

# Explicit and Efficient Construction of Nearly Optimal Rate Codes for the Binary Deletion Channel and the Poisson Repeat Channel

Ittai Rubinstein  

Blavatnik School of Computer Science, Tel-Aviv University, Israel  
QEDMA Quantum Computing, Tel-Aviv, Israel

---

## Abstract

Two of the most common models for channels with synchronisation errors are the Binary Deletion Channel with parameter  $p$  ( $BDC_p$ ) – a channel where every bit of the codeword is deleted i.i.d with probability  $p$ , and the Poisson Repeat Channel with parameter  $\lambda$  ( $PRC_\lambda$ ) – a channel where every bit of the codeword is repeated Poisson( $\lambda$ ) times.

Previous constructions based on synchronisation strings yielded codes with rates far lower than the capacities of these channels [6, 9], and the only efficient construction to achieve capacity on the BDC at the time of writing this paper is based on the far more advanced methods of polar codes [23].

In this work, we present a new method for concatenating synchronisation codes and use it to construct simple and efficient encoding and decoding algorithms for both channels with nearly optimal rates.

**2012 ACM Subject Classification** Mathematics of computing → Coding theory

**Keywords and phrases** Error Correcting Codes, Algorithmic Coding Theory, Binary Deletion Channel

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2022.105

**Category** Track A: Algorithms, Complexity and Games

**Related Version** *Full Version*: <https://arxiv.org/abs/2111.00261?context=math.IT>


**Funding** *Ittai Rubinstein*: The Deutsch institute fund.

**Acknowledgements** We thank Roni Con, Aviad Rubinstein and Muli Safra for their helpful comments on previous drafts.

## 1 Introduction

The theory of error-correcting-codes deals with methods for encoding messages to be sent over noisy media in such a manner that they can be correctly decoded afterwards. Initially introduced by Shannon [22], this field has proven to be instrumental in understanding the theory of computation, and has had a wide variety of applications in other fields, such as communications, and computational biology [24].

The most commonly considered models are “Synchronous Models” - models where the message may be altered or erased, but every letter that was received can be traced back to its original position in the transmitted message. This category includes models such as the Binary Symmetry Channel (BSC) where some of the bits in the transmitted message are flipped (i.e. changed from 1 to 0 or vice versa), and the Binary Erasure Channels (BEC) where some of the bits of the transmitted message are replaced with a question mark (but are not removed, thus preserving the alignment between the transmitted message and the received message). These models can be adversarial (such as [14]), where the code must correct any error the channel may produce, or average-case (such as [22]), where the effect of the channel is random and decoding only needs to succeed w.h.p.

 © Ittai Rubinstein;  
licensed under Creative Commons License CC-BY 4.0  
49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).  
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;  
Article No. 105; pp. 105:1–105:17



Leibniz International Proceedings in Informatics  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Synchronous models are very well researched, and have a variety of efficient encoding and decoding algorithms [24]. The main method used for such channels are linear error-correcting-codes and they rely heavily on the fact that the letters of the received messages can be mapped back into letters of the transmitted messages.

However, in many real world applications, deletions and insertions can cause the received codewords to be misaligned and the decoder must also deal with synchronisation errors [18, 17]. Perhaps the most intuitive asynchronous channel is the Binary Deletion Channel. This is a channel where every transmitted bit is deleted i.i.d with probability  $p$  (where  $1 > p > 0$  is a parameter of the channel).

Here, unlike with the binary erasure channel, deleted bits are not replaced with a question mark, but are completely removed from the sequence, shifting the rest of the received codeword. For instance, when transmitting the codeword  $w = (1, 0, 0, 0, 1, 1)$ , the BEC may result in the received codeword  $r_{\text{BEC}} = (1, 0, ?, ?, 1, ?)$  while a similar pattern of deletions would result in the received codeword  $r_{\text{BDC}} = (1, 0, 1)$ .

This channel represents a simple model for many real-life systems in which there is a loss of information due to synchronisation errors. Moreover, the tools developed for this channel have been instrumental in a variety of other fields [18, 17].

A slightly more complex model which we will also consider, is the Poisson Repeat Channel with parameter  $\lambda > 0$  ( $\text{PRC}_\lambda$ ). This is a channel where every transmitted bit is received  $\text{Poisson}(\lambda)$  times. While originally used to help prove lower bounds on the capacity of binary deletion channels, Poisson repeat channels are interesting in their own right. Indeed, Poisson repeats can model every-day examples like a sticky key in a keyboard, as well as deeper technological issues such as errors in single photon generation (a crucial step in light-based quantum computing) [2].

However, the PRC also presents a slightly greater challenge. This is because bits can now be received more than once, resulting in several new types of synchronisation errors. Continuing with our previous example where the codeword  $w = (1, 0, 0, 0, 1, 1)$  was transmitted, the received codeword in the BDC model will always start with at most a single 1 bit, unless all three 0s were deleted. In the Poisson repeat channel, this is not the case and the received codeword  $r_{\text{PRC}} = (1, 1, 0, 1)$  is a possible outcome.

Several previous results have shown an interesting connection between these two noise models. For instance, Mitzenmacher and Drinea's lower bound for the capacity of the BDC channel [19] is based on their previous lower bound for the capacity of the PRC channel, and Con and Shpilka's constructive codes for the BDC channel [6] are also applicable to the PRC channel.

In this paper we will focus our attention on these two channels, but we believe our tools and approaches can be applicable to other asynchronous channels as well.

## 1.1 Previous Work

Asynchronous channels present us with a varied field of research, and we will not be able to cover all of its results here. The excellent surveys by Mitzenmacher, Cheraghchi et al and Mercier et al [18, 17, 5] give a more detailed background.

Determining the capacity of the  $\text{BDC}_p$  channel, remains an open problem, and so far it has been answered only for some extremal cases. When  $p \rightarrow 0$ , the capacity of this channel is  $1 - h(p)$  [15] (where  $h(\cdot)$  is the binary entropy function), when  $p \rightarrow 1$ , the capacity is  $\mu(1 - p)$  where  $\frac{1}{9} < \mu \leq 0.4143 - o(1)$  [19, 8] (where  $o(1)$  is w.r.t the block size  $n$ ), and [25] give lower bounds for some of the intermediate values of  $p$ .

However, the lower bound on the capacity by Mitzenmacher and Drinea [19] is not based on an efficient construction. Recently, Con and Shpilka constructed a family of error-correcting-codes with rate  $r \geq \frac{1-p}{16}$  for the  $BDC_p$  channel and  $r \geq \frac{\lambda}{17}$  for the  $PRC_\lambda$  (when  $\lambda < \frac{1}{2}$ ) [6], improving upon the results of Guruswami and Li who presented the first explicit codes with rate  $\Theta(1-p)$  in [9]. This was improved upon by Tal et al and by Pfister and Tal who showed that polar codes can be used to construct efficiently decodable codes with optimal rates for the BDC channel [23, 20].

Haeupler and Shahrasbi [13] construct a family of efficient codes for InsDel channels with a sufficiently large alphabet, and Haeupler, Rubinstein and Shahrasbi improve the decoder in [12], reaching quasi-linear complexity.

In the adversarial model, Guruswami and Wang [10] showed that there are codes with rate  $1 - \tilde{O}(\sqrt{\delta})$  that can correct up to  $\delta n$  errors. This rate was improved by Cheng et al. [3] and further by Haeupler [11]. More recently it was shown by Con, Shpilka and Tamo [7] that, surprisingly, linear error correcting codes are also effective for adversarial InsDel channels.

## 1.2 Main Contribution

In this work we will construct a family of codes, with efficient encoding and decoding algorithms, whose rates are arbitrarily close to the capacities of the BDC and PRC channels. Unlike the previous results of Tal et al. and of Pfister and Tal [23, 20], this construction does not require the more advanced machinery of polar codes. Furthermore, the construction presented has a decoding error probability of  $\exp\left(-\Theta\left(n^{\frac{1}{6}}\right)\right)$  (where  $n$  is the block length) with a quasi-linear complexity decoder, while Pfister and Tal's code requires  $n^{\frac{3}{2}+\varepsilon}$  time for the same error probability.

Both our code and Pfister and Tal's construction assume that we are given some inner code which achieves a high rate on the channel but which does not necessarily have efficient encoding and decoding algorithms, and both methods produce a new version of this code with efficient encoding and decoding. However, Pfister and Tal's construction requires this code to be generated by a hidden-Markov distribution, while the construction presented here can be used with any inner code. Li et al. proved that there exists a finite hidden-Markov distribution code that achieves capacity for this channel [16] and this model can clearly be found in  $O(1)$  time using an enumeration technique similar to the one described in Section 4.1, but such a code has not been found yet.

► **Theorem 1 (Main Result (informal)).** *Any (possibly inefficient) family of codes for either the BDC or the PRC channel can be converted into a family of codes for the same channel with an arbitrarily close rate, that has encoding and decoding algorithms with a quasi-linear complexity.*

It should be noted that while this complexity is asymptotically very good, it hides within it a very large constant factor, and while we do not have an exact bound on it, we expect it to grow as the rate of the code approaches the capacity of the channel.

Nonetheless, this allows us to construct to construct a family of efficient codes which achieve rates of  $\frac{1-p}{9}$  for the  $BDC_p$  channel, thus completing the line of works started by Guruswami and Li [9] and continued by Con and Shpilka [6].

Our construction is based on a new technique for tracking inner codewords in a concatenation of synchronisation codes, which we hope may be useful in other cases as well:

Most previous constructions use buffers of 0s as the delimiters between inner codewords. Then, by bounding the probability that the channel would transform any substring of the inner codeword into a sufficiently long sequence of 0s, they can ensure that long sequences of

0s in the received codeword mostly correspond to delimiters. In other words, when separating the received codeword into inner codewords, one searches the entire string for patterns that may have come from a delimiter.

In our construction, we will use delimiters in a very different manner. Instead of searching the entire codeword for the delimiters, we will use our knowledge of the length of the inner codeword and the average expansion of the channel to produce a prior estimate for the distance between consecutive delimiters. Using this prior estimate, we are able to find the delimiters one after another, without looking at the entire codeword.

This new method allows us to drastically reduce the probability that even a single inner delimiter will be missed, while reducing the overhead of the delimiters to a negligible fraction of the codeword. The ability to decode under the assumption that all delimiters will be found simplifies the outer code in our construction, and the fact that we will not search for a delimiter within an inner codeword allows us to use a general inner code, resulting in a nearly optimal rate.

### 1.3 An Overview of Con and Shpilka's Construction

Since our approach will be similar to that of [6], we will begin with a short overview of their construction, which is based on a concatenation of codes.

Initially, the message is divided into segments of length  $\sigma = O(1)$ . These are thought of as members of an alphabet  $\Sigma$  of size  $|\Sigma| = 2^\sigma$  and can be encoded using [13]. Each letter in the encoded message is then converted back to a string of  $\sigma$  binary symbols and is encoded using an inner code. Since the inner code is only applied to strings of length  $\sigma = O(1)$ , it can be inefficient without affecting the asymptotic complexity of the encoding / decoding algorithms. The encoded strings are appended and separated by delimiters - in this case buffers of 0s.

Con and Shpilka's construction has a quasi-linear encoding algorithm and a quadratic decoding algorithm, where the computational bottleneck of the decoding algorithm comes from decoding the outer code. The improved decoding algorithm for Hauptler and Shahrashbi's code [12] can be used with Con and Shpilka's code to reduce the complexity of their decoding algorithm to a quasi-linear time as well.

### 1.4 Sketch of the Proof

Our construction will be based on a similar strategy, but with a few key differences. Firstly, we note that most of the overhead of this code is caused by the fact that the inner code is designed to preserve a certain structure.

By removing this structure we are able to significantly increase the rate of our code. However, this comes at a cost - separating unstructured codewords from the delimiters is made far more difficult. We overcome this using a slightly more complex construction of delimiters and a careful analysis.

In addition, the delimiters themselves account for another constant fraction of the overhead of Con and Shpilka's code. By using a recursive concatenation, we are able to reduce the cost of these delimiters to a negligible fraction of the overhead.

At each step of this recursion, we will assume that there exists a BDC/PRC code with message length  $k$  and block length  $n$ , and we will construct a code with message length  $k^2$  and block length  $n' = (1 + o(1))nk$ .

We will do this by separating the  $k^2$  bit message into  $k$  strings of  $k$  bits each. We will think of each of these  $k$ -bit strings as a member of an alphabet of size  $|\Sigma| = 2^k$  and use a ReedSolomon  $[k + 2t, k, t]_{2^k}$  code to give it some redundancy (i.e. a Reed-Solomon code over a field of size  $2^k$ , message length  $k$  and block length  $k + 2t$  with distance  $t = o(k)$ ).

If we were constructing a code for a discrete memoryless channel (DMC) such as the BEC, this step might not be very surprising, because without synchronisation errors we could map the received codeword back into the letters of the Reed-Solomon codeword. However, in our case this might seem somewhat counter-intuitive, since Reed Solomon codes offer no protection against the synchronisation errors we are trying to correct. This step works for asynchronous channels as well because our delimiters are designed to fully preserve the synchronisation *between* inner codewords and the Reed Solomon code will only need to compensate for a small number of local decoding failures of inner codewords.

We will encode each of the  $k + 2t$  letters of the Reed Solomon codeword using our inner code. This will output a list of  $k + 2t$  strings of length  $n$  bits each. Finally, we will append these strings after inserting a delimiter between each two.

The delimiters will be made up of two parts: a positioning string which will help us find the delimiter and two partitioning strings which will help us separate between the delimiters and the inner codewords themselves. The reason that we need partitioning strings, is that we make no assumptions about the structure of the inner code. Therefore, any sequence of bits that originated from the delimiter could have originated from the inner codeword.

For instance, suppose we had used buffers of 0s as our delimiters. Since we make no assumptions about the structure of the inner code, we have no way of knowing whether or not the inner codeword begins with a sequence of 0s, so we can't tell where the delimiter ends and the inner codeword begins.

This makes separating the two a very difficult task and will be at the heart of our construction. Our separation between inner codewords and delimiters will not be completely accurate, but we will be able to bound the effect this has on the decoding failure probability by using the fact that the inner code is designed to deal with (some) deletions.

Both parts of the delimiter will be based on an idea we call “valleys”. Similar to the markers defined by Cheraghchi et al [4], we will define valleys to be a long sequence of 0s followed by a long sequence of 1s (when looking at the cumulative sum of the string minus  $\frac{1}{2}$ , these translate to local minima - see Figure 1). Unless one of these two sequences is completely deleted by the channel, a valley in the transmitted message will result in a valley in the received message.

We will use this observation to align indices within the received message to their source in the transmitted message. We will start with an estimate of where the center of some valley from the transmitted message should be in the received message, and then we will go downhill to the nearest local minima (see Algorithm 1). By bounding both the probability that one of these sequences was removed and the probability that our initial estimate was outside the bounds of the received valley, we can correlate the center of the received valley to the center of the transmitted valley with high probability.

Each positioning string will be a long valley, and each partitioning string will be a short valley. We set the positioning string to be long, because we need to be able to find its center, given only a very rough estimate. On the other hand, setting the partitioning strings this long would reduce the accuracy of its separation from the inner codewords.

The decoding algorithm will be similar to the encoding algorithm, but in a reversed order. First, we will align the received message by locating the centers of the positioning strings. Then we will use the partitioning strings to separate the delimiters from the inner codewords, and apply the inner code decoding to obtain the inner codewords. Finally, we use the Reed Solomon decoding to correct up to  $t$  errors that may have occurred.

## 1.5 Organization

In Section 2 we will define basic notations, show some well-known inequalities that we will use in our analysis and present the basic building block of our construction. Section 3 contains the construction of our recursive step and in Section 4 we will prove the basis of the recursion and show how we can connect it to the recursive step. In Section 5 we will extend our results to the Poisson repeat channel. Finally, in Section 6 we will discuss the implications and limitations of these results, as well as potential avenues for future research. We leave the slightly more technical proof that of our bound for the decoding failure probabilities to the full version (see Section 6 of the full version [21]).

## 2 Preliminaries

### 2.1 Average-case Codes

► **Definition 2.** Let  $\Sigma$  be a finite set and let  $k, n \in \mathbb{N}$  be positive integers.

We will say that  $C$  is a random channel acting on the alphabet  $\Sigma$  and block-length  $n$  if it maps any member of  $\Sigma^n$  to a distribution on some set  $\mathcal{Y}$ .

Furthermore, we will say that encoding and decoding algorithms  $E : \Sigma^k \rightarrow \Sigma^n, D : \mathcal{Y} \rightarrow \Sigma^k$  for this channel with message length  $k$  have rate  $\rho = \frac{k}{n}$  and a decoding failure probability (DFP) of

$$\delta = \max_{m \in \{0,1\}^k} \{\Pr [D(C(E(m))) \neq m]\}.$$

In other words, the DFP of a code is the probability that a message will be decoded incorrectly if the message was chosen adversarially, but the effects of the channel were random.

► **Definition 3.** Let  $C$  be a random channel. We will say that  $\mathcal{F} = \{(E_i, D_i)\}_{i \in \mathbb{N}}$  is a family of codes for  $C$  if:

- The message lengths  $k_i$  of  $E_i, D_i$  are unbounded ( $k_i \xrightarrow{i \rightarrow \infty} \infty$ ).
- The DFPs  $\delta_i$  of  $E_i, D_i$  in  $C$  are vanishing ( $\delta_i \xrightarrow{i \rightarrow \infty} 0$ ).

Throughout the decoding process we will often attempt to align the received message with the transmitted codeword.

► **Definition 4.** When the channel acts independently on each letter of the input (i.e. when  $C(b_1, \dots, b_n) = C(b_1) \dots C(b_n)$ ), we will say that the  $i$ th coordinate of a message transmitted over some asynchronous channel and the  $j$ th coordinate of the received message are aligned, if the first  $i - 1$  letters of the transmitted message were mapped to a string of length at most  $j$  by the channel and the first  $i$  letters of message were mapped to at least  $j$  letters by the channel.

### 2.2 Probability Inequalities

Throughout this paper we will bound the probability that several parts of our construction will fail. This will require several bounds on the tails of Poisson and binomial distributions, which we will present in this section. Perhaps the most important tool at our disposal will be the Chernoff bound.



■ **Algorithm 1** Align Valley.

---

```

Input : a received codeword  $w \in \{0, 1\}^*$ , estimated center of valley  $j$ 
Output: the center of the valley  $j'$ 
 $j' \leftarrow j$ ;
if  $w[j] = 0$  then
    while  $w[j'] \neq 1$  do
         $i \leftarrow i + 1$ ;
    end
    return  $j' - 1$ ;
else
    while  $w[j'] \neq 0$  do
         $i \leftarrow i - 1$ ;
    end
    return  $j'$ ;
end

```

---

Suppose the channel made the following deletions:

$$d = [\dots] \cdots 0 \cdots \cdots 000 \cdot 0 \cdot \cdots 00 \cdots 0 \cdot 0 \cdot 0 \cdot 0 \cdot 0 \cdot 0 \cdot 1 \cdot 1 \cdot 11 \cdot 11 \cdot 1 \cdots 111 \cdot 1111111 \cdots 1 [\dots]$$

Furthermore, assume that we have some prior estimate that the center of the received valley should be 5 bits from its actual position.

Then the received codeword would be as follows, where the underlined digit signifies our prior estimate for the center of the valley.

$$w = [\dots] 0000000000011111\underline{1}111111111111 [\dots]$$

Algorithm 1 would start from this initial estimate and advance to the left, returning the correct center of the valley.

$$w = [\dots] 000000000000111111111111111111 [\dots]$$

In Figure 1 we show a geometric representation of this algorithm.

■ **3 Recursive Step**

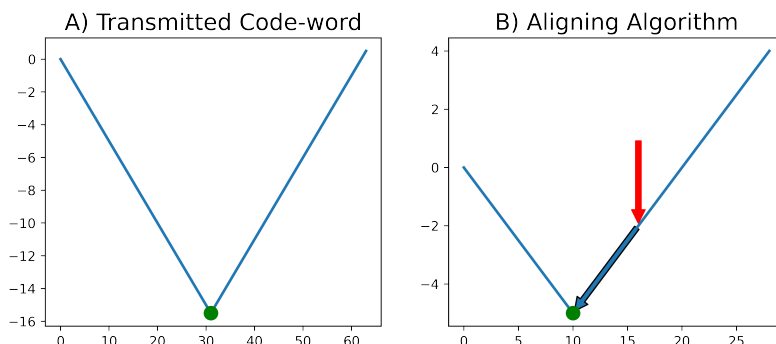
In this section we will define the recursive step in our construction, explain the rationale behind it and begin to prove its correctness. Intuitively, the main theorem we will prove here is that any error correcting code for the BDC with message length  $k$  can be transformed into a code with message length  $k^2$  and that the DFP, rate and complexity of the new code “scale well”. More formally we will show that:

► **Theorem 8 (Recursive Step).** *For some constants  $c_1, c_2, c_3, k_0, \delta_0 > 0$  and for all  $k > k_0, d > 0, 2^{k-1} - k > t > 0, \delta_0 > \delta > 0, 1 > p > 0$  and any error correcting code  $C$  for the  $BDC_p$  channel with message length  $k$ , block length  $n$  and DFP  $\delta$ , there exists an error correcting code  $C'$  for the  $BDC_p$  with message length  $k^2$ , block length  $n' \leq (k + 2t)(n + \frac{d}{1-p})$  and DFP:*

$$\delta' \leq \Pr [Bin(\delta^{c_1}, k + 2t) > t] + c_3 (k + 2t) \exp \left( -c_2 \min \left\{ d, \frac{d^2}{k} \right\} \right).$$

Furthermore, there exist an encoder and decoder for  $C'$  with time complexity  $\tilde{O}(n')$  using up to  $k + 2t$  calls to the encoder and decoder of  $C$ .





■ **Figure 1** A visual representation of the algorithm for aligning valley centers on a specific example of a valley with parameters  $(32, 32)$ , being sent over a  $BDC_p$  channel with parameter  $p = \frac{1}{2}$ . The plots go up by  $\frac{1}{2}$  whenever the relevant string has a 1 and down by  $\frac{1}{2}$  whenever it has a 0. **A)** The transmitted message, with the green dot representing the center of the valley. **B)** First, the channel deletes some of the bits resulting in a skewed valley. We are given some prior estimate for the center of the valley (red arrow). The algorithm goes down the valley (blue arrow), until it terminates at the center (green dot).

We will use this construction in two scenarios: to improve the base of the recursion and for the steps of the recursion. In Table 1, we present the parameters of Theorem 8 and the asymptotic values for both use cases.

■ **Table 1** The parameters of Theorem 8, and their asymptotic values in the two use cases.

Parameter	Description	Recursion Base   Step
$k$	Message Length	
$n$	Block Length	$\Theta\left(\frac{k}{1-p}\right)$
$d$	Delimiter Length	$\Theta\left(k^{\frac{2}{3}}\right)$
$t$	Reed Solomon Redundancy	$o(k) \mid \Theta\left(k^{\frac{2}{3}}\right)$
$\delta$	Inner Code DFP	$o(1) \mid \exp\left(-\frac{c_2}{2} k^{\frac{1}{6}}\right)$
$c_1, c_2, c_3$	Constants	$\frac{1}{34}, \frac{1}{256}, 6$

The first set of values is used for improving of the base of the recursion. In this case, our only bound on  $\delta$  will be that it is an arbitrarily small constant, but its relation to  $k$  will not be exactly known. Our goal in this step of the construction will be to reduce the error probability at the cost of an arbitrarily small but non-negligible cost to the rate of the code, and we will accomplish this by setting  $t$  to be of the order of  $\Theta\left(\delta^{\frac{c_1}{2}} k\right)$ .

## 105:10 Efficient Codes for the BDC and PRC Channels

The second scenario is that of a step in our recursion. We will construct our recursion in such a manner that the DFP of the inner code will be bounded by  $\delta < \exp\left(-\frac{c_2}{2} k^{\frac{1}{6}}\right)$ . By setting  $t = d = \Theta\left(k^{\frac{2}{3}}\right)$  we will be able ensure that on the one hand, the DFP of the final code will be  $\delta' \ll \exp\left(-\frac{c_2}{2} (k^2)^{\frac{1}{6}}\right)$  while on the other, the rate of the code will be reduced by only a factor of  $1 - O\left(\frac{t+d}{k}\right) = 1 - O\left(k^{\frac{1}{3}}\right) = 1 - o(1)$ .

The construction of the code  $C' : \{0, 1\}^{k^2} \rightarrow \{0, 1\}^{n'}$  is as described in the introduction. First, the input string is split into  $k$  parts of length  $k$  each. These are considered as elements in an alphabet of size  $2^k$  and a Reed Solomon encoding with parameters  $[k + 2t, k, t]_{2^k}$  is applied to them. Each of these is encoded using the encoder of  $C$ , a delimiter is appended to each codeword and the concatenation of all of these strings is outputted.

Similarly, the decoding algorithm works by locating the delimiters, separating them from the inner codewords and then decoding each inner codeword using the decoder of  $C$ . The decoded inner codewords are once again viewed as letters in an alphabet of size  $2^k$  and the Reed Solomon decoding is applied.

We will define a delimiter with parameters  $\alpha, \beta$  to be a valley of length  $\beta$  surrounded by two valleys of length  $\alpha$ . In other words

$$\text{Delimiter}(\alpha, \beta) = \text{Valley}(\alpha) \text{Valley}(\beta) \text{Valley}(\alpha) = 0^\alpha 1^\alpha 0^\beta 1^\beta 0^\alpha 1^\alpha$$

The exact values of  $\alpha$  and  $\beta$  are presented in the full version (see Section 6 of [21]), but they will be of the order of  $\alpha = \Theta\left(\frac{\log\left(\frac{1}{\delta}\right)}{1-p}\right)$  and  $\beta = \frac{d}{2(1-p)} - 2\alpha = \Theta\left(\frac{k^{\frac{2}{3}}}{1-p}\right)$ .

In order to complete the construction, we still need to provide methods of locating the delimiters in the received codeword and separating them from the inner codewords. These steps will be explained in the following subsections.

### 3.1 Positioning Strings

We will use the positioning strings to locate the delimiters one at a time.

Let us denote by  $L_i$  the location of the center of the  $i$ th positioning string in the received codeword. We can estimate the location of the center of the first positioning string as being around  $\mathbb{E}[L_1] = (1-p)(n + 2\alpha + \beta)$  bits from the beginning of the received codeword. However, this is only a rough estimate and in reality  $L_1 \sim \text{Bin}(n + 2\alpha + \beta, 1-p)$ . We want to find the exact center.

This is where the valleys come into play. With high probability,  $|L_1 - \mathbb{E}[L_1]|$  will not be much larger than  $\sigma_{L_1} = \sqrt{p(1-p)(n + 2\alpha + \beta)}$  (where  $\sigma_{L_1}$  is the standard deviation of  $L_1$ ), and at least  $\frac{1-p}{2}\beta$  of the bits on either side of the valley will survive the channel. Therefore, so long as  $\sqrt{p(1-p)(n + 2\alpha + \beta)} \ll \frac{1-p}{2}\beta$ , we can expect Algorithm 1 to find the value of  $L_1$  w.h.p.

Once we have found the  $i$ th delimiter, we go on to search for the  $i + 1$ -th. At each step we use the aligned center of the previous positioning string  $L_i$  to obtain an estimate for  $L_{i+1} \sim L_i + \text{Bin}(n + 4\alpha + 2\beta, 1-p)$ , and use its valley to correct our estimate. In the full version we show that the probability that even a single positioning string will not be correctly found is at most  $(k + 2t) \exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right)$  (see Section 6 of [21]).

### 3.2 Partitioning Strings

Once we have located the centers of all of the delimiters, we still need to separate them from the inner codewords. This step is surprisingly difficult, because, unlike Con and Shpilka [6] who designed their inner code to preserve a certain structure that would help them differentiate it from the delimiters, we reduce our overhead precisely by making no assumptions about the structure of the inner code.

For instance, if we were to use buffers of 0s as our delimiters and the inner codewords surrounding the delimiter happened to start / end with sequences of 0s, then we would have had a hard time telling which of the 0s belonged to the delimiter and which belonged to the inner codeword. The same applies for any choice of the delimiters.

Our solution to this problem will be based on two ideas. First, we will attempt to approximate the correct separation as accurately as possible. However, this will not yield a perfect separation and we will need to mitigate the effects of this inaccuracy.

The first step will be accomplished by surrounding the positioning string with two valleys. When decoding we can find the centers of these valleys by going from the positioning string until the ends of its valley and then proceeding to the bottom of the next/previous valleys.

Once we have found the center of a partitioning string, we can estimate the length of the faces of its valley. Each of those will be i.i.d distributed according to  $F \sim \text{Bin}(1-p, \alpha)$ . By guessing  $f = (1-p)\alpha$  we will get a good approximation of  $F$  (to within an error of  $\Theta(\sqrt{p(1-p)\alpha})$ ).

If we were to use this estimate as the separation between the delimiter and the inner codeword we would have a two sided error probability, either due to overshooting (attributing some bits of the inner codeword to the delimiter) or due to undershooting.

In the former case, this would cause us to delete some additional bits from the inner codeword. Since the inner code is designed to deal with deletions, it stands to reason that it might also be able to decode received messages if they were also subject to a small amount of additional deletions. This claim is not straightforward, since the inner code deals with random deletions and we will be subjecting it to a very structured set of deletions. However, with a careful analysis we can bound the effect these deletions can have on the DFP of the inner code.

However, in the latter case we would end up erroneously inserting part of the delimiter to one end of the inner codeword. Since the inner code is for a deletion channel, it might not be able to correctly decode the inner codeword, even after a small number of insertions. Therefore, we have no way of bounding the effect this could have on its DFP.

That is why we want to reduce the probability of undershooting significantly more than the probability of overshooting. To do this, instead of taking the estimate  $f = \mathbb{E}F$ , we will use the estimate  $f = \mathbb{E}[F] + \eta\sigma_F$  for some parameter  $\eta$  (where  $\sigma_F$  is the standard deviation of  $F$ ).

If  $F$  was at most  $\eta$  standard deviations from its expectancy, then we would have  $f - F \in [0, 2\eta\sigma_F]$ . The lower bound means that no bits from the delimiter would ever trickle into the inner codeword, and the upper bound limits the number of bits from the inner codeword we will delete by accidentally attributing them to the delimiter.

In the full version of the paper we bound both the probability that  $f - F \notin [0, 2\eta\sigma_F]$  and the effect these deletions could have on the DFP of the inner code (see Section 6 of [21]).

## 4 The Recursive Construction

In the previous section we defined the manner in which we recursively concatenate our code to increase the message length at a negligible cost to the rate of the code and the complexity of its encoder and decoder. In this section we will construct the base of this recursion and set the parameters for the recursive steps.

### 4.1 The Base of the Recursion

When choosing the base of the recursion we are in essence transforming a lower bound on the capacity of the channel to an actual error correcting code. We will do this in a very inefficient manner, but as with Con and Shpilka's inner code, since this code will have message and block length of  $O(1)$ , this does not affect the asymptotic complexity of our construction.

In order to begin our recursion, we will need a base code with sufficiently low DFP and sufficiently large message length. By definition, a lower bound on the capacity of the channel is a proof that there exists a family of codes for the channel with  $r \geq \text{capacity} - \varepsilon$  for any  $\varepsilon > 0$ . Therefore for any  $k_0, \delta_0 > 0$  there are some codes in that family with message length  $k > k_0$  and DFP  $\delta < \delta_0$ . Let  $\kappa$  be the smallest such block length. Since  $\kappa$  is determined by  $k_0$  and  $\delta_0$ , and since  $k_0$  and  $\delta_0$  are constant parameters of our construction,  $\kappa = f(k_0, \delta_0) = O(1)$  must also be constant.

We will enumerate over values of  $k > k_0$  and for each of them we will attempt to construct a base code. This process will terminate when  $k = \kappa$ , so it will require only a finite number of iterations.

Since we are looking for an encoding map from some finite set of messages  $\{0, 1\}^k$  (where  $k$  is the message length) to some finite set of codewords  $\{0, 1\}^n$  (where  $n < \frac{9}{1-p}k = O(1)$  is the block length), and a decoding map from  $\{0, 1\}^{\leq n} \rightarrow \{0, 1\}^k$ , there are only finitely many pairs of this form. By enumerating over all of these pairs and evaluating their DFP, we will be able to find a base code with message length  $k$  if one exists.

Since all of the steps in this process had a constant complexity, our construction of the inner code had a constant  $O(1)$  complexity. It should be noted that this algorithm is extremely inefficient and that we do not even know how to bound its complexity (except that it is  $O(1)$ ). We hope that future research will address this issue.

### 4.2 Connecting the Recursive Steps

All that remains now is to combine the recursive step shown in Section 3 with the base case constructed in the previous subsection.

Throughout most of the recursion we will use the recursive step defined in Theorem 8 in the following setting:

$$\begin{aligned} \delta_k &< \exp\left(-\frac{c_2}{2}k^{\frac{1}{6}}\right) \\ d_k &= k^{\frac{2}{3}} \\ t_k &= k^{\frac{2}{3}}. \end{aligned} \tag{1}$$

Applying the recursion Theorem 8, for sufficiently large  $k$ , we have:

$$\delta_{k^2} \leq \Pr[\text{Bin}(\delta^{c_1}, k + 2t) > t] + c_3(k + 2t) \exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right). \tag{2}$$

We use Theorem 6 to bound the first term in the new decoding failure probability for sufficiently small  $\delta$ , by

$$\Pr [\text{Bin}(\delta^{c_1}, k + 2t) > t] \leq \exp\left(-\frac{c_1}{4} \log\left(\frac{1}{\delta}\right) t\right) \leq \frac{1}{2} \exp\left(-\frac{c_2}{2} t\right)$$

and the second term for sufficiently large  $k$  by

$$(k + 2t) \exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right) = \exp\left(-c_2 k^{\frac{1}{3}} + O(\log(k))\right) \leq \frac{1}{2} \exp\left(-\frac{c_2}{2} k^{\frac{1}{3}}\right).$$

Combining these inequality gives us a bound on the new DFP:

$$\delta_{k^2} \leq \exp\left(-\frac{c_2}{2} k^{\frac{1}{3}}\right). \quad (3)$$

The overhead of the code can also be easily bounded. From the recursion theorem, we know that  $n_{k^2} \leq \left(n_k + \frac{dk}{1-p}\right)(k + 2t_k)$ . Since  $n_k \geq \frac{k}{1-p}$  (from the upper bounds on the capacity of these channels) and  $d_k = t_k = k^{\frac{2}{3}}$ , we have:

$$\begin{aligned} n_{k^2} &\leq \left(1 + k^{-\frac{1}{3}}\right)^2 \frac{k^2}{k} n_k \leq \left[\left(1 + k^{-\frac{1}{3}}\right) \cdot \left(1 + k^{-\frac{1}{6}}\right)\right]^2 \frac{k^2}{\sqrt{k}} n_{\sqrt{k}} = \\ &= \underbrace{\left[\left(1 + k^{-\frac{1}{3}}\right) \cdot \left(1 + k^{-\frac{1}{6}}\right) \cdot \dots \cdot \left(1 + (k_{\text{base}})^{-\frac{1}{3}}\right)\right]^2}_{=:X} \frac{k^2}{k_{\text{base}}} n_{k_{\text{base}}}. \end{aligned} \quad (4)$$

Since  $n_{k_{\text{base}}} = \frac{1}{r_{\text{base}}} k_{\text{base}}$  (where  $r_{\text{base}}$  is the rate of the base code), it is easy to see that  $r_{k^2} = \frac{k^2}{n_{k^2}} = \frac{1}{X} r_{\text{base}}$ . By bounding the value of  $X$ , we can bound the increase in the overhead of the code due to the recursion.

$$\begin{aligned} X &= \left[\left(1 + k^{-\frac{1}{3}}\right) \cdot \left(1 + k^{-\frac{1}{6}}\right) \cdot \dots \cdot \left(1 + (k_{\text{base}})^{-\frac{1}{3}}\right)\right]^2 \\ &\leq \exp\left(2k^{-\frac{1}{3}} + 2k^{-\frac{1}{6}} + \dots + 2(k_{\text{base}})^{-\frac{1}{3}}\right) \leq \exp\left(2 \frac{(k_{\text{base}})^{-\frac{1}{3}}}{1 - (k_{\text{base}})^{-\frac{1}{3}}}\right). \end{aligned} \quad (5)$$

For a sufficiently large  $k_{\text{base}}$ , it is clear that this value can be set arbitrarily close to 1, giving our code a nearly optimal rate.

However, this does not conclude our construction, since in each step of the recursion we assumed that  $\delta_k \leq \exp\left(-\frac{c_2}{2} k^{\frac{1}{3}}\right)$ , but our base construction only produced a code with an arbitrarily small  $\delta_{\text{base}}$  and its relationship to  $k_{\text{base}}$  is unknown.

### 4.3 Completing the Construction

In order to bridge this gap, we apply the recursive step to the base code one more time, with slightly different parameters. We will denote by  $k_0, \delta_0, d_0, t_0, n_0$  the parameters of the first application of the recursive step and by  $k_{\text{base}} = k_0^2, n_{\text{base}}, \delta_{\text{base}}$  the parameters of the resulting code.

As before, we will set  $d = k^{\frac{2}{3}}$ , but unlike the previous setting, since  $\delta_0$  is not necessarily as small as we would want it to be, we will need to set the value of  $t$  to be somewhat larger.

In particular, we will set  $t_0 = \lceil \delta_0^{\frac{c_1}{2}} k_0 \rceil$ . Theorem 6 shows that, for sufficiently small  $\delta_0$ , the probability that a binomial variable with parameters  $[\delta_0, k_0 + 2t_0]$  will be greater than  $t_0$  is of the order of  $\exp(-\Theta(k_0))$ . Therefore, for sufficiently large  $k_0$ , the DFP after the first step of the recursion would be:

$$\delta_{\text{base}} < \exp(-\Theta(k_0)) + \exp\left(-c_2 k_0^{\frac{1}{3}} + o\left(k_0^{\frac{1}{3}}\right)\right) < \exp\left(-\frac{c_2}{2} k_{\text{base}}^{\frac{1}{6}}\right). \quad (6)$$

## 105:14 Efficient Codes for the BDC and PRC Channels

Finally, we need to bound the rate of the entire code. Using Equations (4) and (5), we are able to bound the rate:

$$\begin{aligned} r_{k^2} &< X r_{\text{base}} \leq \exp\left(2 \frac{(k_{\text{base}})^{-\frac{1}{3}}}{1 - (k_{\text{base}})^{-\frac{1}{3}}}\right) r_{\text{base}} \\ &\leq \exp\left(2 \frac{k_0^{-\frac{1}{3}}}{1 - k_0^{-\frac{1}{3}}}\right) \left(1 + \delta_0 \frac{c_1}{2}\right) r_0. \end{aligned} \quad (7)$$

For sufficiently large  $k_0$  and sufficiently small  $\delta_0$ , this can be arbitrarily close to  $r_0$  which in turn can be arbitrarily close to the capacity of the channel, obtaining an arbitrarily close to optimal rate for our code.

### 4.4 Decoding Complexity

The recursive step promises at most  $k + 2t$  calls to the lower level of the construction and  $\tilde{O}(n')$  other operations. Let  $T_{\text{base}}$  be the encoding / decoding complexity of the base scenario, let  $T_k$  be the total complexity of the operation for our code with message length  $k$  and  $I_k$  be the complexity due to operations which are not part of the lower levels of the recursion (i.e. finding the delimiters, separating them from the inner codewords and decoding the Reed Solomon encoding).

Similar to our bound on the rate of the code, we can bound the decoding / encoding complexities by:

$$\begin{aligned} T_{k^2} &= (k + 2t)T_k + I_{k^2} = I_{k^2} + \left(k + 2k^{\frac{2}{3}}\right) T_k \\ &= I_{k^2} + \left(k + 2k^{\frac{2}{3}}\right) I_k + \left(k + 2k^{\frac{2}{3}}\right) \left(\sqrt{k} + 2k^{\frac{1}{3}}\right) T_{\sqrt{k}} \\ &\leq I_{k^2} + \exp\left(2 \frac{k_0^{-\frac{1}{3}}}{1 - k_0^{-\frac{1}{3}}}\right) \frac{k^2}{k} I_k + \left(k + 2k^{\frac{2}{3}}\right) \left(\sqrt{k} + 2k^{\frac{1}{3}}\right) T_{\sqrt{k}} \leq \dots \\ &\dots \leq \exp\left(2 \frac{k_0^{-\frac{1}{3}}}{1 - k_0^{-\frac{1}{3}}}\right) \left(\frac{k^2}{k} I_k + \frac{k^2}{\sqrt{k}} I_{\sqrt{k}} + \dots + \frac{k^2}{k_{\text{base}}} I_{k_{\text{base}}} + \frac{k^2}{k_{\text{base}}} T_{\text{base}}\right). \end{aligned} \quad (8)$$

The recursive step can have at most a quasi-linear complexity on top of its calls to the inner code (see Theorem 8). Therefore,  $I_k = \tilde{O}(n)$ . Inserting this into Equation (8), we are can see that  $T_{k^2} = \tilde{O}(n')$ .

## 5 Adaptation to the Poisson Repeat Channel

In this section we will adapt the construction of our code for the BDC channel detailed in the last two sections, to the PRC channel. Adapting the recursive step will be fairly straightforward. However, adapting the base step will be a bit more tricky.

When working with the BDC, we used the fact that the received message could not be longer than the transmitted one. This allowed us to build a decoding table that can return some value for any of the possible received messages. However, the PRC could (with very low probability) expand a transmitted message to an arbitrarily long received message, and we will need to adapt our construction to address this issue.

## 5.1 Adapting the Recursive Step

► **Theorem 9** (Recursive-Step for the PRC). *There exist some constants  $c_1, c_2, k_0, \delta_0 > 0$ , such that for any  $k > k_0$ ,  $d > 0$ ,  $2^{k-1} - k > t > 0$ ,  $\delta_0 > \delta > 0$ ,  $\lambda > 0$  and any error correcting code  $C$  for the  $PRC_\lambda$  channel with block length  $k$ , message length  $n$  and DFP  $\delta$ , there exists an error correcting code  $C'$  for the  $PRC_\lambda$  with block length  $k^2$ , message length  $n' \leq (k+2t)(n + \frac{d}{\lambda})$  and DFP  $\delta' \leq \Pr[\text{Bin}(\delta^{c_1}, k+2t) > t] + (k+2t) \exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right)$ .*

*Furthermore, there exist an encoder and decoder for  $C'$  with time complexity  $\tilde{O}(n')$  using up to  $k+2t$  calls to the encoder and decoder of  $C$ .*

We will construct the recursive step almost exactly as in Section 3, only changing  $1-p$  to  $\lambda$  in our conversion of lengths of bits over the channel.

## 5.2 Adapting the Base of the Recursion

In this section, we will adapt our construction of the base of our recursion from Section 4 to the PRC. It is easy to see that for any  $\lambda > 0$ , there exists a family of codes for this channel with some non-negligible rate  $\rho = \Theta(1)$  w.r.t the message length (for instance, by applying the jigsaw construction of [19] with the Morse code distribution). We will set  $k_0, \delta_0$  to be the minimal value of the message length  $k$  and DFP  $\delta$  for which there exists such a base code that will suffice for our construction.

Unlike the previous construction, here we will only be able to approximate the DFP of our base code, so we will need to set two bounds. Let  $\kappa_1 \geq k_0$  be the minimal message length for which there is a code in the family of codes such that it has a DFP of at most  $\delta_0$ , and let  $\kappa_2 \geq k_0$  be the minimal message length for which there is a code in the family of codes such that it has a DFP of at most  $\frac{\delta_0}{2}$ . Similar to the construction in Section 4.1, we do not have an explicit bound on  $\kappa_1, \kappa_2$ , but we know that they are bounded by some  $O(1)$  constant.

As in Section 4.1, we will enumerate over values of  $k \geq k_0$ , but this time we only promise that our enumeration will end somewhere between  $\kappa_1$  and  $\kappa_2$ . For each such  $k$ , we enumerate over all  $n \leq \rho k$ , encoders  $C : \{0, 1\}^k \rightarrow \{0, 1\}^n$  and decoders  $D : \{0, 1\}^m \rightarrow \{0, 1\}^k$ , where  $m$  is the smallest integer for which:

$$\Pr[\text{Poisson}(\lambda n) \geq m] \leq \frac{\delta}{2}.$$

For each of these encoder-decoder pairs, we enumerate over all messages in  $\{0, 1\}^k$  and encode them using the encoder. For each codeword, we enumerate over all possible results of applying the channel to the codeword that have output length at most  $m$ .

We sum the probabilities of the eventualities where this process would result in a decoding failure (cases where the channel outputted more than  $m$  bits are counted as failures), and take the message with the highest DFP. This gives us an approximation of

$$\text{DFP}_{\text{actual}} \leq \text{DFP}_{\text{estimate}} \leq \text{DFP}_{\text{actual}} + \frac{\delta}{2}.$$

If the estimated DFP is at most  $\delta$ , then we output the pair of encoder-decoder tables. It is easy to see that if the actual DFP is at most  $\frac{\delta}{2}$  then the estimate DFP is at most  $\delta$  and we will output it. Therefore our process either terminates before  $\kappa_2$  or at  $\kappa_2$  and must have a constant complexity.

## 6 Discussion

In this paper we presented several new techniques for constructing error-correcting-codes for asynchronous channels, and used them to prove a method of transforming lower bounds on the capacities of the BDC and PRC channels to efficient encoding and decoding algorithms. This answers the main question considered [6] and [9].

However, this construction is far from practical. For instance, the inner code at the basis of our recursion is constructed in a doubly exponential time in the size of the inner codes message length. Even our bound on the decoding failure of the inner code is  $\delta^{\frac{1}{34}} = \exp\left(\frac{k^{\frac{1}{6}}}{34}\right)$  which is technically  $o(1)$  but converges extremely slowly.

Furthermore, this work deals only with BDC and PRC channels. While these channels offer us a chance to model the behaviour of asynchronous channels, they do not represent the more realistic channels which contain both synchronisation errors and bit-flipping errors. To this end, the binary InsDel channel offers a more comprehensive model and it remains an open question whether the methods described here can be used to construct efficient codes for it as well.

Finally, now that we have a framework for efficient encoding and decoding for the BDC and PRC channels, we can return to the question of the capacity of these channels. We know that when  $p \rightarrow 1$  this capacity scales proportionally to  $1 - p$ , but the factor of this conversion is still unknown. Mitzenmacher and Drinea [19] showed that these capacities are at least  $\frac{1-p}{9}$ , and [8] gave an upper bound of  $0.4143(1 - p) \pm o_n(1)$ , but this gap is far from closed.

---

## References

- 1 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.
- 2 GS Buller and RJ Collins. Single-photon generation and detection. *Measurement Science and Technology*, 21(1):012002, 2009.
- 3 Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 200–211. IEEE, 2018.
- 4 Mahdi Cheraghchi, Ryan Gabrys, Olgica Milenkovic, and Joao Ribeiro. Coded trace reconstruction. *IEEE Transactions on Information Theory*, 66(10):6084–6103, 2020.
- 5 Mahdi Cheraghchi and João Ribeiro. An overview of capacity results for synchronization channels. *IEEE Transactions on Information Theory*, 67(6):3207–3232, 2020.
- 6 Roni Con and Amir Shpilka. Explicit and efficient constructions of coding schemes for the binary deletion channel and the poisson repeat channel. *arXiv preprint*, 2019. [arXiv:1909.10177](https://arxiv.org/abs/1909.10177).
- 7 Roni Con, Amir Shpilka, and Itzhak Tamo. Linear and reed solomon codes against adversarial insertions and deletions. *arXiv preprint*, 2021. [arXiv:2107.05699](https://arxiv.org/abs/2107.05699).
- 8 Marco Dalai. A new bound on the capacity of the binary deletion channel with high deletion probabilities. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 499–502. IEEE, 2011.
- 9 Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 65(4):2171–2178, 2018.
- 10 Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.
- 11 Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 334–347. IEEE, 2019.
- 12 Bernhard Haeupler, Aviad Rubinfeld, and Amirbehshad Shahrabi. Near-linear time insertion-deletion codes and  $(1 + \epsilon)$ -approximating edit distance via indexing. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 697–708, 2019.



- 13 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 33–46, 2017.
- 14 Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.
- 15 Adam Kalai, Michael Mitzenmacher, and Madhu Sudan. Tight asymptotic bounds for the deletion channel with small deletion probabilities. In *2010 IEEE International Symposium on Information Theory*, pages 997–1001. IEEE, 2010.
- 16 Yonglong Li and Vincent YF Tan. On the capacity of channels with deletions and states. *IEEE Transactions on Information Theory*, 67(5):2663–2679, 2020.
- 17 Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials*, 12(1):87–96, 2010.
- 18 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.
- 19 Michael Mitzenmacher and Eleni Drinea. A simple lower bound for the capacity of the deletion channel. *IEEE Transactions on Information Theory*, 52(10):4657–4660, 2006.
- 20 Henry D Pfister and Ido Tal. Polar codes for channels with insertions, deletions, and substitutions. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 2554–2559. IEEE, 2021.
- 21 Ittai Rubinstein. Explicit and efficient construction of (nearly) optimal rate codes for binary deletion channel and the poisson repeat channel. *arXiv preprint*, 2021. [arXiv:2111.00261](https://arxiv.org/abs/2111.00261).
- 22 Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- 23 Ido Tal, Henry D Pfister, Arman Fazeli, and Alexander Vardy. Polar codes for the deletion channel: Weak and strong polarization. *IEEE Transactions on Information Theory*, 2021.
- 24 Scott A Vanstone and Paul C Van Oorschot. *An introduction to error correcting codes with applications*, volume 71. Springer Science & Business Media, 2013.
- 25 Ramji Venkataramanan, Sekhar Tatikonda, and Kannan Ramchandran. Achievable rates for channels with deletions and insertions. *IEEE transactions on information theory*, 59(11):6990–7013, 2013.