit 46080 bits by encoding six independent 7680-bit sequences to determine the optimal threshold. Using those thresholds, we then send 967680 bits (six contiguous runs of 161280-bit encodings) to estimate BER. As shown in Fig. 3.1, Viterbi with generic tie-breaking (first-wins or random) yields higher BER, whereas RLIM (i.e., Viterbi last-wins) achieves the lowest BER. Accordingly, for the remainder of this work we adopt the RLIM error-correction algorithm when benchmarking classical RLL and for RLIM codes themselves. There is a thesis [48], where RLL codes(including order $(1, \infty)$) have been applied to MC. However, the author obtained numerical results in which RLL codes were surpassed by uncoded transmissions in terms of BER. With Algorithm 3.2 implemented to RLL codes, in the performance evaluation section, we will show that RLL codes gain a significant advantage over uncoded transmissions.

3.2.4 Analytical Estimation of Static Threshold

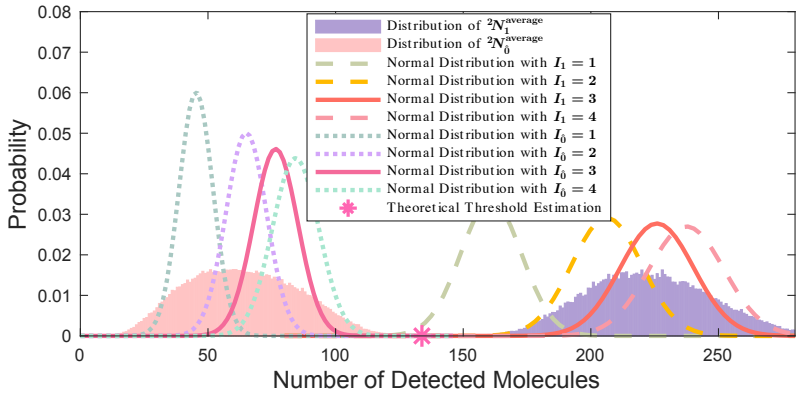
Both the adaptive and static threshold techniques described in the previous section requires sending pilot signals at the start of the communication. However, if the receiver has the knowledge of the channel parameters, transmitting pilot signals beforehand may no longer be necessary.

Our contribution in this subsection is adapting the analytical derivation method for ISI-mitigating codes [12] to $\text{RLIM}_i(n)$, with significant modifications which will be clarified at the end of this subsection. Define $ISI_j = \mathcal{N}(M \cdot p_j, M \cdot p_j \cdot (1 - p_j))$ as per equation (1.5), which approximates the distribution of the expected number of molecules detected from the emission of M molecules at the $(j-1)^{th}$ previous signal slot. Assuming $\text{RLIM}_i(n)$ is used, when a 1-bit is transmitted in the current signal slot, the distribution of the maximum expected number of detected molecules can be given by the equation (3.9).

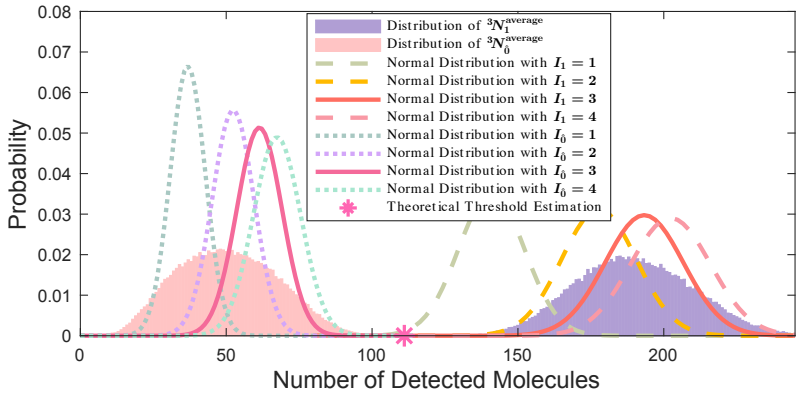
$$ {}^{i}N_1^{max} = \lim_{I_1 \to \infty} \left(\sum_{k=1}^{I_1} ISI_{1+(i+1)\cdot(k-1)} \right) + \mathcal{N}(0, \sigma_n^2) \tag{3.9} $$



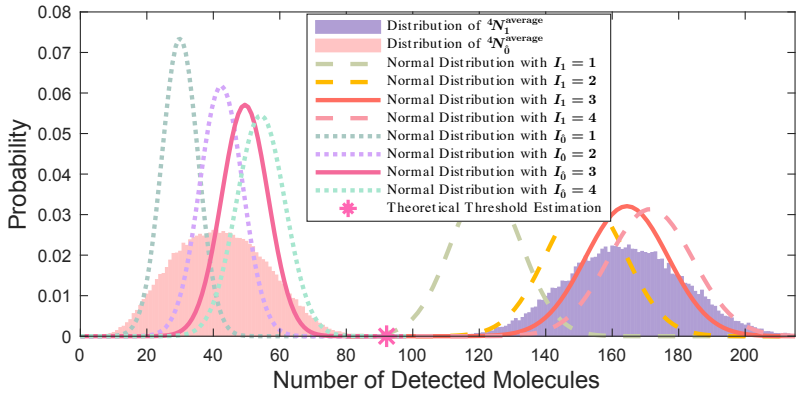
(a) $\text{RLIM}_1(24, 16)$ [12] with $M = 1294$ and $t_s = ((16/24) \cdot 200)$ ms



(b) $\text{RLIM}_2(31, 16)$ with $M = 1484$ and $t_s = ((16/31) \cdot 200)$ ms



(c) $\text{RLIM}_3(37, 16)$ with $M = 1590$ and $t_s = ((16/37) \cdot 200)$ ms



(d) $\text{RLIM}_4(42, 16)$ with $M = 1621$ and $t_s = ((16/42) \cdot 200)$ ms

Figure 3.2: Detected Molecule Distributions for coding schemes $\text{RLIM}_i(n_i, 16)$ in an MC channel with, $D = 79.4$ µm^2/s, $r_R = 5$ µm, $r_0 = 10$ µm, $I = 200$, $\sigma_n^2 = 0$, and unnormalized signal interval and molecule counts of $t_s = 200$ ms, and $M = 1000$.

Rather than deriving a distribution for the maximum possible number of detected molecules, we aim to derive a distribution for the average case. To approximate mean value of a such a distribution, I_1 should be a positive integer. To illustrate this phenomenon, consider the detection count distribution from a binomial simulation of 10000 consecutive codewords, each encoding a random bit sequence of length 16. The simulation uses RLIM codes of orders from 1 to 4 in an MC channel whose simulation parameters are provided in the caption of Fig 3.2. with corresponding normalized parameters given in the sub-captions. The simulator selection and normalization procedure will be explained in the next subsection. As can be seen in the right sections of each subfigure in Fig. 3.2, $I_1 = 3$ provides a more accurate representation of the molecule count distribution of 1-bits (shown in purple) compared to other values of I_1, based on its similarity to the mean value of the distribution. Please note that we observed a similar phenomena for molecule count (M) values smaller than 1000. Accordingly, we assume the following approximation in (3.10):

$$ {}^{i}N_{1}^{average} \approx \left(\sum_{k=1}^{3} ISI_{1+(i+1)\cdot(k-1)} \right) + \mathcal{N}(0, \sigma_n^2) \tag{3.10} $$

Now, consider the case where the current signal slot corresponds to a 0-bit, and all preceding signal slots have been detected accurately. For any code of $\mathrm{RLIM}_i(n)$, this implies that if there are any 1-bits within i previous signal slots or if the current interval corresponds to one of the first i positions of a code, then regardless of the threshold value, the error correction algorithm will always correctly detect the current signal slot as a 0-bit. Accordingly, we denote a 0-bit, which is neither in one of the first i positions of a code nor has any preceding 1-bits within i slots, as a $\hat{0}$-bit. For the transmission of a $\hat{0}$-bit, the maximum expected number of detected molecules is given by the following distribution:

$$ {}^{i}N_{\hat{0}}^{max} = \lim_{I_{\hat{0}} \to \infty} \left(\sum_{k=1}^{I_{\hat{0}}} ISI_{1+(i+1)\cdot(k)} \right) + \mathcal{N}(0, \sigma_n^2) \tag{3.11} $$

Setting $I_{\hat{0}} = 2$ provides a better approximation for the mean of ${}^{i}N_{\hat{0}}^{\text{average}}$ (as shown in the left sections of the subfigures in Fig. 3.2), but, like other values of $I_{\hat{0}}$, it fails to accurately model the variance. Moreover, $I_{\hat{0}} = 2$ is insufficient for predicting the distribution of ${}^{i}N_{\hat{0}}^{\text{average}}$ at higher molecule counts. We chose $I_{\hat{0}} = 3$, as given in Eq. (3.12), as it more accurately approximates the distribution at larger detected molecule counts.

$$ {}^{i}N_{\hat{0}}^{average} \approx \left(\sum_{k=1}^{3} ISI_{1+(i+1)\cdot(k)} \right) + \mathcal{N}(0, \sigma_n^2) \tag{3.12} $$

Future research should focus on developing more accurate theoretical distributions for ${}^{i}N_{1}^{average}$ and ${}^{i}N_{\hat{0}}^{average}$. While the complex and recursive nature of $\mathrm{RLIM}_i(n, k)$ makes theoretically obtaining these distributions a challenging task, achieving more precise distributions will lead to BER-wise improved static threshold values. This, when the channel parameters are known, will further discard the need to send pilot signals in advance of the communication.

Let P_1 denote the appearance probability of 1-bits in the code space $\mathrm{RLIM}_i(n,k)$, and similarly let $P_{\hat{0}}$ denote the appearance probability of $\hat{0}$-bits inside the code space $\mathrm{RLIM}_i(n,k)$. Then, using the Formulae (3.10) and (3.12), the probability of correctly detecting the current signal slot, assuming all preceding slots have been detected accurately, is given as

$$\begin{aligned} P = & P_{\hat{0}} \cdot \left[1 - Q\left(\frac{\tau_{static} - (\sum_{k=1}^{3} M \cdot (p_{(1+(i+1)\cdot k)}))}{\sqrt{(\sum_{k=1}^{3} M \cdot (p_{(1+(i+1)\cdot k)}) \cdot (1 - p_{(1+(i+1)\cdot k)})) + \sigma_n^2}} \right) \right] \\ & + P_1 \cdot Q\left(\frac{\tau_{static} - (\sum_{k=1}^{3} M \cdot p_{(1+(i+1)\cdot (k-1))})}{\sqrt{(\sum_{k=1}^{3} M \cdot (p_{(1+(i+1)\cdot (k-1))}) \cdot (1 - p_{(1+(i+1)\cdot (k-1))})) + \sigma_n^2}} \right), \end{aligned} \tag{3.13}$$

where Q is the tail probability function of the Gaussian normal distribution (i.e., $Q\left(\frac{x-a}{\sqrt{b}}\right)$ gives the integration of $\mathcal{N}(a,b)$ from x to ∞), and τ_{static} is the static threshold constant. We need to find the highest value of (3.13) in terms of τ_{static}. For simplicity, we make the following substitutions:

$$\begin{aligned} A &= \left(\textstyle\sum_{k=1}^{3} M \cdot (p_{1+(i+1)\cdot k}) \right) \\ B &= \left(\textstyle\sum_{k=1}^{3} M \cdot (p_{1+(i+1)\cdot k}) \cdot (1 - p_{1+(i+1)\cdot k}) \right) + \sigma_n^2 \\ C &= \left(\textstyle\sum_{k=1}^{3} M \cdot p_{1+(i+1)\cdot (k-1)} \right) \\ D &= \left(\textstyle\sum_{k=1}^{3} M \cdot (p_{1+(i+1)\cdot (k-1)}) \cdot (1 - p_{1+(i+1)\cdot (k-1)}) \right) + \sigma_n^2 \end{aligned} \tag{3.14}$$

Then, taking the derivative of (3.13) with respect to τ_{static}, and equating it to zero, the analytical derivation of τ_{static} is obtained as follows:

$$\tau_{static} = \frac{D \cdot A - B \cdot C + \sqrt{B \cdot D \cdot \left((C-A)^2 - 2 \cdot (B-D) \cdot \log_e \left(\frac{\sqrt{D} \cdot P_{\hat{0}}}{\sqrt{B} \cdot P_1} \right) \right)}}{D - B} \tag{3.15}$$

As an example, for $\mathrm{RLIM}_4(42,16)$, the values of $\mathrm{P}_{\hat{0}}$ and P_1 are $\frac{996497}{42 \cdot 2^{16}}$ and $\frac{323397}{42 \cdot 2^{16}}$, respectively. Then, we calculate the corresponding $\{p_x\}$ values from equation (1.3) based on the normalized channel parameters outlined in the sub-caption of Fig. 3.2(d) and the remaining parameters that are given in the main caption of Fig. 3.2.

Substituting all these values into equation (3.15) yields a static threshold estimate of ≈92.13, as marked in Fig. 3.2(d).

In [12], the dynamic threshold formula $(a \cdot m_{min} + (1-a) \cdot m_{max})$ is used in place of τ_{static} within equation (3.15). The values of I_1 and $I_{\hat{0}}$ are both set to 1. The derivative of equation (3.15) is then taken with respect to a, set equal to zero, and solved for a. As a result, in [12], a new analytical value for a is determined for each detected codeword molecule count sequence, m, based on m_{max} and m_{min}. The dynamic threshold, $\tau^m_{dynamic}$, is updated accordingly, as per Eq. (3.6).

However, in such an analytical approach, regardless of the values m_{max} and m_{min} take, $\tau^m_{dynamic}$ always equals to τ_{static}. Therefore, while our approach in this paper for $\text{RLIM}_1(n)$ is equivalent to that in [12] for ISI-mitigating codes when $I_{\hat{0}} = 1$ and $I_1 = 1$, it is computationally more efficient: Our method requires computation only once and can be applied across all codeword detections. Moreover, as demonstrated in Fig. 3.2, the values $I_1 = 3$ and $I_{\hat{0}} = 3$ generally provide a better estimate than $I_1 = 1$ and $I_{\hat{0}} = 1$. This is because, if $I_1 = 1$ and $I_{\hat{0}} = 1$ were used, the threshold value would decrease in all four subfigures of Fig. 3.2; it is evident that this would result in a poorer estimate, as the threshold would intersect with the distribution of $\hat{0}$-bits, leading to a greater number of detection errors.

3.3 Performance Evaluation

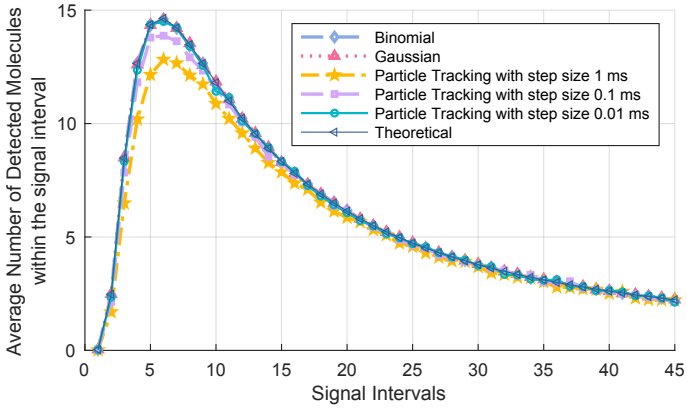
There are 3 main approaches to simulate an MC channel. The first is a discrete particle-tracking-based simulator. This method operates in discrete time steps (Δt), updating the 3D positions of all information molecules by independently drawing each coordinate from the normal distribution given in equation (1.1) It then counts how many molecules are absorbed by the receiver during each time step and removes the absorbed molecules from the simulation. The second approach is to use the summation of binomial distributions expressed in equation (1.4) to compute the absorbed (i.e., detected) molecule count within a signal interval. The third and

computationally fastest approach utilizes the normal distribution formula given in equation (1.5). Although this method is efficient, it may introduce slight errors for lower molecule counts and can occasionally produce negative detection counts.

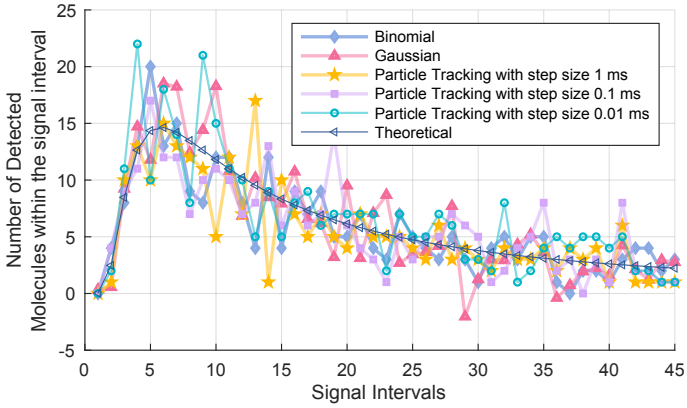
To highlight the distinctions between these three approaches, we calculated the averaged number of detected molecules during each signal interval over 10^4 samples, as illustrated in Fig. 3.3(a). These simulations are based on an initial release of 1000 molecules at the start of the first signal interval. The corresponding channel parameters are provided in the caption of Fig. 3.3. Additionally, Fig. 3.3(b) presents a graph of individual simulation runs to provide a clearer conceptual illustration. As shown in Fig. 3.3(a), smaller time steps yield more accurate results in particle-tracking simulations. However, reducing the time step below 1 ms results in excessively longer simulation run-times. For simulating non-drift MC channel conditions, we employ the binomial distribution approach for its accuracy and computational efficiency.

3.3.1 Normalizations and Simulation Parameters

To fairly compare different coding strategies, it is essential to normalize the signal interval and the molecule count per transmission of a 1-bit values. This ensures that the same amount of information is transmitted across different coding schemes within equal time periods and using an identical number of information molecules. The normalization, as given in Chapter 2 and [26], is done as follows: Let I be the set of all available information. Assume each element of I is equally likely to be transmitted. Suppose that a coding scheme C_1 encodes all the information of I, using S_1 bits, and M_1 1-bits. Also assume that a coding scheme C_2 encodes all the information of I, using S_2 bits, and M_2 1-bits. The signal interval value for coding scheme C_2 should be S_1/S_2 times that of the coding scheme C_1. Likewise, the number of molecules transmitted per 1-bit in coding scheme C_2 should be M_1/M_2 times that in coding scheme C_1. For our simulation, let I be the set $\{0,1\}^{16}$. To



(a) Averaged Detection Comparisons



(b) Exemplary Single-Run Detection Graphs

Figure 3.3: Simulator Comparisons for Channel Parameters of $M = 1000$, $D = 79.4$ µm^2/s, $r_R = 5$ µm, $r_0 = 10$ µm, and $t_s = 10$ ms.

encode I using $\text{RLIM}_i(n)$ we derive the following inequalities:

$$|\text{RLIM}_1(23)| < 2^{16} < |\text{RLIM}_1(24)| = 75024 < 2^{17} \tag{3.16}$$

$$|\text{RLIM}_2(30)| < 2^{16} < |\text{RLIM}_2(31)| = 85625 < 2^{17} \tag{3.17}$$

$$|\text{RLIM}_3(36)| < 2^{16} < |\text{RLIM}_3(37)| = 82628 < 2^{17} \tag{3.18}$$

$$|\text{RLIM}_4(41)| < 2^{16} < |\text{RLIM}_4(42)| = 67984 < 2^{17} \tag{3.19}$$

Table 3.1: Normalized Signal Interval Values

Coding Method	**Signal Interval of**	
	200 ms	**250 ms**
Uncoded [16→16]	200 ms	250 ms
$\text{RLIM}_1(24, 16)$ [16→24] [12]	$(16/24) \cdot 200$ ms	$(16/24) \cdot 250$ ms
$\text{RLIM}_2(31, 16)$ [16→31]	$(16/31) \cdot 200$ ms	$(16/31) \cdot 250$ ms
$\text{RLIM}_3(37, 16)$ [16→37]	$(16/37) \cdot 200$ ms	$(16/37) \cdot 250$ ms
$\text{RLIM}_4(42, 16)$ [16→42]	$(16/42) \cdot 200$ ms	$(16/42) \cdot 250$ ms
Hamming(4,7) [16→28] [37]	$(16/28) \cdot 200$ ms	$(16/28) \cdot 250$ ms
ISI-Free(4,2,1) [16→32] [36]	$(16/32) \cdot 200$ ms	$(16/32) \cdot 250$ ms

Table 3.2: Number of 1-bits and Normalized Molecule Counts

Coding Method	**Number of 1-bits**	**Molecule Count ($M \in \mathbb{N}^+$)**
Uncoded [16→16]	$16 \cdot 2^{15} = 524288$	$1 \cdot M$
$\text{RLIM}_1(24, 16)$ [16→24] [12]	405251	$\lfloor 1.2937 \cdot M \rceil$
$\text{RLIM}_2(31, 16)$ [16→31]	353228	$\lfloor 1.4842 \cdot M \rceil$
$\text{RLIM}_3(37, 16)$ [16→37]	329724	$\lfloor 1.5900 \cdot M \rceil$
$\text{RLIM}_4(42, 16)$ [16→42]	323397	$\lfloor 1.6211 \cdot M \rceil$
$\text{RLL}_1(24, 16)$ [16→24]	416350	$\lfloor 1.2592 \cdot M \rceil$
$\text{RLL}_2(31, 16)$ [16→31]	370310	$\lfloor 1.4158 \cdot M \rceil$
$\text{RLL}_3(37, 16)$ [16→37]	343276	$\lfloor 1.5273 \cdot M \rceil$
$\text{RLL}_4(42, 16)$ [16→42]	325735	$\lfloor 1.6095 \cdot M \rceil$
Hamming(4,7) [16→28] [37]	917504	$\lfloor 0.5714 \cdot M \rceil$
ISI-Free(4,2,1) [16→32] [36]	1048576	$\lfloor 0.5 \cdot M \rceil$

Thus, each of the sets $\text{RLIM}_1(24, 16)$, $\text{RLIM}_2(31, 16)$, $\text{RLIM}_3(37, 16)$, and RLIM_4 $(42, 16)$ is a valid codebook for encoding the information set $\{0, 1\}^{16}$. As explained

Table 3.3: Simulation Parameters

Parameter	**Value**
Diffusion coefficient (D)	79.4 µm^2/s
Distance between T_x and R_x (r_0)	$9.5 - 11.5$ µm
Receiver radius (r_R)	5 µm
Molecule count per a 1-bit for Uncoded (M)	$100 - 1000$
Signal interval for Uncoded (t_s)	$200 - 250$ ms
Receiver Gaussian counting noise variance (σ_n^2)	$0 - 20$
Channel Memory (I)	200

previously, these codebooks are created in a way that minimizes the total number of 1-bits contained inside them. The normalized signal interval periods are given in Table 3.1, for two different values: 200 ms and 250 ms. Table 3.2 presents the number of 1-bits in RLIM codebooks, along with the molecule count per a transmission of 1-bit values, normalized according to the Uncoded case, where $\lfloor . \rceil$ rounds the given number to the nearest integer. The full range of the parameters used in our comparative MC simulations are listed in Table 3.3 [3.4]. Please note that the RLL codes of order i have the same order and normalized signal interval values with the corresponding RLIM codes of order i.

To determine the optimal static threshold and scaling constant[3.5] values for each

[3.4]The values of the parameters D, r_R, and r_0 (with $r_0 = 10$ µm) in Table 3.3 are standard in MC literature. They model an MC channel where human insulin hormone is utilised as information molecules [4].

[3.5] Dynamic detection algorithm assumes at least one 1-bit per code, which is the case for all RLIM$_i(n)$ but not necessarily for Uncoded, ISI-free, Hamming, and RLL schemes. To address this issue for these coding schemes, we used a modified dynamic detection algorithm from Chapter 2, which introduces additional channel-specific parameters *min* and *spacing* alongside a. Through pilot signal transmissions for these schemes, we determined *min*, optimal a, and optimal *spacing* values. For RLL codes, we adopted the same static spacing values (i.e. the n values) as those used for the RLIM codes.

coding scheme across different channel parameters, we transmitted a total of 53760 bits by encoding 7 different randomly chosen 7680-bits sequences. Each of the 7 runs was independent. For instance, $\text{RLIM}_1(24, 16)$ encodes 7680 bits using 11520 bits, where all of the bits are transmitted in a contiguous manner. In MC simulations, imitating ISI, which stem from the accumulation of molecules over time, is important. Thus, uninterruptedly simulating long contiguous codewords with a high value of channel capacity (which we took as 200) provides a realistic representation of the MC channel, as we do here.

After having determined the optimal values of the static threshold and scaling constants (along with *min* and optimal *spacing* values for Uncoded, ISI-free, and Hamming schemes [3.5]), we sent a total of 2257920 bits (7 contiguous runs of the encodings of randomly chosen bit sequences of length 322560) [3.6] to calculate the respective BER values. Please note that these specific number of bits have been chosen to ensure that all different block lengths of 4, 8, 12, and 16 perfectly divide these numbers, ensuring a fair comparison among coding methods in Fig. 3.4. Note that in Figs. 3.4-3.6, $\text{RLIM}_1(n, k)$ is referred to as ISI-mitigating codes [12] of length n with block length k (i.e., ISIM (n, k)), as they are equivalent [3.7].

3.3.2 Evaluation of Simulation Results

Impact of Block Length on Performance

The simulation results for different block lengths k (i.e., different information sets $\{0, 1\}^k$) are provided in Fig. 3.4(a) and (b). The corresponding values of $\{x_i^k\}$, whose derivations follow a similar process to those given in formulae 3.16-3.19, are

[3.6] When t_s=250 ms and r_0=9.5 µm, we multiplied the number of test bits by 2 to obtain more accurate BER values.

[3.7] The ISIM code construction in [12] does not enforce the 1-bit minimization criterion or provide the full details of how subset selection is done. Consequently, we assume ISIM [12] selects codewords by minimizing the number of 1-bits, which makes them equal to our RLIM codes of order 1.

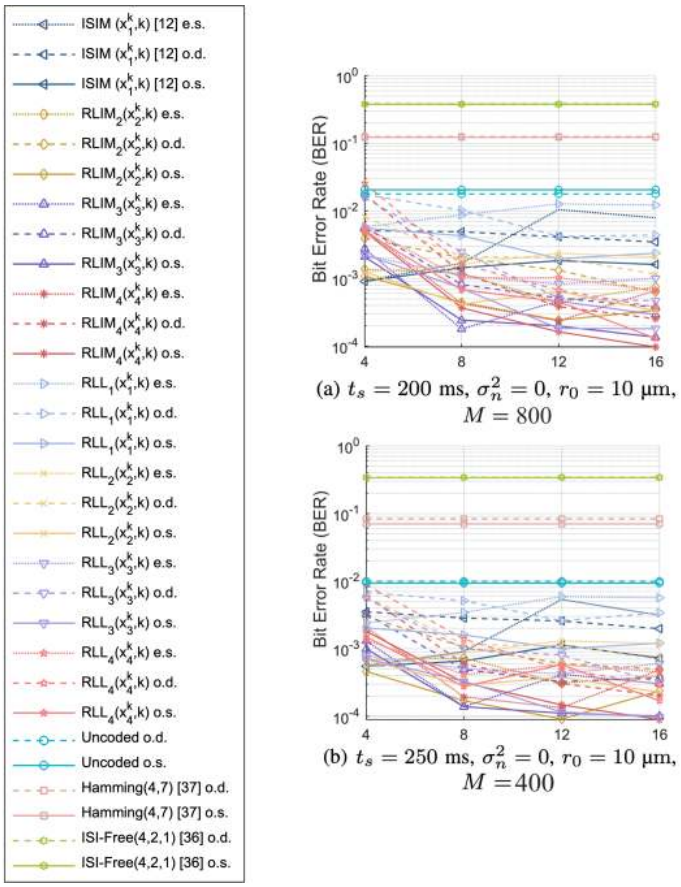


Figure 3.4: MC Simulation Results for Different Block Lengths

shown in Table 3.4. Note that, in Figs. 3.4, 3.5, and 3.6, o.d., e.s., and o.s., stand for optimal dynamic, estimated static, and optimal static respectively. These figures show that, as k increases, the best attainable BER, defined as the minimum across

Table 3.4: Values of $\{x_i^k\}$ in Fig. 3.3(a) and (b)

$\boldsymbol{k}$	$\boldsymbol{x_1^k}$	$\boldsymbol{x_2^k}$	$\boldsymbol{x_3^k}$	$\boldsymbol{x_4^k}$
4	7	9	11	13
8	13	16	20	23
12	18	24	28	33
16	24	31	37	42

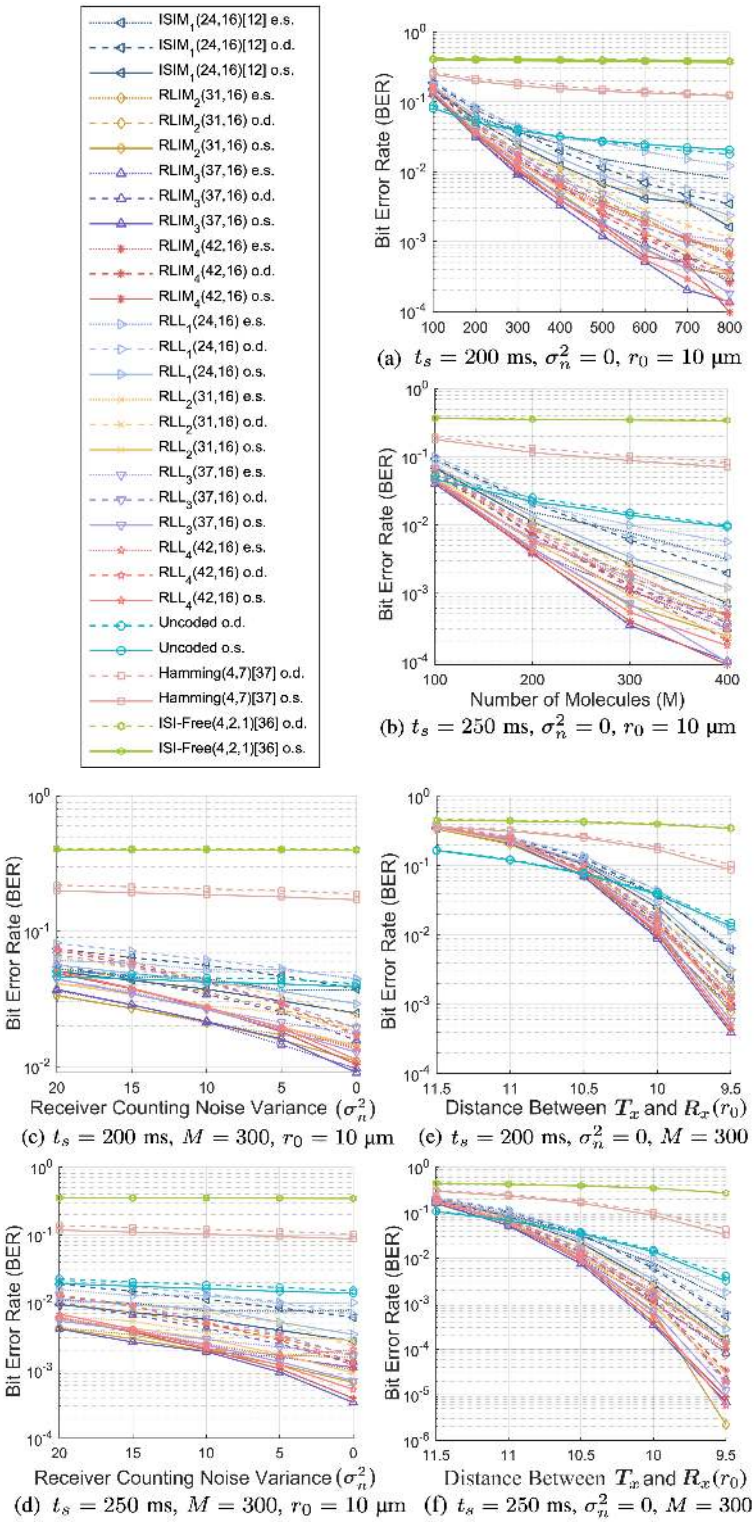


Figure 3.5: MC Simulation Results for Different Parameters

all compared schemes at each k, decreases; for example, the best BER at $k = 8$ is lower than at $k = 4$. Notably, in both Figs. 3.4(a) and (b), the best overall performance among all coding schemes and block lengths is achieved when a block length of 16 is used. This is why we chose $k = 16$ for the remainder of comparisons. Note that the normalizations of molecule count and signal interval values for block lengths 4, 8, and 12 (not shown for brevity) follow the same process as for block length 16, as detailed in Tables 3.1 and 3.2.

Comparison Between RLL and RLIM

As shown in Section 3.2.3, Algorithm 3.2 (RLIM Error Correction) is equivalent to Viterbi with a last-wins tie-break and, on our MC channels, achieves substantially lower BER than hard-decision Viterbi with other common tie-breaks (e.g., first-wins and random). We now compare RLIM–corrected RLL directly against our RLIM codes, which additionally minimise the total number of 1–bits in the codebook (see Table 3.2); this weight minimisation increases the number of molecules available per transmitted 1-bit under the normalisation in Section 3.3, and therefore is expected to further reduce BER.

We perform an elementwise comparison across all distinct simulation parameters. For each order $i \in \{1, 2, 3, 4\}$ and each distinct data point, we mark a "win" for the

Table 3.5: Elementwise BER comparison of RLIM_i vs. RLL_i.

i	RLIM_i wins		RLL_i wins		ties
	count	mean fold (RLL/RLIM)	count	mean fold (RLIM/RLL)	
1	101	1.518×	1	1.143×	0
2	97	1.669×	5	1.100×	0
3	94	1.542×	7	1.260×	1
4	78	1.321×	24	1.167×	0

method with the lower BER and record the BER fold ratio as the winner's margin. The summary in Table 3.5 reports, for each i, how many positions each method wins and the mean fold among those winning positions. Across all orders, RLIM wins 370 of 408 comparisons with a weighted mean improvement of $\approx$1.522$\times$ at the winning points, whereas the rarer RLL wins average $\approx$1.175$\times$. The only order with a noticeable number of RLL wins is i=4 (24/102). This aligns with Table 3.2: at i=4, RLIM and RLL have the closest 1-bit counts, yielding smaller normalisation gaps and, consequently, tighter BER margins. Overall, RLIM maintains a clear advantage over RLL.

Effects of Different Parameters

As can be inferred from Fig. 3.5, higher molecule count and signal interval values reduce BER, while higher r_0 and receiver noise variance σ_n^2 [3.8] levels degrade it. As the noise variance increases, the advantage of RLL and RLIM codes over uncoded transmissions decreases. Increasing the coding order i initially improves BER, but we conjecture that benefits diminish after a certain value of i, especially with shorter coding block lengths, as can be seen in Fig. 3.5(a) and (b). Additionally, at greater distances between T_x and R_x (r_0), the performance of all coding methods (including Uncoded) degrades significantly, becoming unreliably worse. In such situations, the Uncoded method, however, surpasses all others, as shown in Fig. 3.5(e) and (f).

Storage Requirements and Trade-offs

While increasing the block length k can reduce BER, it exponentially raises memory requirements for RLIM codes, as each $\mathrm{RLIM}_i(n,k)$ requires $n \cdot 2^k$ bits of storage space. These trade-offs must be carefully considered for practical applications of the proposed coding scheme. For $k = 8, 12, 16$, the proposed RLIM codes of order $i \geq 2$ still outperform others (in their respective detection technique categories), as

[3.8] Because the receiver's counter can register only integer values, we have rounded each sample of Gaussian counting noise to the nearest integer in our simulations.

demonstrated in Fig. 3.4(a) and (b). Accordingly, if data storage capacity is limited, opting for $k = 8$ or $k = 12$ instead of $k = 16$ is advisable. For future comparative simulations of our proposed family of codes, we recommend using a coding block length of at least 12 for a fair comparison, with $k = 16$ being a more preferable option.

Constraint-Level Coincidence with (i, ∞)–RLL in Drift-Free Channels

For a non-drift channel, a key observation is that for all RLIM, the optimal static threshold detection method consistently outperforms the optimal dynamic threshold method. Consequently, dynamic threshold detection would only be necessary when the parameters of MC channel fluctuates over time. Henceforth, it is no longer necessary to assume the presence of a single 1-bit in a code. Thus, for future uses with static threshold through a non-drift static channel, we make the enhancement in Eq. (3.20) for RLIM codes, so the constraint set coincides with the classical run-length-limited (RLL) codes of order (i, ∞) of length n [23] with leading merging 0-bits. Please note that implementing this enhancement would eliminate the need for Lines 7–10 in Algorithm 3.1. The critical minimization of 1-bits property of practical $\text{RLIM}_i(n, k)$ codes still preserves their distinct character and makes them advantageous over RLIM-corrected classical RLL codes as shown in the simulation results.

$$\widehat{\text{RLIM}}_i(n) \;=\; \left[\mathbf{0}^{i}_{|C_i(n-i)|} \;\vdots\; C_i(n-i)\right] \;=\; (i,\infty) - \text{RLL codes of length n} \quad (3.20)$$

3.3.3 Time-Varying Drift Channel Model and Gains from Dynamic Threshold

In many real-world MC scenarios, the channel may exhibit random or time-varying drift, causing the arrival profiles of information molecules to fluctuate from one symbol interval to the next. Under such non-stationary conditions, a well-tuned dynamic threshold scheme can significantly outperform an optimal static threshold,

Table 3.6: Drift and Particle-Tracking Parameters for Time-Varying MC Simulations

Parameter	**Value**
Time step (Δt)	1 ms
Drift correlation timescale (τ_{drift})	10 s
Drift standard deviation (σ_v)	10 µm/s
Mean drift velocity ($\mathbf{v}_{\text{mean}}$)	[1, 0, 0] µm/s
Max molecule age	5 s
Emitter (T_x) location	[0, 0, 0] µm
Receiver (R_x) center	[r_0, 0, 0] µm

since the latter fails to adapt to shifting channel conditions. To illustrate this advantage, we adopt a particle-tracking model, based on the simulator [27], that extends the discrete-time update formula given in Eq. 1.1.

At each simulation time step of duration Δt, the drift velocity $\mathbf{v}(t)$ is updated according to the Euler–Maruyama discrete-time approximation of an Ornstein–Uhlenbeck (OU) process with a nonzero mean:

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{1}{\tau_{\text{drift}}}\Big(\mathbf{v}_{\text{mean}} - \mathbf{v}(t)\Big)\Delta t + \sqrt{\frac{2\,\sigma_v^2\,\Delta t}{\tau_{\text{drift}}}}\;\boldsymbol{\xi}(t), \tag{3.21}$$

where τ_{drift} is the drift correlation timescale, σ_v^2 governs the variability of the drift, $\mathbf{v}_{\text{mean}}$ is the mean drift velocity, and $\boldsymbol{\xi}(t)$ is a three-dimensional vector whose components are independently drawn from $\mathcal{N}(0,1)$. The Ornstein–Uhlenbeck process is a classical stochastic model widely employed to capture fluctuating physical quantities in many natural and engineered systems (see, e.g., [49] and [50]). Once the drift velocity is updated via (3.21), each molecule's position is updated by

$$\mathbf{r}(t+\Delta t) \;=\; \mathbf{r}(t) \;+\; \mathbf{v}(t+\Delta t)\,\Delta t \;+\; \Delta\mathbf{w}(t), \tag{3.22}$$

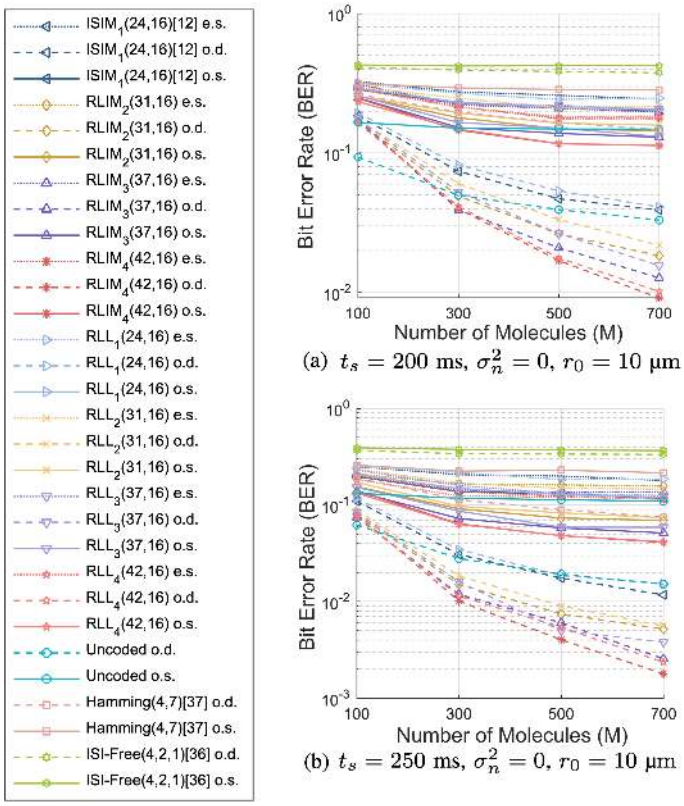


Figure 3.6: MC Simulation Results with Time-Varying Drift

where $\Delta\mathbf{w}(t)$ is a three-dimensional Gaussian random vector. Each component of $\Delta\mathbf{w}(t)$ is drawn independently from $\mathcal{N}\big(0,\, 2D\,\Delta t\big)$, thereby modeling the diffusive displacement in accordance with Eq. 1.1.

We use the same simulation parameters from Table 3.3. Table 3.6 further specifies the drift and particle-tracking parameters employed in our drift-included MC simulations. The settings in Table 3.6 produce a moderate but slowly varying drift that causes noticeable changes in the mean molecule arrival count over successive intervals. To prevent excessive computational overload, the channel capacity is set to 5 seconds. For a signal interval of 0.2 seconds, this corresponds to a channel memory of 25 in the Uncoded case. Since we have a discrete time step of 1 ms, signal interval values given in Table 3.1 are rounded to the nearest integer.

The corresponding BER values of compared coding schemes for a generic MC channel with a time-varying drift are provided in Fig. 3.6. Under these non-stationary conditions, a dynamic threshold consistently achieves lower bit error rates[3.9] than the corresponding fixed threshold-based method, demonstrating the need for adaptive detection in such MC channels. As can be seen in Fig. 3.6, RLIM and RLL codes with dynamic detection surpass the compared methods, except at $M = 100$, where uncoded transmission obtain the lowest BER. In an elementwise comparison between RLIM and the RLIM-corrected RLL baseline across 96 distinct drifted settings, RLIM wins 72/96 cases (75%) with a mean fold improvement of $\approx 1.1\times$ at its winning points, whereas RLL wins 24/96 with a mean fold of $\approx 1.056\times$. By order: i=1: 18 vs. 6 (1.069× vs. 1.042×); i=2: 20 vs. 4 (1.113× vs. 1.051×); i=3: 20 vs. 4 (1.146× vs. 1.095×); i=4: 14 vs. 10 (1.080× vs. 1.051×), with the margin narrowing at i=4 because the two codebooks have very similar 1-bit counts (hence smaller normalization gaps in Table 3.2). These results, together with the fact that the dynamic detector of RLIM avoids the extra *min* constant required by the dynamic detector for our RLL baseline, justify including at least one 1-bit in each RLIM code under time-varying channels.

3.4 Code Availability

At https://github.com/MelihSahinEdu/McChannelCoding.git, Python implementations of binomial simulator, code space generation, encoding, decoding, analytical threshold estimation, detection, optimal static threshold finder, and error correction algorithms for any chosen $\text{RLIM}_i(n, k)$ are provided.

[3.9] For particle-tracking simulations, to determine the optimal detection parameters, we initially sent a total of 53760 bits (7 contiguous runs of the encodings of randomly chosen bit sequences of length 7680). After having determined the optimal detection parameters, to calculate each respective BER value, we then sent a total of 112000 bits (7 contiguous runs of the encodings of randomly chosen bit sequences of length 16000).

Chapter 4

CONCLUSION

This section is based on our paper [52][4.1] and preprint [53]. This thesis has developed novel coding techniques to address two fundamental challenges in diffusion-based molecular communication (MC): the high cost of molecule releases and severe inter-symbol interference (ISI).

Chapter 2 proposes Molecular Arithmetic Coding (MoAC), a novel arithmetic source coding method to mitigate ISI in MC, which builds upon the general constrained arithmetic channel coding scheme in [18]. To address the finite-precision limitations of MoAC, we have introduced Molecular Arithmetic with Prefix Coding (MoAPC), which guarantees unique decodability. Simulation results show that MoAPC outperforms both classical arithmetic coding (AC) and substitution arithmetic coding (SAC), which is our trivial adaptation of AC to MC. We have shown that MoHuffman [13], though it significantly improves channel reliability compared to conventional Huffman coding, does not always produce an optimal Molecular Prefix Coding (MoPC*) codebook. Accordingly, we have used a brute-force algorithm in deriving MoPC* codebooks.

Future work should focus on developing polynomial-time algorithms for finding MoPC* codebooks. Additionally, integrating self-synchronization into MoAC can improve its reliability. In MC, accurate normalization of length and power values is essential. We have adopted the normalization procedure from [26], fitting it to the probabilistic nature of source coding. This normalization procedure should be adhered to in all future MC source coding research to ensure a fair comparison among

[4.1] Personal use permitted; for any other use, contact IEEE.

different coding methods. The application areas of MoAC extend well beyond MC, addressing broader contexts (such as wireless sensor networks) where transmitting a 1-bit incurs higher energy costs than a 0-bit—a phenomenon known as energy consumption disparity (ECD) [51]. We proved that MoAC reduces the transmission of 1-bits approximately by 25% compared to AC, leading to significant energy savings.

Chapter 3 introduces a $O(n)$ time, $O(1)$ auxiliary space error-correction algorithm (Algorithm 3.2) that maps any detected word to its nearest sequence (in Hamming distance) satisfying the (i, ∞)–RLL constraint with leading i zeros. Algorithm 3.2 is equivalent Viterbi with a last-wins tie-break, but it is strictly lighter computationally: it runs in $O(n)$ time with $O(1)$ auxiliary memory, whereas a standard trellis Viterbi decoder for (i, ∞)–RLL requires $O((i+1)n)$ time and $O((i+1)n)$ space with full backpointers (or $O(i+1)$ space with checkpointed or streaming traceback). On our MC channels, Algorithm 3.2 attains substantially lower BER than hard-decision Viterbi with other common tie-breaks (e.g., first-wins and random).

We also define $\mathrm{RLIM}_i(n, k)$ codes as a fixed-rate subset of size 2^k chosen to minimise the total number of 1-bits across the RLL constraint set. Under the standard normalisation, this increases molecules per transmitted 1-bit and delivers further BER gains over RLL codes. In drift-free channels with an optimally tuned static threshold, dropping the "at least one 1-bit" property of RLIM codes makes them coincide with classical (i, ∞)–RLL (with leading merging zeros); nevertheless $\mathrm{RLIM}_i(n, k)$ remains distinct because of its minimum-weight selection. When drift is time-varying, "at least one 1-bit" property is useful and dynamic detection is preferred.

Future work should focus on developing more precise theoretical distributions for ${}^{i}N_{\hat{0}}^{\text{average}}$ and ${}^{i}N_{1}^{\text{average}}$, which could improve the BER of static threshold estimation and potentially eliminate the necessity of sending pilot signals before communication when the channel parameters are known.

Overall, the success of the MoAPC and MoPC* source codes, along with the

$\mathrm{RLIM}_i(n, k)$ channel codes, represents a significant advancement in molecular communication, reducing molecule consumption and driving bit- and symbol-error rates markedly lower than prominent baselines. These results demonstrate that efficient and reliable diffusion-based communication is feasible and set a clear benchmark for the next generation of molecular communication research.

BIBLIOGRAPHY

[1] M. Kuscu, E. Dinc, B. A. Bilgin, H. Ramezani, and O. B. Akan, "Transmitter and receiver architectures for molecular communications: A survey on physical design with modulation, coding, and detection techniques," *Proceedings of the IEEE*, vol. 107, no. 7, pp. 1302–1341, 2019.

[2] T. Nakano, A. W. Eckford, and T. Haraguchi, *Molecular Communication*. Cambridge University Press, 2013.

[3] D. Kilinc and O. B. Akan, "Receiver design for molecular communication," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 705–714, 2013.

[4] B. Tepekule, A. E. Pusane, H. B. Yilmaz, C.-B. Chae, and T. Tugcu, "Isi mitigation techniques in molecular communication," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, no. 2, pp. 202–216, 2015.

[5] E. B. Pehlivanoglu, B. D. Unluturk, and O. B. Akan, *Modulation in Molecular Communications: A Look on Methodologies*. Cham: Springer International Publishing, 2017, pp. 79–97.

[6] M. Kuran, H. B. Yilmaz, T. Tugcu, and I. Akyildiz, "Modulation techniques for communication via diffusion in nanonetworks," 12 2013.

[7] N. Garralda, I. Llatser, A. Cabellos-Aparicio, and M. Pierobon, "Simulation-based evaluation of the diffusion-based physical channel in molecular nanonet-

works," in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2011, pp. 443–448.

[8] D. L. Ermak and J. A. McCammon, "Brownian dynamics with hydrodynamic interactions," *The Journal of Chemical Physics*, vol. 69, no. 4, pp. 1352–1360, 08 1978.

[9] H. B. Yilmaz, A. C. Heren, T. Tugcu, and C.-B. Chae, "Three-dimensional channel characteristics for molecular communications with an absorbing receiver," *IEEE Communications Letters*, vol. 18, no. 6, pp. 929–932, 2014.

[10] H. B. Yilmaz, C.-B. Chae, B. Tepekule, and A. E. Pusane, "Arrival modeling and error analysis for molecular communication via diffusion with drift," in *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, ser. NANOCOM' 15. New York, NY, USA: Association for Computing Machinery, 2015.

[11] M. Hosseini, D. Pratas, and A. J. Pinho, "A survey on data compression methods for biological sequences," *Information*, vol. 7, no. 4, 2016. [Online]. Available: https://www.mdpi.com/2078-2489/7/4/56

[12] A. O. Kislal, B. C. Akdeniz, C. Lee, A. E. Pusane, T. Tugcu, and C.-B. Chae, "Isi-mitigating channel codes for molecular communication via diffusion," *IEEE Access*, vol. 8, pp. 24 588–24 599, 2020.

[13] H. Hyun, C. Lee, M. Wen, S.-H. Kim, and C.-B. Chae, "Isi-mitigating character encoding for molecular communications via diffusion," *IEEE Wireless Communications Letters*, pp. 1–1, 2023.

[14] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[15] J. J. Rissanen, "Generalized kraft inequality and arithmetic coding," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198–203, 1976.

[16] D. R. Hankerson, P. D. Johnson, and G. A. Harris, *Introduction to Information Theory and Data Compression*, 2nd ed. GBR: Chapman & Hall, Ltd., 2003.

[17] A. Shahbahrami, R. Bahrampour, M. S. Rostami, and M. A. Mobarhan, "Evaluation of huffman and arithmetic algorithms for multimedia compression standards," *CoRR*, vol. abs/1109.0216, 2011. [Online]. Available: http://arxiv.org/abs/1109.0216

[18] G. N. N. Martin, G. G. Langdon, and S. J. P. Todd, "Arithmetic codes for constrained channels," *IBM Journal of Research and Development*, vol. 27, no. 2, pp. 94–106, 1983.

[19] K. Kerpez, "Runlength codes from source codes," *IEEE Transactions on Information Theory*, vol. 37, no. 3, pp. 682–687, 1991.

[20] A. Steadman and I. Fair, "Variable-length constrained sequence codes," *IEEE Communications Letters*, vol. 17, no. 1, pp. 139–142, 2013.

[21] P. Bradford, M. J. Golin, L. L. Larmore, and W. Rytter, "Optimal prefix-free codes for unequal letter costs: Dynamic programming with the monge property," *Journal of Algorithms*, vol. 42, no. 2, pp. 277–303, 2002.

[22] M. Golin and G. Rote, "A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs," *IEEE Transactions on Information Theory*, vol. 44, no. 5, pp. 1770–1781, 1998.

[23] B. Marcus, R. Roth, and P. Siegel, *Introduction to Coding for Constrained Systems*, 2001. [Online]. Available: https://personal.math.ubc.ca/~marcus/Handbook/

[24] D. Tang and L. Bahl, "Block codes for a class of constrained noiseless channels," *Information and Control*, vol. 17, no. 5, pp. 436–461, 1970.

[25] J. V. E. Hoggatt and M. Bicknell-Johnson, "Fibonacci convolution sequences," *Fibonacci Quarterly*, vol. 15, no. 2, pp. 117–122, Apr. 1977.

[26] M. C. Gursoy, E. Basar, A. E. Pusane, and T. Tugcu, "Index modulation for molecular communication via diffusion systems," *IEEE Transactions on Communications*, vol. 67, no. 5, pp. 3337–3350, 2019.

[27] H. B. Yilmaz, "Molecular communication (mucin) simulator matlab central file exchange. 2020." [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/46066-molecular-communication-mucin-simulator

[28] C. Freiling, D. Jungreis, F. Theberge, and K. Zeger, "Self-synchronization of huffman codes," in *IEEE International Symposium on Information Theory, 2003. Proceedings.*, 2003, pp. 49–.

[29] T. Guionnet and C. Guillemot, "Soft decoding and synchronization of arithmetic codes: Application to image transmission over noisy channels," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 12, pp. 1599–609, 02 2003.

[30] C. Boyd, J. Cleary, S. Irvine, I. Rinsma-Melchert, and I. Witten, "Integrating error detection into arithmetic coding," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 1–3, 1997.

[31] G. Elmasry, "Joint lossless-source and channel coding using automatic repeat request," *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 953–955, 1999.

[32] I. Sodagar, B.-B. Chai, and J. Wus, "A new error resilience technique for image compression using arithmetic coding," in *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, vol. 4, 2000, pp. 2127–2130 vol.4.

[33] A. Zribi, S. Zaibi, R. Pyndiah, and A. Bouallegue, "Chase-like decoding of arithmetic codes with image transmission applications," in *2009 Fifth International Conference on Signal Image Technology and Internet Based Systems*, 2009, pp. 200–206.

[34] J. Chou and K. Ramchandran, "Arithmetic coding based continuous error detection for efficient arq-based image transmission," in *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No.97CB36136)*, vol. 2, 1997, pp. 1005–1009 vol.2.

[35] D. Malak and O. B. Akan, "Molecular communication nanonetworks inside human body," *Nano Communication Networks*, vol. 3, no. 1, pp. 19–35, 2012.

[36] P.-J. Shih, C. han Lee, P.-C. Yeh, and K.-C. Chen, "Channel codes for reliability enhancement in molecular communication," *IEEE Journal on Selected Areas in Communications*, vol. 31, pp. 857–867, 2013.

[37] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[38] Y. Lu, M. D. Higgins, and M. S. Leeson, "Diffusion based molecular communications system enhancement using high order hamming codes," in *2014 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP)*, 2014, pp. 438–442.

[39] D. Tang and L. Bahl, "Block codes for a class of constrained noiseless channels," *Information and Control*, vol. 17, no. 5, pp. 436–461, 1970.

[40] C. Bai, M. S. Leeson, and M. D. Higgins, "Minimum energy channel codes for molecular communications," *Electronics Letters*, vol. 50, no. 23, pp. 1669–1671, 2014. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2014.3345

[41] O. F. Kurmaev, "Constant-weight and constant-charge binary run-length limited codes," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4497–4515, 2011.

[42] V. Braun and K. Schouhamer immink, "An enumerative coding technique for dc-free runlength-limited sequences," *IEEE Transactions on Communications*, vol. 48, no. 12, pp. 2024–2031, 2000.

[43] A. Hareedy and R. Calderbank, "Loco codes: Lexicographically-ordered constrained codes," *IEEE Transactions on Information Theory*, vol. 66, no. 6, pp. 3572–3589, 2020.

[44] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[45] H. Balakrishnan, C. Termanm, and J. White, "Viterbi decoding of convolutional codes (lecture notes)," MIT 6.02 Course Notes, 2011, hard-decision decoding, Hamming branch metric, ACS algorithm.

[46] A. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 751–772, 1971.

[47] A. G. Shapin, D. V. Kleyko, E. V. Osipov, and O. G. Melentyev, "Performance peculiarities of viterbi decoder in mathworks simulink, gnu radio and other systems with likewise implementation," in *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, 2018, pp. 288–292.

[48] M. Damrath, "Channel coding in molecular communication," 2020, online. [Online]. Available: https://nbn-resolving.org/urn:nbn:de:gbv:8-mods-2020-00366-4

[49] C. Gardiner, *Stochastic Methods: A Handbook for the Natural and Social Sciences*, 4th ed., ser. Springer Series in Synergetics. Springer Berlin, Heidelberg, Jan.16 2009, hardcover, XVIII, 447 pages. Copyright Springer-Verlag Berlin Heidelberg 2009.

[50] N. V. Kampen, "Stochastic processes in physics and chemistry," ser. North-Holland Personal Library, N. V. Kampen, Ed. Amsterdam: Elsevier, 2007.

[51] Y.-H. Zhu, E. Li, and K. Chi, "Encoding scheme to reduce energy consumption of delivering data in radio frequency powered battery-free wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 4, pp. 3085–3097, 2018.

[52] M. Şahin, B. E. Ortlek, and O. B. Akan, "Molecular arithmetic coding (moac) and optimized molecular prefix coding (mopc) for diffusion-based molecular communication," *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, vol. 11, no. 1, pp. 66–77, 2025.

[53] M. Şahin and O. B. Akan, "Run-length-limited isi-mitigation (rlim) coding for molecular communication," 2025, version 2; https://arxiv.org/abs/2411.15955v2.