

# Improved Upper and Lower Bounds on the Capacity of the Binary Deletion Channel

Ittai Rubinstein<sup>\*1</sup> and Roni Con<sup>†2</sup>

<sup>1</sup>Qedma Quantum Computing, Tel-Aviv, Israel

<sup>2</sup>Blavatnik School of Computer Science, Tel Aviv University, Tel-Aviv, Israel

May 15, 2023

## Abstract

The *binary deletion channel* with deletion probability  $d$  ( $\text{BDC}_d$ ) is a random channel that deletes each bit of the input message i.i.d with probability  $d$ . It has been studied extensively as a canonical example of a channel with synchronization errors [Mit09, MBT10, CR20].

Perhaps the most important question regarding the BDC is determining its capacity. Mitzenmacher and Drinea [MD06] and Kirsch and Drinea [KD09] show a method by which distributions on run lengths can be converted to codes for the BDC, yielding a lower bound of  $\mathcal{C}(\text{BDC}_d) > 0.1185 \cdot (1 - d)$ . Fertoni and Duman [FD10], Dalai [Dal11] and Rahmati and Duman [RD14] use computer aided analyses based on the Blahut-Arimoto algorithm to prove an upper bound of  $\mathcal{C}(\text{BDC}_d) < 0.4143 \cdot (1 - d)$  in the high deletion probability regime ( $d > 0.65$ ).

In this paper, we show that the Blahut-Arimoto algorithm can be implemented with a lower space complexity, allowing us to extend the upper bound analyses, and prove an upper bound of  $\mathcal{C}(\text{BDC}_d) < 0.3745 \cdot (1 - d)$  for all  $d \geq 0.68$ . Furthermore, we show that an extension of the Blahut-Arimoto algorithm can be used to select better run length distributions for Mitzenmacher and Drinea's construction, yielding a lower bound of  $\mathcal{C}(\text{BDC}_d) > 0.1221 \cdot (1 - d)$ .

---

\*ittai.rubinstein@gmail.com

†roni.con93@gmail.com

The work of Roni Con was partially supported by the European Research Council (ERC grant number 852953) and by the Israel Science Foundation (ISF grant number 1030/15).

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>3</b>  |
| 1.1      | Previous Work . . . . .  | 4         |
| 1.2      | Our results . . . . .  | 4         |
| 1.3      | Organization . . . . .   | 5         |
| <b>2</b> | <b>Upper Bound Theory</b>  | <b>6</b>  |
| 2.1      | Discrete Memoryless Channels and the Blahut-Arimoto Algorithm . . . .  | 6         |
| 2.2      | Auxiliary Deletion Channels . . . . .                                  | 8         |
| <b>3</b> | <b>Memory Efficient Implementation of the Blahut-Arimoto Algorithm</b> | <b>9</b>  |
| 3.1      | Caching and Loop Nest Optimization . . . . .                           | 10        |
| 3.2      | Using sparsity . . . . .   | 12        |
| 3.3      | Results . . . . .  | 14        |
| <b>4</b> | <b>A new Lower Bound</b>   | <b>14</b> |
| 4.1      | The framework of Drinea and Mitzenmacher . . . . .                     | 14        |
| 4.2      | Our heuristic approach to find better input distributions . . . . .    | 17        |
| 4.3      | Analysis of the heuristic . . . . .                                    | 18        |
| 4.4      | Reduction to Convex Optimization . . . . .                             | 21        |

# 1 Introduction

In this paper we focus on the *binary deletion channel* (BDC) which deletes each bit from the input message randomly and independently with a given *deletion probability*  $d$ . Loosely speaking, a channel is a medium over which messages are sent. A channel is defined by the way in which it introduces errors to the transmitted messages (also called codewords when they come from an error correcting code).

Two of the most well-studied channels are the Binary Erasure Channel ( $\text{BEC}_d$ ) where each bit is independently replaced by a question mark with probability  $d$  and the Binary Symmetric Channel ( $\text{BSC}_d$ ) where each bit is independently flipped with probability  $d$ . We note that the BDC is very different from the BEC and the BSC. For example, consider the case where the transmitted message was 1110101 and corruptions occurred in locations 2 and 5. In this scenario, the BEC will return the word 1?10?01 and the BSC will return 1010001, while the BDC would return 11001.

In other words, while the BEC and the BSC may corrupt some bits in the message, the BDC can change the length of a codeword and the index in which a bit may appear. This makes the BDC a “synchronization channel”, and significantly complicates its analysis. In fact, one of the main reasons for introducing the BDC was to model synchronization errors in communication. More recently, codes correcting from deletions found applications in a variety of fields such as computational biology and DNA storage [BLC<sup>+</sup>16, YGM17, HMG19]. For a more detailed review of synchronization channels and their applications, we refer the interested reader to the excellent surveys by Mitzenmacher [Mit09], Mercier et al. [MBT10], and Cheraghchi and Ribeiro [CR20].

To explain the question that we study we need some basic notions from coding theory. Recall that a binary error correcting code can be described either as an encoding map  $C : \{0, 1\}^k \rightarrow \{0, 1\}^n$  or, abusing notation, as the image of such a map  $C$ . The rate of such a code  $C$  is  $\text{Rate}(C) = k/n$ , which intuitively captures the amount of information encoded in every bit of a codeword. Naturally, we would like the rate to be as large as possible, but there is a tension between the rate of the code and the amount of errors/noise it can tolerate.

One of the most fundamental questions when studying a channel is to determine its capacity, i.e., the maximum achievable transmission rate over the channel that still allows recovering from the errors introduced by the channel, with high probability. Shannon proved in his seminal work [Sha48] that the capacity of the  $\text{BSC}_d$  is  $1 - h(d)$ , where  $h(\cdot)$  is the binary entropy function (for  $0 < x < 1$ ,  $h(x) = -x \log x - (1 - x) \log 1 - x$ ).<sup>1</sup> Elias [Eli55], who introduced the  $\text{BEC}_d$ , proved that its capacity is  $1 - d$ .

What about the capacity of the  $\text{BDC}_d$ ? In spite of significant efforts by many researchers, much less is known about the capacity of the BDC. This is because the asynchronous nature of the BDC which makes it interesting also makes it harder to analyse.

In the extremal parameter regimes, the behavior of the capacity of the BDC is partially understood. When  $d \rightarrow 0$  the capacity approaches  $1 - h(d)$  [KMS10]. For all  $d \in (0, 1)$ , the capacity is bounded from below by  $\mathcal{C}(\text{BDC}_d) > 0.1185 \cdot (1 - d)$  [MD06], and when  $d > 0.65$ , it is bounded from above by  $\mathcal{C}(\text{BDC}_d) < 0.4143 \cdot (1 - d)$  [FD10, Dal11, RD14], giving us the asymptotic scaling as  $d \rightarrow 1$ . For a more detailed picture of the known bounds on the capacity of the BDC, see Figure 1.

---

<sup>1</sup>All logarithms in this paper are base 2.

## 1.1 Previous Work

Our focus in this paper is in bounding the capacity of the BDC in either the bulk of the parameter regime  $d \in (\varepsilon, 1 - \varepsilon)$ , or in the high-deletion probability regime  $d \rightarrow 1$ . We will list here the best known results for these regimes.

**Lower bounds** The best known lower bound on the capacity is due to Mitzenmacher and Drinea [MD06, DM07] who showed a lower bound of  $0.1185 \cdot (1 - d)$  for all  $d$ , meaning that there are codes of this rate such that every transmitted codeword is decoded correctly with high probability. This lower bound is the best known lower bound for the high deletion probability regime ( $d \geq 0.95$ ). For smaller values of  $d$ , Drinea and Mitzenmacher prove stronger bounds in [DM07, Table 1]. Their proof is constructive, but it does not directly yield an efficient decoding algorithm for the family of codes they construct.

Since then, several constructions of efficiently decodable codes for the BDC have been published [GL18, CS22], culminating in several constructions of efficiently decodable codes that achieve capacity [TPFV21, Rub22, PLW22]. In particular, [Rub22] presents a method for converting any code for the BDC to an efficiently decodable one with an arbitrarily close rate. Therefore, the codes generated using Mitzenmacher and Drinea’s construction can be converted to explicit and efficient constructions of high rate codes for the BDC.

**Upper bounds** Fertoni and Duman [FD10] proved several upper bounds on the capacity of the BDC. They do this by providing the transmitter and the receiver with “hints” about the noise of the channel. Adding these hints only increases the information rate, allowing them to bound the capacity of the BDC from above by bounding the capacity of some auxiliary channels using the Blahut-Arimoto algorithm.

Dalai [Dal11] and Rahmati and Duman [RD14] refine this analysis and prove that for any  $d \in (0.65, 1)$ , the capacity of the BDC is bounded from above by  $\mathcal{C}(\text{BDC}_d) \leq 0.4143(1 - d)$ . Given unlimited computational resources, these methods will converge to the capacity of the channel, but this convergence is extremely slow.

## 1.2 Our results

In this work, we improve both the upper and the lower bounds on the capacity of the BDC. To improve the upper bound, we integrate into the classical Blahut-Arimoto algorithm several ingredients that reduce its memory requirements. These ingredients include loop nest optimization, caching, symmetries, and compressing sparse matrices. This allows us to extend the previous analyses to more computationally challenging regimes and gives us the following new upper bound

**Theorem 1.1.** *For any  $d \geq 0.68$ , it holds that*

$$\mathcal{C}(\text{BDC}_d) \leq 0.3745 \cdot (1 - d) .$$

Our second contribution is an improved lower bound on the capacity of the BDC based on Mitzenmacher and Drinea’s construction [DM07]. Mitzenmacher and Drinea’s construction is parametrized by a distribution on “run lengths” and outputs a provable

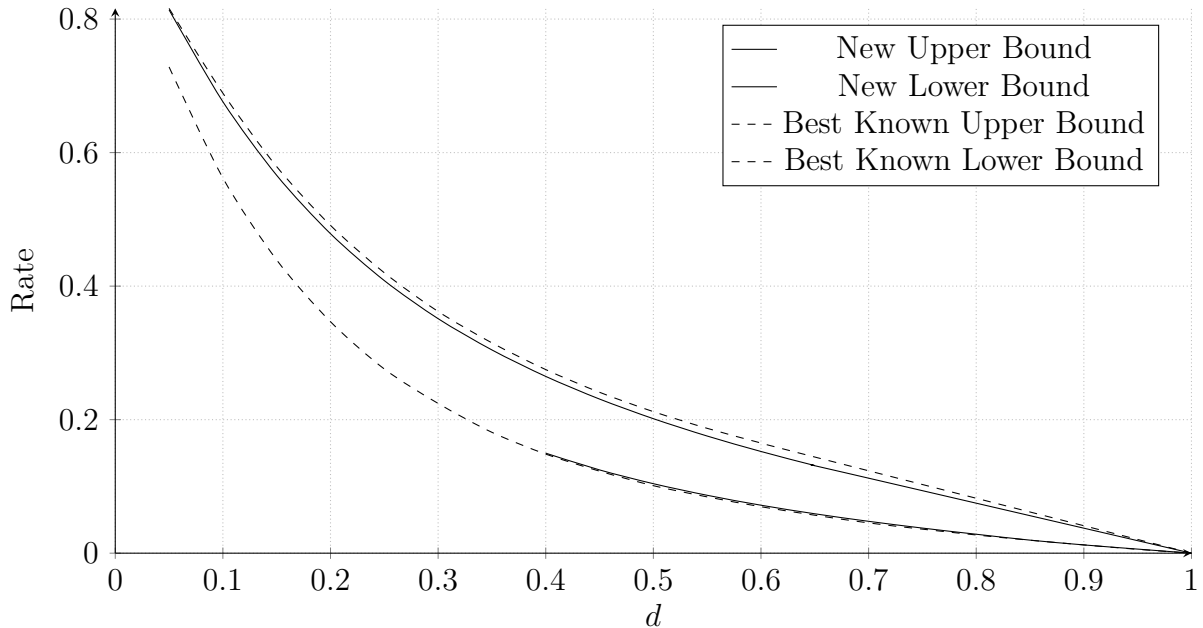


Figure 1: Our new upper and lower bounds compared to previous state-of-the-art upper and lower bounds. The best known lower bound is given in [DM07, MD06] and the best known upper bounds are taken from [DMP07, FD10, Dal11, RD14]

lower bound on the capacity of the BDC. Mitzenmacher and Drinea apply their construction to various geometric run length distributions, but give no argument for why geometric distributions should be optimal. We propose a heuristic approach to selecting better run length distributions to be used as parameters for Mitzenmacher and Drinea’s rigorous analysis, yielding an improved lower bound (Theorem 1.2).

**Theorem 1.2.** *For any  $d \in (0, 1)$*

$$\mathcal{C}(\text{BDC}_d) > 0.1221 \cdot (1 - d) .$$

Using the result of [Rub22] as a black box, we can efficiently construct a binary code that achieves this rate and has efficient encoding and decoding algorithms.

For a full characterization of the lower and upper bounds on the capacity of the BDC as a function of  $d$ , see Tables 1 and 2, and Figure 1.

### 1.3 Organization

In Section 2, we describe the main components of the Blahut-Arimoto algorithm and the best known upper bounds. In Section 3, we construct a memory efficient version of the Blahut-Arimoto algorithm and show that it can be used to prove tighter upper bounds on the capacity of the BDC. Finally, in Section 4 we present a heuristic approach to optimizing the input to Mitzenmacher and Drinea’s construction, allowing us to prove tighter lower bounds on the capacity of the BDC.

In the interest of reproducibility, we have made our code publicly available through github. The code used to generate the upper bounds in Sections 2 and 3 is available here. The code used to generate our lower bounds (see Section 4) is available here.

## 2 Upper Bound Theory

In this section, we will give an overview of the theory involved in our upper bound analysis. In Section 2.1, we will introduce discrete memoryless channels and the Blahut-Arimoto algorithm which can be used to compute their capacity. Then, in Section 2.2, we will introduce several auxiliary channels which will aid our analysis of the BDC. We will show that these auxiliary channels are DMCs, allowing us to use the Blahut-Arimoto algorithm to find their capacity, and that their capacities are related to those of the BDC.

### 2.1 Discrete Memoryless Channels and the Blahut-Arimoto Algorithm

Our main results are achieved through improvements and extensions of the Blahut-Arimoto algorithm for finding the capacity of discrete memoryless channels. In this section, we will give a brief overview of these channels and the Blahut-Arimoto algorithm (for a more detailed review, see [Yeu08, Chapter 9]).

A *discrete memoryless channel* (DMC) is defined by a finite input alphabet  $\mathcal{X}$ , a finite output alphabet  $\mathcal{Y}$ , and a transition probability matrix  $P \in \mathbb{R}_+^{|\mathcal{X}| \times |\mathcal{Y}|}$ . The channel sends each input letter  $x$  to each output letter  $y$  with probability  $P_{x \rightarrow y}$  independently at random. We note that the BDC is not a memoryless channel, so the results discussed here cannot be used to bound its capacity directly. In Section 2.2, we will show that there exists a family of DMCs whose capacities converge on the capacity of the BDC from above.

In general, the capacity  $\mathcal{C}$  of a channel represents the maximum rate at which information can be communicated reliably over it [Yeu08]. For DMCs, the capacity is given by

$$\mathcal{C} = \max_{\substack{\mathcal{P} \\ x \leftarrow \mathcal{P}}} \{I(x; y)\} = \sup_{\mathcal{P} \in \mathbb{R}_+^{\mathcal{X}}} \left\{ \sum_{x,y} \mathcal{P}_x P_{x \rightarrow y} \log \left( \frac{P_{x \rightarrow y}}{\sum_{x'} \mathcal{P}_{x'} P_{x' \rightarrow y}} \right) \right\} \quad (2.1)$$

where the supremum is taken over input distributions  $\mathcal{P}(x)$  that have a nonzero probability to output any letter  $x \in \mathcal{X}$ . Both the input and the output alphabets are of finite size, so the information rate associated with any given input distribution  $\mathcal{P}$  can be computed using the right-hand-side of eq. (2.1). However, it is not obvious that one can efficiently find a distribution  $\mathcal{P}$  that maximizes this formula.

The *Blahut-Arimoto algorithm* (BAA - see Algorithm 1) is an iterative algorithm which rapidly converges to the capacity for any given DMC. It was introduced independently by Blahut and Arimoto in [Bla72, Ari72], who showed that it can be used to quickly find an input distribution whose rate is arbitrarily close to optimal.

We denote by  $\mathcal{P}^{(\ell)}$  the input distribution used in the  $\ell$ th iteration, and by  $\mathcal{Q}_{y,x}^{(\ell)}$  the transition probabilities from the output alphabet to the input alphabet which can be thought of as the transitions of a probabilistic decoding algorithm. The algorithm runs by performing an alternating maximization on  $\mathcal{P}$  and  $\mathcal{Q}$ .

**Claim 2.1.** [Ari72, Corollary 1] *Let  $a > 0$ . The BAA algorithm converges within  $\mathcal{O}(1/a)$  iterations, and returns a distribution  $\mathcal{P}$  with an information rate of at least  $\mathcal{R} \geq \mathcal{C}(\text{Ch}) - a$ .*

---

**Algorithm 1:** The Blahut-Arimoto algorithm.

---

**input :** Input and output alphabets  $\mathcal{X}, \mathcal{Y}$

$P : [0, 1]^{\mathcal{X} \times \mathcal{Y}}$  - the channel's transition probability matrix

$a$  - capacity approximation parameter

**output:**  $\mathcal{R} \in \mathbb{R}$  - an upper bound on the capacity of this channel

$\mathcal{P} \in \mathbb{R}^{\mathcal{X}}$  - an input distribution with a nearly optimal rate for this channel

[1] Set  $\ell = 0$  and set  $\mathcal{P}^{(1)}$  to be the uniform distribution.

[2] **do**

1.  $\ell \leftarrow \ell + 1$

2. For every  $x \in \mathcal{X}$  and  $y \in \mathcal{Y}$  compute

$$\mathcal{Q}_{y,x}^{(\ell)} = \frac{\mathcal{P}^{(\ell)}(x)P_{x \rightarrow y}}{\sum_{x'} \mathcal{P}^{(\ell)}(x')P_{x' \rightarrow y}}$$

3. For every  $x \in \mathcal{X}$ , compute

$$\mathcal{P}^{(\ell+1)}(x) = \frac{\prod_y (\mathcal{Q}_{y,x}^{(\ell)})^{P_{x \rightarrow y}}}{\sum_{x'} \prod_y (\mathcal{Q}_{y,x'}^{(\ell)})^{P_{x' \rightarrow y}}}$$

where the product is over all  $y$  such that  $\mathcal{Q}_{y,x}^{(\ell)} > 0$  and  $P_{x \rightarrow y} > 0$ .

**while**  $\max_x \log_2 \left( \frac{\mathcal{P}^{(\ell+1)}(x)}{\mathcal{P}^{(\ell)}(x)} \right) \geq a$

[3] **Compute**

$$\mathcal{R} = \sum_x \sum_y \mathcal{P}^{(\ell)}(x) \cdot P_{x \rightarrow y} \cdot \log \left( \frac{\mathcal{Q}_{y,x}^{(\ell)}}{\mathcal{P}^{(\ell)}(x)} \right)$$

where the sums are over all  $x, y$  such that  $\mathcal{Q}_{y,x}^{(\ell)} > 0$  and  $P_{x \rightarrow y} > 0$ .

[4] **Return**  $\mathcal{P}^{(\ell+1)}(x), \mathcal{R}$ .

---

## 2.2 Auxiliary Deletion Channels

In Section 2.1, we defined discrete memoryless channels and presented the Blahut-Arimoto algorithm which can be used to bound their capacities. In this section, we will present several auxiliary channels. Originally introduced by Fertoni and Duman [FD10] and also used by Dalai [Dal11], these auxiliary channels are both discrete memoryless channels, allowing us to bound their capacities with the Blahut-Arimoto algorithm, and related to the BDC, allowing us to use them to bound the capacity of the BDC.

Let  $W_n^d$  be the binary deletion channel with fixed input length  $n$ . Denote by  $\mathcal{C}_n(d) := \max I(X_1^n; Y)$  its capacity where the maximum is taken over all input distributions on  $n$  bits, and  $Y = W_n^d(X_1^n)$  (i.e., the output of the channel on input  $X$ ).

**Claim 2.2.** [Dal11, Lemma 1] For any  $n \in \mathbb{N}$  and  $d \in [0, 1]$ ,

$$\mathcal{C}(BDC_d) \leq \frac{1}{n} \cdot \mathcal{C}_n(d).$$

Similarly, we denote by  $W_{n,k}$  the channel with  $n$ -bit input whose output is uniformly chosen within the  $\binom{n}{k}$   $k$ -bit subsequences of the input (i.e. the channel deletes  $n - k$  of the bits at random). Denote its capacity by  $\mathcal{C}_{n,k} := \max I(X_1^n; Y)$  where  $Y = W_{n,k}(X_1^n)$ .

Fertonani and Duman, and then Dalai showed that the values of  $\mathcal{C}_{n,k}$  can be used to bound  $\mathcal{C}_n(d)$  from above with the following inequality.

$$\mathcal{C}_n(d) \leq \sum_{k=1}^n \binom{n}{k} d^{n-k} (1-d)^k \cdot \mathcal{C}_{n,k} \quad (2.2)$$

*Proof of (2.2).* The proof of this inequality is contained in the proof Lemma 4 in [Dal11]. We repeat the details for completeness. Let  $X_1^n$  be an optimal distribution for  $W_n^d$  and let  $Y = W_n^d(X_1^n)$  be the output distribution of the channel on  $X_1^n$ . Define the random variable  $L = |Y|$  (the length of  $Y$ ). Let  $S_d(n, k)$  be the probability that a string of length  $n$  is transformed into a string of length  $k$  after being transmitted through  $W_n^d$ , namely,

$$S_d(n, k) = \binom{n}{k} d^{n-k} (1-d)^k.$$

Then,

$$\mathcal{C}_n(d) = I(X_1^n; Y|L) = \sum_{k=0}^n S_d(n, k) \cdot I(X_1^n; Y|L = k) \leq \sum_{k=1}^n S_d(n, k) \cdot \mathcal{C}_{n,k}$$

Where the first equality is due to the fact that  $X \rightarrow Y \rightarrow L$  is a Markov chain. The inequality follows since  $I(X_1^n; Y|L = k) \leq \mathcal{C}_{n,k}$  and  $\mathcal{C}_{n,0} = 0$ .  $\square$

Clearly,  $\mathcal{C}_{n,k}$  describes the capacity of a channel with a finite input and a finite output size. Therefore, we can describe it with a transition matrix  $P \in \mathbb{R}_+^{2^n \times 2^k}$  and use the BAA algorithm to compute its capacity to within a given additive error. In [FD10], Fertoni and Duman, bounded  $\mathcal{C}_{n,k}$  for all  $1 \leq k \leq n \leq 17$ , and for the specific case where  $k = n - 1$  up to  $n = 22$ . However, they did not manage to go beyond these parameters due to the high computational complexity required [FD10]. In particular, we note the high memory



complexity of the BAA which is the main bottleneck in applying it to channels with large alphabets.

In Section 3, we will discuss several ways of decreasing the space complexity of the naïve BAA implementation (Algorithm 1). Before that, we present several properties of  $\mathcal{C}_{n,k}$  that we will use when computing the upper bound on  $\mathcal{C}_n(d)$ .

**Lemma 2.3.** [FD10, Lemma 1] *For all  $n$  and  $k$ , it holds that  $\mathcal{C}_{n+1,k} \leq \mathcal{C}_{n,k}$ .*

**Lemma 2.4.** [FD10, Lemma 3] *For all  $n$  and  $k \geq 1$ , it holds that*

$$\mathcal{C}_{n+1,k} \leq \mathcal{C}_{n,k-1} \cdot \left(1 - \frac{k}{n+1}\right) + (\mathcal{C}_{n,k} + 1) \cdot \frac{k}{n+1}.$$

The following claim can be seen as a straight forward generalization of Lemma 2.4.

**Lemma 2.5.** *For every  $s \in [n]$ , it holds that*

$$\mathcal{C}_{n,k} \leq \sum_{i=0}^s \frac{\binom{s}{i} \cdot \binom{n-s}{k-i}}{\binom{n}{k}} \cdot (\mathcal{C}_{s,i} + \mathcal{C}_{n-s,k-i})$$

*Proof.* Let  $X_1^n$  be an optimal distribution for the channel  $W_{n,k}$ . Denote by  $R_s$  the random variable that corresponds to the number of bits that survived the transmission of the last  $s$  bits of  $X$ . It holds that

$$\mathcal{C}_{n,k} = I(X_1^n; Y) \leq I(X_1^n; Y | R_s) = \sum_{i=0}^s \Pr[R_s = i] \cdot I(X_1^n; Y | R_s = i)$$

where the inequality follows since  $R_s$  and  $X_1^n$  are independent. Now,  $\Pr[R_s = i] = \frac{\binom{s}{i} \cdot \binom{n-s}{k-i}}{\binom{n}{k}}$  so we are left to show that  $I(X_1^n; Y | R_s = i) \leq \mathcal{C}_{s,i} + \mathcal{C}_{n-s,k-i}$ . Indeed, let  $Z^{(0)} := W_{n-s,k-i}(X_1^{n-s})$ ,  $Z^{(1)} := W_{s,i}(X_{n-s+1}^n)$ , and consider the following markov chain  $X_1^n \rightarrow (Z^{(0)}, Z^{(1)}) \rightarrow Z$  where the second step is just the concatenation of  $Z^{(0)}$  and  $Z^{(1)}$ . Thus, by the data processing inequality,

$$I(X_1^n; Y | R_s = i) = I(X_1^n; Z) \leq I(X_1^n; (Z^{(0)}, Z^{(1)})) \leq \mathcal{C}_{s,i} + \mathcal{C}_{n-s,k-i}. \quad \square$$

### 3 Memory Efficient Implementation of the Blahut-Arimoto Algorithm

Our ultimate goal is to run the Blahut-Arimoto algorithm on these auxiliary channels with larger input and output alphabets in order to find tighter upper bounds on the capacity of the BDC.

The main bottleneck in applying Algorithm 1 to larger input alphabets ( $\mathcal{X}$ ) and output alphabets ( $\mathcal{Y}$ ), is its memory requirement. Recall that Algorithm 1 stores two  $|\mathcal{X}| \times |\mathcal{Y}|$ -sized arrays: the transition probability matrix of the channel and the intermediate array  $\mathcal{Q}_{y,x}^{(\ell)}$  computed in each iteration. This requires us to store and frequently access large arrays of floating point numbers, which increases the memory complexity of the algorithm, and poses the main technical limitation on our ability to scale to larger alphabets.

To get a sense of how quickly the memory usage of Algorithm 1 becomes impractical, assume that we want to compute upper bound on  $\mathcal{C}_{25,12}$  using the BAA algorithm (Algorithm 1). In this case,  $|\mathcal{X}| = 2^{25}$ ,  $|\mathcal{Y}| = 2^{12}$  and saving just the transition matrix,  $P$ , where each entry is represented by a *double* type floating point, would require a terabyte of RAM storage!

In the following section, Section 3.1, we use a combination of caching computation results and loop nest optimization to significantly reduce the asymptotic memory requirements of the BAA algorithm, albeit with a small increase in runtime. In Section 3.2, we present a “sparse” version of Algorithm 1 that enables more efficient storage of the arrays. By employing these techniques, we were able to compute upper bounds on  $\mathcal{C}_{n,k}$  for values of  $n$  and  $k$  that were unknown prior to this work. Finally, in Section 3.3, we present our computation which takes as input the upper bounds on  $\mathcal{C}_{n,k}$  and produces the improved upper bounds on the capacity of the BDC.

### 3.1 Caching and Loop Nest Optimization

The main method we use to compute the bulk of our  $\mathcal{C}_{n,k}$  capacity bounds is by removing the requirement for storing the  $|\mathcal{X} \times \mathcal{Y}|$  sized matrices  $P_{x \rightarrow y}$ ,  $Q_{y,x}$  altogether, and replacing them with asymptotically smaller arrays instead.

We can remove the memory requirements for the transition probability matrix from the Blahut-Arimoto algorithm by accepting an oracle access to its entries instead. Doing this requires a method of computing the channel’s transition probabilities quickly. The transition probabilities are proportional to the number of ways in which a given output string can be represented as a subsequence of a given input string, and the common approach to this computation would be to use the dynamic programming method shown in Algorithm 2.

However, Algorithm 2 runs in time  $\Theta(n \cdot k)$ , which proved to be too slow in practice. Therefore, we used a slightly different approach shown in Algorithm 3 which runs in time  $O(k)$ , but requires a precomputed cache table of size  $\Theta(2^{k+\lceil n/2 \rceil})$  that are computed using Algorithm 2.

Using large caches might seem counter-intuitive when our goal is to reduce the memory complexity of an algorithm, but this added memory is far smaller than the original memory complexity. For instance, in the numerical example given above with  $n = 25$  and  $k = 12$ , this precomputed table would take up only about 256MB of RAM, making it much more practical.

To avoid storing the intermediate array  $Q_{y,x}^{(\ell)}$ , we use an algorithmic concept called loop nest optimization [Wol92], meaning that we alter the order in which the operations of the algorithm are performed, in order to reduce its memory storage and retrieval requirements. We perform this optimization to construct Algorithm 4, where we avoid computing the entries of  $Q_{y,x}^{(\ell)}$ . Instead, we compute the same outputs of each iteration of the Blahut-Arimoto algorithm using only the  $\mathcal{O}(|\mathcal{X}| + |\mathcal{Y}|)$  sized intermediate arrays  $\mathcal{D}_y^{(\ell)}$ ,  $\mathcal{W}_x^{(\ell)}$ .

Since each step of the Blahut-Arimoto optimization is logically unchanged, Algorithm 4 converges with exactly the same number of iterations as Algorithm 1. In both algorithms, each iteration has  $\Theta(|\mathcal{X} \times \mathcal{Y}|)$  arithmetic operations and Algorithm 4 also requires  $\Theta(|\mathcal{X} \times \mathcal{Y}|)$  queries to the transition probability oracle, each of which takes

---

**Algorithm 2:** A dynamic programming algorithm for computing the transition probabilities of the  $\mathcal{C}_{n,k}$  channel.

This algorithm has time and memory complexity  $\Theta(nk)$ .

---

**input :** Input string  $x \in \mathcal{X} \in \{0, 1\}^n$

Output string  $y \in \mathcal{Y} \in \{0, 1\}^k$

**output:** Transition probability  $P_{x \rightarrow y}$

[1] Initialize a 2D array  $A$  of size  $(n + 1) \times (k + 1)$  with all values as 0

**for**  $i$  from 0 to  $n$  **do**

**for**  $j$  from 0 to  $k$  **do**

**if**  $i = 0$  and  $j = 0$  **then**

            Set  $A[i][j] := 1$

**end**

**if**  $i > 0$  and  $j = 0$  **then**

            Set  $A[i][j] := A[i - 1][j]$

**end**

**if**  $i > 0$  and  $j > 0$  and  $x_i = y_j$  **then**

            Set  $A[i][j] := A[i - 1][j - 1] + A[i - 1][j]$

**end**

**if**  $i > 0$  and  $j > 0$  and  $x_i \neq y_j$  **then**

            Set  $A[i][j] := A[i - 1][j]$

**end**

**end**

**end**

[2] Return

$$\frac{A[n][k]}{\binom{n}{k}}$$


---

---

**Algorithm 3:** An algorithm for computing the transition probabilities of the  $\mathcal{C}_{n,k}$  channel.

---

**input :** Input string  $x \in \mathcal{X} = \{0, 1\}^n$

Output string  $y \in \mathcal{Y} = \{0, 1\}^k$

Cache table  $P_{x' \rightarrow y'}$  of all transition probabilities of  $\mathcal{C}_{\leq \lceil n/2 \rceil, \leq k}$  channels

**output:** Transition probability  $P_{x \rightarrow y}$

[1] Set  $n_1 = \lceil \frac{n}{2} \rceil$  and  $n_2 = \lfloor \frac{n}{2} \rfloor$ .

[2] Define  $x_1 = x_{:n_1}$  and  $x_2 = x_{n_1+1:}$ : to be the first  $n_1$  and the last  $n_2$  bits in  $x$  (resp).

[3] Return

$$P_{x \rightarrow y} = \sum_{0 \leq k' \leq k} P_{x_1 \rightarrow y_{:k'}} P_{x_2 \rightarrow y_{k'+1:}}$$


---

$\mathcal{O}(k)$  time (when implemented using Algorithm 3). Therefore, the total runtime of our optimized BAA Algorithm 4 is  $\mathcal{O}(k) \cdot \Theta(|\mathcal{X} \times \mathcal{Y}|) \cdot \mathcal{O}(1/a) = \mathcal{O}\left(k \cdot \frac{2^{n+k}}{a}\right)$ .

Using Algorithm 4, we computed bounds on  $\mathcal{C}_{n,k}$  for all  $n + k \leq 39$  where  $n \in [28]$ .

### 3.2 Using sparsity

In this section, we will use the fact that when  $k$  is close to  $n$ , the matrices  $P$  and  $\mathcal{Q}_{y,x}^{(\ell)}$  are (very) sparse and thus can be represented in a compressed form. Recall that a major bottleneck in Algorithm 1 is the memory usage that it requires. Consider for example the case where  $k = n - 1$ . In this case, the size of  $P$  is  $2^{2n-1}$  but each row of  $P$  has at most  $n$  nonzero values which implies that each row contains at least  $2^{n-1} - n$  zeros. Also, recall that Algorithm 1 does not consider zero values in its computations. This motivates us to try to modify Algorithm 1 and create a version that saves only the nonzero entries.

There are many ways of representing sparse matrices. We shall use the compressed sparse column format (CSC), in which the sparse matrix is saved using three (one dimensional) arrays: the *data* array, the *col-ind* array and the *row-ind* array. The data array will store all the nonzero values of the sparse matrix and the row-ind will store the corresponding row indices of these values, respectively. The col-ind at index  $j$  will encode the total number of nonzero values before the  $j$ th column (the size of col-ind is the number of columns in the matrix plus 1). Another representation that we will use is the compressed sparse row format (CSR), that is similar to the CSC format. Here, the col-ind and row-ind array change roles. The col-ind will store the column indices of the data and the row ind at index  $j$  will encode the total number of nonzero values above the  $j$ th row.

In our implementation, we extend the CSC representation by incorporating two additional arrays to compress the row data in a manner similar to the CSR format. The first of these arrays, called *perm-data*, stores the permutation that maps the data array of the CSC matrix to the data array of a CSR matrix. The second array, called *row-ind-CSR*, represents the row indices of the corresponding CSR matrix.

The modifications to Algorithm 1 are:

- The input to the algorithm is now the channel transition matrix  $P$  in the CSC format.
- Prior to the loop in Step 2, we compute the two additional arrays discussed above: the perm-data array and row-ind-CSR array.
- In Step 2, we loop over only the nonzero values of  $P$  and  $\mathcal{Q}^{(\ell)}y, x$  to compute  $\mathcal{Q}^{(\ell)}y, x$  and  $\mathcal{P}^{(\ell+1)}(x)$ . The extension arrays we added allow us to iterate over the rows and columns of the sparse matrix.

Another major advantage of the modifications is the significantly improved runtime of the algorithm. By looping over just the non-zero values of  $\mathcal{Q}_{y,x}^{(\ell)}$  and  $\mathcal{P}^{(\ell+1)}$ , the number of arithmetic operations becomes linear in the sparsity order, rather than linear in the size of the matrices. Specifically, the runtime of the sparse BAA algorithm is  $\mathcal{O}(1/a) \cdot \mathcal{O}\left(\binom{n}{k} \cdot 2^n\right) = \mathcal{O}\left(\frac{n^k \cdot 2^n}{a}\right)$ .

---

**Algorithm 4:** A loop nest optimized implementation of the Blahut-Arimoto algorithm.

Note that this algorithm contains arithmetic operations over real numbers, which could lead to undefined intermediate values (e.g., due to a division by 0). We work with the convention that the outputs of such operations are fixed to 0.

---

**input** : Input and output alphabets  $\mathcal{X}, \mathcal{Y}$

$P : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$  - an oracle to the channel transition probability

$a$  - capacity approximation parameter

**output:**  $\mathcal{P} \in \mathbb{R}^{\mathcal{X}}$  - an input distribution with a nearly optimal rate for this channel

$\mathcal{R} \in \mathbb{R}$  - the information rate of  $\mathcal{P}$

[1] Set  $\ell = 0$  and set  $\mathcal{P}^{(1)}$  to be the uniform distribution.

[2] **do**

1.  $\ell \leftarrow \ell + 1$

2. For every  $y \in \mathcal{Y}$  compute

$$\mathcal{D}_y^{(\ell)} = \sum_{x \in \mathcal{X}} \mathcal{P}^{(\ell)}(x) P_{x \rightarrow y}$$

3. For every  $x \in \mathcal{X}$  compute

$$\mathcal{W}_x^{(\ell)} = \prod_y \left( \frac{\mathcal{P}^{(\ell)}(x) P_{x \rightarrow y}}{\mathcal{D}_y^{(\ell)}} \right)^{P_{x \rightarrow y}}$$

4. For every  $x \in \mathcal{X}$ , compute

$$\mathcal{P}^{(\ell+1)}(x) = \frac{\mathcal{W}_x^{(\ell)}}{\sum_{x'} \mathcal{W}_{x'}^{(\ell)}}$$

**while**  $\max_x \log_2 \left( \frac{\mathcal{P}^{(\ell+1)}(x)}{\mathcal{P}^{(\ell)}(x)} \right) \geq a$

[3] Compute

$$\mathcal{R} = \sum_x \sum_y \mathcal{P}(x) \cdot P_{x \rightarrow y} \cdot \log \left( \frac{P_{x \rightarrow y}}{\mathcal{D}_y^{(\ell)}} \right)$$

Return  $\mathcal{P}^{(\ell+1)}(x), \mathcal{R}$ .

---

For example, when  $k = n - 2$ , computing  $\mathcal{Q}_{y,x}^{(\ell)}$  and  $\mathcal{P}^{(\ell)}$  in Step 2 takes  $O(n^2 \cdot 2^n)$  time in the sparse implementation, while the naïve implementation would take  $O(2^{2n-2})$  time. Using these modifications, we were able to compute  $\mathcal{C}_{n,k}$  for all  $k, n$  such that  $n \in [18, 24]$  and  $k \in [n - 3, n - 1]$ .

### 3.3 Results

In Section 2, we showed how the values  $\mathcal{C}_{n,k}$  for all  $k \in [n]$  can be used to obtain an upper bound on  $\mathcal{C}_n(d)$  via Equation (2.2). By employing our improved BAA algorithms, we were able to compute many previously unknown  $\mathcal{C}_{n,k}$  values. Nevertheless, for  $22 \leq n \leq 28$ , there are  $\mathcal{C}_{n,k}$  values that we could not compute even with the improved algorithms. To obtain upper bounds on these unknown  $\mathcal{C}_{n,k}$  values, we used the minimum value resulting from Lemma 2.3 and Lemma 2.5. Then, for every fixed  $d$ , we simply compute

$$\mathcal{C}(\text{BDC}_d) \leq \min_{n \in [28]} \left( \frac{1}{n} \sum_{k=1}^n \binom{n}{k} d^{n-k} (1-d)^k \mathcal{C}_{n,k} \right). \quad (3.1)$$

To prove Theorem 1.1, we will use Rahmati and Duman's result [RD14, Theorem 1].

**Theorem 3.1.** [RD14, Theorem 1] *Let  $\lambda, d' \in [0, 1]$  and denote  $d = \lambda d' + 1 - \lambda$ . Then,*

$$\frac{\mathcal{C}(\text{BDC}_d)}{1-d} \leq \frac{\mathcal{C}(\text{BDC}'_{d'})}{1-d'}.$$

When  $d' = 0.68$ , we get that  $\mathcal{C}(\text{BDC}'_{d'})/(1-d') = 0.3745$  and thus, for every  $d \geq 0.68$ , we have that  $\mathcal{C}(\text{BDC}_d) \leq 0.3745 \cdot (1-d)$ . Results for smaller values of  $d$  are listed in Table 1 and plotted in Figure 1.

## 4 A new Lower Bound

### 4.1 The framework of Drinea and Mitzenmacher

We give a brief description of the main framework given in [DM07]. Their framework suits any channel that introduce i.i.d. deletions and also i.i.d. duplications. Such a channel is described by a probability distribution  $G$  over the nonnegative integers, where a nonnegative integer  $j$  is sampled with probability  $G_j$ . The  $\text{BDC}_d$  is defined with  $G_0 = p$ ,  $G_1 = 1 - p$ , and for every  $j \geq 2$ ,  $G_j = 0$ . Another channel we consider in this work is the  $\text{PRC}_\lambda$  that is defined as follows

**Definition 1.** *Let  $\lambda > 0$ . The Poisson repeat channel with parameter  $\lambda$  ( $\text{PRC}_\lambda$ ) replaces each transmitted bit randomly (and independently of other transmitted bits), with a discrete number of copies of that bit, distributed according to the Poisson distribution with parameter  $\lambda$ .*

This channel was first defined by Mitzenmacher and Drinea in [MD06] who used it to prove a lower bound of  $(1-d)/9$  on the rate of the BDC. What they observed is that a code for the  $\text{PRC}_\lambda$  having rate  $\mathcal{R}$ , yields a code for the  $\text{BDC}_d$  of rate  $(1-d) \cdot \mathcal{R}/\lambda$ .

| $p$  | New Upper Bound            | Best Known<br>Upper Bound [FD10, DMP07] |
|------|----------------------------|---|
| 0.01 | <b>0.9583</b> ( $n = 24$ ) | 0.963                                   |
| 0.02 | <b>0.9189</b> ( $n = 24$ ) | 0.926                                   |
| 0.03 | <b>0.8817</b> ( $n = 24$ ) | 0.891                                   |
| 0.04 | <b>0.8467</b> ( $n = 24$ ) | 0.858                                   |
| 0.05 | <b>0.8139</b> ( $n = 24$ ) | 0.816                                   |
| 0.10 | <b>0.6762</b> ( $n = 22$ ) | 0.689                                   |
| 0.15 | <b>0.5660</b> ( $n = 22$ ) | 0.579                                   |
| 0.20 | <b>0.4786</b> ( $n = 22$ ) | 0.491                                   |
| 0.25 | <b>0.4083</b> ( $n = 22$ ) | 0.420                                   |
| 0.30 | <b>0.3513</b> ( $n = 22$ ) | 0.362                                   |
| 0.35 | <b>0.3045</b> ( $n = 22$ ) | 0.315                                   |
| 0.40 | <b>0.2648</b> ( $n = 23$ ) | 0.275                                   |
| 0.45 | <b>0.2309</b> ( $n = 23$ ) | 0.241                                   |
| 0.50 | <b>0.2015</b> ( $n = 24$ ) | 0.212                                   |
| 0.55 | <b>0.1755</b> ( $n = 25$ ) | 0.187                                   |
| 0.60 | <b>0.1524</b> ( $n = 27$ ) | 0.165                                   |
| 0.65 | <b>0.1313</b> ( $n = 28$ ) | 0.144                                   |
| 0.68 | <b>0.1199</b> ( $n = 28$ ) | —                                       |

Table 1: Our upper bound compared to the upper bound computed in [FD10]. We note that for  $d = 0.05$  the previous best known upper bound is given in [DMP07]. Alongside the bound, we write the value of  $n$  for which the minimum was obtained in left hand side of (3.1).

Drinea and Mitzenmacher define their code using a *run length distribution*  $\mathcal{P}$ , which samples each non-negative integer  $j$  with probability  $\mathcal{P}_j$ .<sup>2</sup>

Each codeword of length  $N$  in the code is constructed as follows. The first symbol (0 or 1) is chosen uniformly at random. The rest of the codeword is generated by deciding on the lengths of the *runs* that form it, where runs are defined as the maximal length substrings of the codeword of the same symbol (e.g., the string 110001 consists of 3 runs: 11, 000 and 1, of lengths 2, 3, and 1, respectively). The lengths of the runs of the codeword are sampled i.i.d. from  $\mathcal{P}$  and the symbols of the runs are alternating. The sampling stops when the length of the generating string is  $\geq N$ . If the length of the generated string is strictly greater than  $N$ , it is truncated.

We now define the notion of *types*. Consider a transmitted codeword  $X$  and let  $Y$  be the output of the channel upon transmitting  $X$ . We write  $X$  and  $Y$  as a concatenation of their runs, that is,  $X = r_0^X \circ r_1^X \circ \dots \circ r_m^X$ ,  $Y = r_0^Y \circ r_1^Y \circ \dots \circ r_{m'}^Y$

**Definition 2.** Let  $r_j^Y$  be a run of length  $k$ . Let  $j_1, \dots, j_s$  be a sequence of consecutive indices such that,

- The first bit of  $r_j^Y$  corresponds to the first bit of  $r_{j_1}^X$  that was not deleted by the channel.
- The runs  $r_{j_2}^X, r_{j_4}^X, \dots, r_{j_{s-1}}^X$  were deleted by the channel.
- The run  $r_{j_{s+1}}^X$  is not completely deleted by the channel.

Then, the type of  $r_j^Y$  is an  $s$  tuple that represent the lengths of the runs indexed at  $j_1, \dots, j_s$ , namely  $(|r_{j_1}^X|, \dots, |r_{j_s}^X|)$ .

The probability of a type  $t = (z, s_1, r_1, \dots, s_i, r_i)$ , is [DM07, Equation 3]

$$\mathcal{P}_z(1 - d^z) \left( \prod_{\ell=1}^i \mathcal{P}_{s_\ell} \mathcal{P}_{r_\ell} \right) d^s,$$

where  $s := s_1 + \dots + s_i$  and  $d := G_0$ . Denote by  $T$  and  $K$  the random variables representing the length and type of a runs in  $Y$ , respectively. It holds that [DM07, Equation 34],

$$\Pr[T = t, K = k] = \mathcal{P}_z \mathcal{P}_{s_1} \mathcal{P}_{s_1} \dots \mathcal{P}_{s_i} \mathcal{P}_{r_i} \cdot d^s \cdot (\rho_{z+r,k} - \rho_{r,k} \cdot d^z) \quad (4.1)$$

where  $\rho_{a,b}$  is the probability that  $a \geq 1$  bits transmitted over a channel with distribution  $G$  generate  $b$  bits. Note that  $\rho_{z,0} = G_0^z = d^z$ .

Let  $Q_{n,m}$  be the probability that the length of  $m$  consecutive runs is exactly  $n$ . It holds that  $Q_{n,m} = \sum_{\ell=1}^{n-m+1} \mathcal{P}_\ell \cdot Q_{n-\ell, m-1}$  where  $Q_{0,0} = 1$ . Let  $D = \sum_z \mathcal{P}_z d^z$  be the probability that a run is deleted from  $X$ , the distribution  $\mathcal{K}$  for run lengths in  $Y$  is given by [DM07, Equation 35]

$$\mathcal{K}_k := \Pr[K = k] = \sum_{i=1}^{\infty} D^i \sum_{z=1}^{\infty} \sum_{r=i}^{\infty} \mathcal{P}_z Q_{r,i} \cdot (\rho_{z+r,k} - \rho_{r,k} \cdot d^z). \quad (4.2)$$

---

<sup>2</sup>The distribution,  $\mathcal{P}$ , must have a *geometric decreasing tail*, that is, there are two real constants  $c_{\mathcal{P}} \in (0, 1]$ ,  $\alpha_{\mathcal{P}} \in [0, 1)$  and an integer constant  $M_{\mathcal{P}}$  such that (i)  $\mathcal{P}_j \leq c_{\mathcal{P}}$  for  $1 \leq j \leq M_{\mathcal{P}}$  and (ii)  $\mathcal{P}_j \leq (1 - \alpha_{\mathcal{P}}) \alpha_{\mathcal{P}}^{j-1}$  for  $j > M_{\mathcal{P}}$ .



Given the distribution  $\mathcal{P}$  we can compute the rate of the respective code constructed using this method with the following theorem.

**Theorem 4.1.** [DM07, Theorem 4] *The capacity of the the channel defined by the distribution  $G$  with  $G_0 > 0$  is lower bounded by*

$$\frac{1}{\frac{1+D}{1-D} \cdot \sum_z z \mathcal{P}_z} \left[ \frac{1+D}{1-D} \cdot H(\mathcal{P}) - (H(T, K) - H(\mathcal{K})) \right] \quad (4.3)$$

where  $\Pr[T = t, K = k]$  is given in (4.1) and  $\mathcal{K}$  is given in (4.2).

Note that evaluating numerically (4.3) for a given distribution  $\mathcal{P}$  is not an easy task since it involves infinite sums.

In [MD06, DM07], the authors limit themselves to the case where  $\mathcal{P}$  is a geometric distribution. In this case, they managed to derive simpler expressions to (4.3) (see [MD06, Theorem 2] for the  $\text{PRC}_\lambda$  and [DM07, Corollary 1] for the  $\text{BDC}_d$ ). However, the authors do not provide an argument for why geometric distributions should be optimal. Indeed, we will improve upon this lower bounds by constructing a better distribution  $\mathcal{P}$ .

## 4.2 Our heuristic approach to find better input distributions

In this section, we will present a heuristic approach to optimizing the input run length distribution  $\mathcal{P}$ . Note that even though our optimization of  $\mathcal{P}$  is heuristic, because  $\mathcal{P}$  is used only as parameters for Mitzenmacher and Drinea's rigorous construction, the resulting bounds are provably correct. Indeed, we plug in the resulting distribution in Theorem 4.1.

Our approach is based on a heuristic score function  $R_{\text{heur}}$ , which should approximate the information rate of Mitzenmacher and Drinea's code for the given distribution. We select the score function so that it can be optimized with a Blahut-Arimoto type algorithm.

Our main observation is that for the distributions used by Mitzenmacher and Drinea, only a small fraction of the runs are deleted. If no runs were deleted, then we could view the channel as a DMC that maps input run lengths into output run lengths and its information rate would be given by eq. 4.4, where  $P_{x \rightarrow y}$  is the transition probability matrix of input run lengths to output run lengths.

$$R_{\text{DMC}} = \sum_{x,y} \mathcal{P}_x P_{x \rightarrow y} \log \left( \frac{P_{x \rightarrow y}}{\sum_{x'} \mathcal{P}_{x'} P_{x' \rightarrow y}} \right) \quad (4.4)$$

Equation (4.4) overlooks two key effects: that the binary deletion channel can delete entire runs causing a loss of information, and the rate given in equation (4.4) is in bits *per run*. To correct for the first effect, we subtract from  $R_{\text{DMC}}$  a loss function  $\Delta \cdot \mathbb{E}_{x \leftarrow \mathcal{P}} D_x$  where  $\Delta$  is a parameter and  $D_x$  is the probability that a run of length  $x$  will be deleted by the channel. To correct for the latter effect, we normalize the rate by the inverse of the average run length  $L(\mathcal{P}) = \mathbb{E}_{x \leftarrow \mathcal{P}} x$ .

After correcting for these effects, we get our heuristic score function in eq. (4.5). In Section 4.3, we give a more detailed analysis of these approximations.

$$R_{\text{heur}} \stackrel{\text{def}}{=} \frac{1}{\sum_r \mathcal{P}_r r} \left[ R_{\text{DMC}} - \sum_r \mathcal{P}_r \Delta D_r \right] \quad (4.5)$$

| $d$  | Best previous lower bound | Direct lower bound | PRC based lower Bound |
|------|---------------------------|--------------------|-----------------------|
| 0.40 | 0.148410                  | <b>0.149810</b>    | 0.073313              |
| 0.45 | 0.122860                  | <b>0.124700</b>    | 0.067204              |
| 0.50 | 0.101860                  | <b>0.104075</b>    | 0.061094              |
| 0.55 | 0.084323                  | <b>0.086712</b>    | 0.054985              |
| 0.60 | 0.069564                  | <b>0.071838</b>    | 0.048875              |
| 0.65 | 0.056858                  | <b>0.059012</b>    | 0.042766              |
| 0.70 | 0.045324                  | <b>0.047726</b>    | 0.036657              |
| 0.75 | 0.035984                  | <b>0.037593</b>    | 0.030547              |
| 0.80 | 0.027266                  | <b>0.028371</b>    | 0.024438              |
| 0.85 | 0.019380                  | <b>0.019531</b>    | 0.018328              |
| 0.90 | 0.012378                  | <b>0.012379</b>    | 0.012219              |
| 0.95 | 0.005741                  | 0.005631           | <b>0.006105</b>       |

Table 2: Our lower bound compared to the lower bounds computed in [MD06, DM07].

In Section 4.4, we show how a Blahut-Arimoto type algorithm can be used to maximize eq. (4.5) w.r.t the input distribution  $\mathcal{P}$  in polynomial time. Roughly speaking, this optimization works by enumerating over polynomially many potential values of  $L(\mathcal{P})$ , and using an extension of the BAA similar to the one used in [LC19] to optimize  $\mathcal{P}$  under the different constraints on the average run lengths.

In practice, this enumeration converges too slowly, so we use a heuristic basin-hopping optimization algorithm to select the external parameters  $\Delta, L_0$ . For each setting of  $\Delta, L_0$ , we use a BAA type algorithm to find a distribution  $\mathcal{P}(\Delta, L_0)$  which maximizes eq. (4.5) under the constraint of  $L(\mathcal{P}) = L_0$ . We use the basin-hopping optimization to select the parameters for which the distribution  $\mathcal{P}(\Delta, L_0)$  yields the best lower bound from Mitzenmacher and Drinea’s construction.

Recall that we use this heuristic only to optimize the parameters needed for applying Mitzenmacher and Drinea’s rigorous construction. Therefore, the lower-bounds obtained by this method are rigorous (though not necessarily tight).

We use this heuristic approach to optimize the input distribution for different deletion probabilities  $d \in (0, 1)$ , giving us an improved lower bound in the intermediate deletion probability regime  $d \in [0.4, 0.9]$  (see Table 2). In order to lower bound the capacity of the BDC in the  $d \rightarrow 1$  regime, we use the same technique to lower bound the capacity of the PRC with a given parameter  $\lambda_0 = 0.19$ , yielding the bound

$$\frac{\mathcal{C}(\text{BDC}_d)}{1-d} \geq \frac{\mathcal{C}(\text{PRC}_{\lambda_0})}{\lambda_0} > \frac{0.0232}{0.19} > 0.1221.$$

### 4.3 Analysis of the heuristic

In Section 4, we recalled Mitzenmacher and Drinea’s approach to converting distributions on lengths of runs into error correcting codes for the binary deletion channel and the Poisson repeat channel [DM07]. We gave a very high-level heuristic formula for the

information rate a Mitzenmacher and Drinea’s type code would achieve for a given run length distribution and optimized the input run length distribution using this heuristic formula. In this section, we will give a slightly more formal derivation of the heuristic formula from Section 4.

Let  $\mathcal{P}$  be a distribution on run lengths to be input into Mitzenmacher and Drinea’s construction. If none of the input runs were completely deleted, then the channel could effectively be seen as a discrete memory-less channel on runs, and the information rate of the code (per run) would be given by

$$R_{\text{memoryless}} = \sum_{x,y} \mathcal{P}_i P_{x \rightarrow y} \log \left( \frac{P_{x \rightarrow y}}{\sum_{x'} \mathcal{P}_{x'} P_{x' \rightarrow y}} \right) \quad (4.6)$$

However, this formula for the rate overlooks two key aspects of the deletion channel and the code. Namely, that runs can be deleted (i.e., the BDC and the PRC are not memoryless channels) and that different run lengths have a different cost for the rate of the code.

Consider the effects of runs being deleted by the channel. Every time a run is deleted by the channel, this causes the preceding and following input runs to be merged and results in a loss of information as the output run can no longer be assigned to a single input run.

Let  $d$  denote the probability that the channel will delete any single bit ( $d$  is the deletion rate for the deletion channel and  $d = e^{-\lambda}$  for the  $\text{PRC}_\lambda$  channel), and let  $D = \mathbb{E}_{x \leftarrow \mathcal{P}} d^x$  denote the probability that any single input run is deleted.

The first approximation that we will make is that  $D$  is small. This assumption is reasonable for most useful distributions. For instance, for the distributions used in [MD06, DM07], the run deletion probabilities were around  $D \approx 4.4\%$ .

If we completely neglected the deletion probability  $D$ , then we would return to the formula for a memoryless channel and our optimization will no longer have any pressure to prefer longer runs which are less likely to be deleted by the channel, effectively increasing  $D$  and possibly invalidating our assumption. Therefore, we need to take into account some effects of order  $O(D)$ , but we allow ourselves to neglect those of order  $O(D^2)$ . In other words, we will take into account the effects of output runs resulting from merging at most 2 input runs.

Consider an input run of length  $r$ . The probability that this run will be deleted by the channel is  $D_r = d^r$ . Denote by  $r_{\text{before}}, r_{\text{after}}$  the lengths of the runs immediately before and after it, respectively. If the run  $r$  is deleted, then the latter two ( $r_{\text{before}}, r_{\text{after}}$ ) will be merged by the channel.

The possibility of runs being deleted or merged can lead to two adverse effects. First, the added uncertainty of determining which runs were merged with which, can increase the entropy required for the decoding. Due to the high degree of difficulty in estimating this effect and some evidence that it is less significant (see e.g. [KD09]), we neglect it. The second effect, on which we will focus most of our efforts, is that some amount of information  $\Delta_I(r_{\text{before}}, r_{\text{after}})$  is lost because we are given the output of the channel only on the merged run of length  $r_{\text{before}} + r_{\text{after}}$  (and not on the individual runs of lengths  $r_{\text{before}}, r_{\text{after}}$ ). The loss of information is quantified in eq. (4.7), by comparing the information rate of two separate runs of lengths  $r_1, r_2$ , with the rate of a single merged run of length  $r_1 + r_2$ .

$$\begin{aligned} \Delta_I(r_1, r_2) \stackrel{\text{def}}{=} & \sum_{o_1, o_2} P_{r_1 \rightarrow o_1} P_{r_2 \rightarrow o_2} \left( \log \left( \frac{P_{r_1 \rightarrow o_1}}{\sum_r \mathcal{P}_r P_{r \rightarrow o_1}} \right) + \log \left( \frac{P_{r_2 \rightarrow o_2}}{\sum_r \mathcal{P}_r P_{r \rightarrow o_2}} \right) \right) - \\ & - \sum_o P_{r_1+r_2 \rightarrow o} \log \left( \frac{P_{r_1+r_2 \rightarrow o}}{\sum_{\rho_1, \rho_2} \mathcal{P}_{\rho_1} \mathcal{P}_{\rho_2} P_{\rho_1+\rho_2 \rightarrow o}} \right) \end{aligned} \quad (4.7)$$

Note that this information loss depends only on the lengths of the runs being merged, and that these lengths are independent of the length of the run being deleted. Let  $\Delta_I = \mathbb{E}_{r_1, r_2 \leftarrow \mathcal{P}} \Delta_I(r_1, r_2)$  denote the average information lost due to such a merger. Denote by  $X$  the distribution of input codewords, by  $Y$  the output codeword and let  $\mathcal{T}$  denote the random variable containing the division of input runs into types (i.e. which runs in  $X$  were merged due to deletions by the channel). Putting our approximations into an information theoretic language, we have:

$$\begin{aligned} \text{Information Rate} &= I(X; Y) = H(X) - H(X | Y) \\ &= H(X) - H(\mathcal{T}, X | Y) + H(\mathcal{T} | X, Y) \\ &= H(X) - H(X | Y, \mathcal{T}) - H(\mathcal{T} | Y) + H(\mathcal{T} | X, Y) \\ &= \frac{|X|}{\mathbb{E}_{r \leftarrow \mathcal{P}} r} [R_{\text{memoryless}} - D\Delta_I + O(D^2)] - H(\mathcal{T} | Y) + H(\mathcal{T} | X, Y) \end{aligned} \quad (4.8)$$

Equation (4.8) gives us our separation of the exact information rate into an approximate formula (eq. (4.9)) and error terms.

$$R_{\text{heuristic}} = \frac{1}{\mathbb{E}_{r \leftarrow \mathcal{P}} r} [R_{\text{memoryless}} - D\Delta_I] \approx \frac{\text{Information Rate}}{|X|} \quad (4.9)$$

The neglected error terms in eq. (4.8) are  $D^2$ ,  $H(\mathcal{T} | Y) / |X|$  and  $H(\mathcal{T} | X, Y) / |X|$ . The first term is negligible due to our assumption that  $D \ll 1$ , and we neglect the other two mainly because of the difficulty of including them in the optimization (the latter term is also neglected by Mitzenmacher and Drinea [DM07] and both are difficult to compute directly [KD09]).

The last heuristic step in our analysis is to neglect the dependence of  $\Delta_I$  on the input distribution  $\mathcal{P}$ . Instead, we will make estimates  $\Delta \stackrel{?}{=} \Delta_I$ , and maximize  $R_{\text{heuristic}}$  assuming this value  $\Delta$  of information loss per merge (but without limiting  $\mathcal{P}$  to distributions that maintain the equation  $\Delta_I(\mathcal{P}) = \Delta$ ). This is a heuristic approximation, but it is somewhat justified assuming that  $\Delta_I$  doesn't vary too wildly between otherwise "good" input distributions and that this variation is then multiplied by  $D \ll 1$  in eq. (4.8)

Recall that the goal of eq. (4.9) is not to directly prove a lower bound, and may be far less accurate than using a calculation similar to the one described in [DM07]. The main reason to use this approximation is that it gives a closed formula, and that a BAA-style convex optimization algorithm can be used to maximize it, which can then be used as parameters for Mitzenmacher and Drinea's construction.

## 4.4 Reduction to Convex Optimization

In Section 4.3, we gave a heuristic argument for a simplified formula that can be used to estimate the information rate of Mitzenmacher-Drinea type codes for the binary deletion channel and the Poisson repeat channel. This formula can be seen as an information rate formula with 2 correction terms corresponding to different “costs” of sending different run lengths (the cost due to the information lost in the event of a deletion and the overhead of sending longer runs). In this section we will construct a Blahut-Arimoto type algorithm to maximize it.

In some sense, generalizing the Blahut-Arimoto algorithm to the cost types needed for our construction can be seen as an extension of the algorithm presented by Li and Cai [LC19] who show how to extend the Blahut-Arimoto to quantum-classical codes where only distributions below a certain total cost are allowed. We will reduce our optimization problem into an optimization problem in the class of problems solved by Li and Cai, and prove that our optimization yields a  $\pm\varepsilon$ -approximation of the optimal distribution score within  $\text{poly}(1/\varepsilon, 1/(1-d))$  and  $\text{poly}(1/\varepsilon, \lambda, 1/\lambda)$  time for the binary deletion channel and the poisson repeat channel with parameters  $d$  and  $\lambda$  resp.

Our approximation for the rate of the resulting code is given by the formula

$$R_{\text{heuristic}} \stackrel{\text{def}}{=} \frac{1}{\sum_r \mathcal{P}_r r} \left[ R_{\text{memoryless}} - \sum_r \mathcal{P}_r \Delta D_r \right] \quad (4.10)$$

This approximation can be viewed as a combination of three terms:

- The basic information rate of a memory-less channel of runs  $R_{\text{memoryless}}$ .
- A correction term for the information lost due to deletions  $\sum_r \mathcal{P}_r \Delta D_r$ .
- A “price factor” corresponding to the average resource cost of transmitting a run  $\frac{1}{\sum_r \mathcal{P}_r r}$ .

The main claim we will prove in this section is that a nearly optimal distribution for this heuristic formula can be efficiently found:

**Lemma 4.1.** *There exists an algorithm that returns a distribution  $\mathcal{P}$  for which*

$$R_{\text{heuristic}}(\mathcal{P}) \geq \sup_{\mathcal{P}'} \{R_{\text{heuristic}}(\mathcal{P}')\} - \varepsilon$$

*in  $\text{poly}(1/\varepsilon, 1/(1-d), \Delta)$  time for the deletion channel with deletion probability  $d$  and  $\text{poly}(1/\varepsilon, \lambda, 1/\lambda, \Delta)$  time for Poisson repeat channel with parameter  $\lambda$ .*

*Proof of Lemma 4.1.* The main difficulty in maximizing eq. (4.10) is that the price factor term  $\frac{1}{\sum_r \mathcal{P}_r r}$  causes the relationship between  $R_{\text{heuristic}}$  and  $\mathcal{P}$  to be non-convex. This issue will be the main focus of this section, and we will overcome it by extracting  $L(\mathcal{P}) \stackrel{\text{def}}{=} \sum_r \mathcal{P}_r r$  to be an external hyperparameter of the optimization. We then separate the search into many searches under the external condition the  $L(\mathcal{P}) = \ell$  for different values of  $\ell$  and show that polynomially many searches suffice.

Unlike  $\Delta$ , neglecting the effect of the fact that  $\mathcal{P}$  must satisfy  $L(\mathcal{P}) = \ell$  in each of these searches would lead us to select distributions of runs too heavily skewed towards

long runs, lowering the rate they can achieve. So for any given value  $\ell$ , we limit our search to distributions  $\mathcal{P}$  under the condition that  $L(\mathcal{P}) = \ell$ . The central point of our analysis will be to show that the supremum of  $R_{\text{heuristic}}$  with respect to  $\mathcal{P}$  under the condition that  $L(\mathcal{P}) = \ell$ , cannot change too rapidly with  $\ell$ , this is what allows us to enumerate over only polynomially many guesses for  $\ell$  in order to approximate the optimal distribution  $\mathcal{P}$  to within a small additive error.

Let  $\eta \stackrel{\text{def}}{=} \max\{\log \Delta, 1\}$ . We begin with the simple observation that

$$R_{\text{heuristic}}(\mathcal{P}) \geq \Omega\left(\frac{1-d}{\eta}\right), \Omega\left(\frac{\min\{1, \lambda\}}{\eta}\right)$$

can be achieved for the BDC and the PRC respectively. In particular, this is true for the Morse distribution

$$\mathcal{P}_i = \frac{1}{2}\delta_{i,t} + \frac{1}{2}\delta_{i,2t}$$

which returns either  $t$  or  $2t$  each w.p.  $1/2$ , for

$$t = \left\lceil \eta \frac{100}{1-d} \right\rceil, \left\lceil \eta \max\left\{100, \frac{100}{\lambda}\right\} \right\rceil$$

This distribution obtains a non-negligible rate, because  $L(\mathcal{P}) \leq t = O\left(\frac{\eta}{1-d}\right)$ , the deletion probability for a run from this distribution is  $D < \frac{e^{-100}}{\Delta}$  and the probability of missing the reconstruction of any run is also very small, implying that

$$R_{\text{heuristic}}(\mathcal{P}) L(\mathcal{P}) = R_{\text{memoryless}}(\mathcal{P}) - \Delta D(\mathcal{P}) = \Omega(1)$$

It is easy to see that setting  $L(\mathcal{P}) = \ell$  to be extremely large would result in an information transfer rate of at most  $O(\log(\ell)/\ell) = o(1)$ . Therefore, in order to approximately maximize  $R_{\text{heuristic}}(\mathcal{P})$ , the average lengths of runs in this distribution  $L(\mathcal{P})$  is at most polynomially large in the parameters of Lemma 4.1.

Our next goal will be to prove that maximizing  $R_{\text{heuristic}}$  for a (small) discrete subset of values of  $\ell$  suffices to approximate its maximum on the whole range of run lengths. We denote by  $h(\mathcal{P})$  the non-normalized version of eq. 4.10:

$$\begin{aligned} h(\mathcal{P}) &\stackrel{\text{def}}{=} R_{\text{heuristic}}(\mathcal{P}) L(\mathcal{P}) = R_{\text{memoryless}} - \Delta D \\ &= \sum_{i,j} \mathcal{P}_i \mathcal{P}_{i \rightarrow j} \log\left(\frac{\mathcal{P}_{i \rightarrow j}}{\sum_{i'} \mathcal{P}_{i'} \mathcal{P}_{i' \rightarrow j}}\right) - \Delta \sum_i \mathcal{P}_i D_i \end{aligned} \quad (4.11)$$

Let

$$S_\ell = \left\{ \mathcal{P} \in \mathbb{R}^{\mathbb{N}} \mid \mathbb{E}_{r \leftarrow \mathcal{P}} r = \ell \wedge \sum_i \mathcal{P}_i = 1 \wedge \forall i \mathcal{P}_i \geq 0 \right\}$$

be the set of distributions with average cost  $\ell$ . We define  $I(\ell) \stackrel{\text{def}}{=} \sup_{\mathcal{P} \in S_\ell} h(\mathcal{P})$  to be the optimal value of the non-normalized rate when fixing the average input run length to some value  $\ell$ .

The main property of  $I$  that we will use in our analysis is that it is monotonously non-decreasing in  $\ell$  on the range  $(0, \infty)$ .

**Claim 4.2.** For any  $0 < \ell_1 < \ell_2$ , it holds that  $I(\ell_2) \geq I(\ell_1)$ .

We leave the proof of Claim 4.2 to the end of the section. We use it to bound from above the speed with which our approximation for the rate of the code can change when optimizing under slightly different  $L = \ell$  constraints. In particular, the fact that  $I$  is non-decreasing means that for any  $\ell_1, \ell_2$  and for any  $\ell \in (\ell_1, \ell_2)$ , we have:

$$\begin{aligned} \frac{\ell_1}{\ell_2} \sup_{\mathcal{P}_1 \in S_{\ell_1}} R_{\text{heuristic}}(\mathcal{P}_1) &= \frac{I(\ell_1)}{\ell_2} < \sup_{\mathcal{P} \in S_{\ell}} R_{\text{heuristic}}(\mathcal{P}) = \\ &= \frac{I(\ell)}{\ell} < \frac{I(\ell_2)}{\ell_1} = \frac{\ell_2}{\ell_1} \sup_{\mathcal{P}_2 \in S_{\ell_2}} R_{\text{heuristic}}(\mathcal{P}_2) \end{aligned} \quad (4.12)$$

Eq. 4.12 proves that it suffices to compute  $I(\ell)$  only in a discrete set of points  $\ell_i$ , strictly separated on the logarithmic scale, in order to approximate it everywhere. Because we already showed that the value of  $\ell$  for which  $R_{\text{heuristic}}$  is maximized is polynomially bounded as a function of the parameters of Lemma 4.1, this implies that polynomially many samples suffice to approximate this optimization.

The last step in our construction is to show that  $I(\ell)$  can be efficiently approximated for any given  $\ell$ . To this end we employ the algorithm proposed by Li and Cai [LC19] who show that the Blahut-Arimoto algorithm can be extended to maximize functions of the form of  $h(\mathcal{P})$ , under “cost limit” constraints of the form  $L(\mathcal{P}) \leq \ell$ . For any given  $\ell$  we can generate a candidate distribution  $\mathcal{P}' \in \bigcup_{\ell' < \ell} S_{\ell'}$  using Li and Cai’s algorithm, and then convert it into a distribution  $\mathcal{P} \in S_{\ell}$  for which  $h(\mathcal{P})$  is arbitrarily close to  $h(\mathcal{P}')$  using the same construction as in the proof of Claim 4.2. □

*Proof of Claim 4.2.* Let  $\frac{1}{2} > \varepsilon > 0$  be some number, and let  $\mathcal{P}_1 \in S_{\ell_1}$  be some distribution for which  $h(\mathcal{P}_1) \geq I(\ell_1) - \varepsilon$ . Our goal will be to show that there exists some  $\mathcal{P}_2 \in S_{\ell_2}$  for which  $I(\ell_2) \geq h(\mathcal{P}_2) \geq h(\mathcal{P}_1) - \varepsilon \geq I(\ell_1) - 2\varepsilon$ . Because we prove this for an arbitrarily small  $\varepsilon$ , it will imply that  $I(\ell_2) \geq I(\ell_1)$ .

Let  $\mathcal{P}_2 = (1 - c)\mathcal{P}_1 + c\delta_N$  be the distribution that returns a random sample from  $\mathcal{P}_1$  with probability  $1 - c$  and the value  $N$  otherwise, where  $c = \frac{\ell_2 - \ell_1}{N - \ell_1} > 0$ , and

$$N > \max \left\{ 2\ell_1, \frac{1}{(\ell_2 - \ell_1)^2}, \left( \frac{R_{\text{memoryless}}(\mathcal{P}_1)}{10\varepsilon} \right)^2, \left( \frac{1}{10\varepsilon} \right)^2 \right\}$$

is a sufficiently large integer. From its construction  $\mathcal{P}_2 \in S_{\ell_2}$ .

Our next goal is to show that  $h(\mathcal{P}_2) \geq h(\mathcal{P}_1) - \varepsilon$ . We do this by opening up the definition of  $h$ :

$$h(\mathcal{P}_2) = \underbrace{R_{\text{memoryless}}(\mathcal{P}_2)}_{\geq (1-c)R_{\text{memoryless}}(\mathcal{P}_1)} - \underbrace{\Delta \mathbb{E}_{r \leftarrow \mathcal{P}_2} d^r}_{\leq \Delta \mathbb{E}_{r \leftarrow \mathcal{P}_1} d^r + c\Delta} \geq h(\mathcal{P}_1) - \underbrace{c(\Delta + R_{\text{memoryless}}(\mathcal{P}_1))}_{\leq \varepsilon} \quad (4.13)$$

□

## References

- [Ari72] Suguru Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.
- [Bla72] Richard Blahut. Computation of channel capacity and rate-distortion functions. *IEEE transactions on Information Theory*, 18(4):460–473, 1972.
- [BLC<sup>+</sup>16] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. *ACM SIGARCH Computer Architecture News*, 44(2):637–649, 2016.
- [CR20] Mahdi Cheraghchi and João Ribeiro. An overview of capacity results for synchronization channels. *IEEE Transactions on Information Theory*, 67(6):3207–3232, 2020.
- [CS22] Roni Con and Amir Shpilka. Improved constructions of coding schemes for the binary deletion channel and the poisson repeat channel. *IEEE Transactions on Information Theory*, 2022.
- [Dal11] Marco Dalai. A new bound on the capacity of the binary deletion channel with high deletion probabilities. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 499–502. IEEE, 2011.
- [DM07] Eleni Drinea and Michael Mitzenmacher. Improved lower bounds for the capacity of iid deletion and duplication channels. *IEEE Transactions on Information Theory*, 53(8):2693–2714, 2007.
- [DMP07] Suhas Diggavi, Michael Mitzenmacher, and Henry D Pfister. Capacity upper bounds for the deletion channel. In *2007 IEEE International Symposium on Information Theory*, pages 1716–1720. IEEE, 2007.
- [Eli55] Peter Elias. Coding for noisy channels. *IRE Convention Record*, 4:37–46, 1955.
- [FD10] Dario Fertonani and Tolga M Duman. Novel bounds on the capacity of the binary deletion channel. *IEEE Transactions on Information Theory*, 56(6):2753–2765, 2010.
- [GL18] Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 2018.
- [HMG19] Reinhard Heckel, Gediminas Mikutis, and Robert N Grass. A characterization of the DNA data storage channel. *Scientific reports*, 9(1):1–12, 2019.
- [KD09] Adam Kirsch and Eleni Drinea. Directly lower bounding the information capacity for channels with iid deletions and duplications. *IEEE Transactions on Information Theory*, 56(1):86–102, 2009.



- [KMS10] Adam Kalai, Michael Mitzenmacher, and Madhu Sudan. Tight asymptotic bounds for the deletion channel with small deletion probabilities. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 997–1001. IEEE, 2010.
- [LC19] Haobo Li and Ning Cai. A blahut-arimoto type algorithm for computing classical-quantum channel capacity. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 255–259. IEEE, 2019.
- [MBT10] 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.
- [MD06] 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.
- [Mit09] Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.
- [PLW22] Francisco Pernice, Ray Li, and Mary Wootters. Efficient capacity-achieving codes for general repeat channels. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 3097–3102. IEEE, 2022.
- [RD14] Mojtaba Rahmati and Tolga M Duman. Upper bounds on the capacity of deletion channels using channel fragmentation. *IEEE Transactions on Information Theory*, 61(1):146–156, 2014.
- [Rub22] Ittai Rubinfeld. Explicit and efficient construction of nearly optimal rate codes for the binary deletion channel and the poisson repeat channel. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 105:1–105:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.
- [TPFV21] 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*, 68(4):2239–2265, 2021.
- [Wol92] Michael Edward Wolf. *Improving locality and parallelism in nested loops*. stanford university, 1992.
- [Yeu08] Raymond W Yeung. *Information theory and network coding*. Springer Science & Business Media, 2008.
- [YGM17] S.M. Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and error-free DNA-based data storage. *Scientific reports*, 7(1):1–6, 2017.