Flip a Failure into a Success: Improved Bit Flipping Decoding for QC-MDPC Codes

Paolo Santini [1], Davide De Zuane [1,2], Alessio Baldelli [1], and Marco Baldi [1]

[1] Università Politecnica delle Marche, Ancona, Italy
[2] Scuola IMT Alti Studi, Lucca, Italy
{p.santini, m.baldi}@univpm.it,
davide.dezuane@imtlucca.it, a.baldelli@pm.univpm.it

Abstract. Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) codes are a family of error correcting codes admitting parity-check matrices composed of sparse circulant blocks. QC-MDPC codes have been used for the design of BIKE, one of the finalists in the NIST competition for the standardization of post-quantum cryptography. Decoding of QC-MDPC codes with cryptographically relevant parameters is intrinsically bound to fail, resulting in a decoding failure rate (DFR) that is nonzero. To achieve INDistinguishability under Adaptively Chosen Ciphertext Attacks (IND-CCA2), the DFR must not exceed $2^{-\lambda}$, with λ being the security parameter. QC-MDPC codes are customarily decoded with a Bit Flipping (BF) algorithm. Especially at very low DFR values, error patterns having a large intersection with *near-codewords* (which are vectors corresponding to columns of the parity-check matrix, up to some shift) are the main cause of decoding failures.
In this paper, we show how a BF decoder can be tweaked to exploit the knowledge about near-codewords. Since error vectors that cause decoding failures are likely making the decoder converge to the closest near-codeword (i.e., to the near-codeword with the largest amount of overlapping positions with the error vector), we exploit such a harmful but predictable behavior: we let the decoder recognize, and consequently correct, syndromes of near-codewords. This modification comes with a very mild computational overhead and can be applied to any BF decoder. As a concrete application, we focus on BIKE parameters for NIST security category 1. We show that a recently proposed BF variant called BF-Max outperforms significantly the two decoders used by BIKE within the NIST competition, achieving a significantly lower DFR with a comparable computational complexity.

1 Introduction

Problems arising from coding theory provide the basis for developing some of the most promising *post-quantum cryptosystems*. For instance, the three finalist Key Encapsulation Mechanisms (KEMs) in the fourth round of the NIST post-quantum cryptography competition, namely, BIKE, Classic McEliece and HQC, are based on error correcting codes.

As a common feature of both BIKE and HQC, decryption may fail even for genuine ciphertexts: this is because the underlying decoder may terminate the decoding process with a failure, and opens up for attacks leveraging decryption queries[1]. To achieve INDistinguishability under Adaptively Chosen Ciphertext Attacks (IND-CCA2), the decryption failure probability must be smaller than $2^{-\lambda}$, with λ being the security parameter [1].

In BIKE, decryption failures correspond to *decoding failures*, that is, events in which the decoder cannot correct the error pattern sampled during encapsulation. Thus, the decryption failure probability corresponds to the Decoding Failure Rate (DFR) and IND-CCA2 can be achieved only if the (average) DFR is less than $2^{-\lambda}$. In particular, BIKE employs Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) codes, for which efficient decoding strategies are intrinsically characterized by some non-zero DFR. To guarantee that the DFR is $< 2^{-\lambda}$, one needs an efficient decoding algorithm with a sound and convincing theoretical assessment about its DFR. Even though, in recent years, a variety of works have studied this topic [2]–[11], arguably, none of these works is a silver bullet and this problem remains open. As highlighted in the NIST report on the fourth round of the standardization process [12], the DFR issue was a relevant obstacle standing between BIKE and its standardization.

The remainder of this section introduces, with some more technical details, the QC-MDPC codes used in BIKE, their decoding and the related issues.

LDPC, MDPC and BF decoding. MDPC codes are close relatives of Low Density Parity-Check (LDPC) codes, introduced by Gallager in 1962 [13]. LDPC and MDPC codes admit *sparse parity-check matrices*, that is, matrices where rows and columns have low Hamming weight. For MDPC codes, parity-check matrices are somewhat denser than those of LDPC codes[2].

In BIKE, the secret key is a parity-check matrix of the form $\mathbf{H} = (\mathbf{H}_1, \mathbf{H}_2)$, with each $\mathbf{H}_1$ and $\mathbf{H}_2$ being a sparse $r \times r$ *circulant matrix*, i.e., such that each row (and column) is the cyclic shift of the previous row (and column). The resulting code is sometimes called *double circulant*. For genuine ciphertexts, decryption boils down to decoding the syndrome of a moderately low weight error vector, sampled uniformly at random from the set of all errors with the same weight t.

LDPC and MDPC codes are efficiently decoded through iterative decoders. One of the simplest decoders of this type is the Bit Flipping (BF) algorithm. Let $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$ be the syndrome provided as input to the algorithm, with $\mathbf{e}$ being the weight-t error vector to be obtained as output. The BF decoder works by iteratively establishing an estimate $\widehat{\mathbf{e}}$ for the error vector and updating the syndrome accordingly as $\mathbf{s} + \widehat{\mathbf{e}}\mathbf{H}^\top$. This iterative procedure continues until the updated syndrome is an all-zero vector (in this case, decoding is successful, unless

[1] By genuine ciphertexts, we mean ciphertexts that have not been manipulated by adversaries. With overwhelming probability, manipulations are detected and the ciphertexts rejected.

[2] An increased density for MDPC codes is necessary to avoid certain types of key recovery attacks.

there is an extremely rare occurrence of an undetected error), or the maximum number of iterations is reached (in such a case, we experience a decoding failure). In each iteration, decisions are taken using so-called *counters*: geometrically, the i-th counter is the Hamming weight of the projection of the *syndrome* on the i-th column of the parity-check matrix. Whenever a counter is high, it is likely that the current estimate for the associated bit is wrong and, thus, such an estimate gets flipped.

Because of its great trade-off between simplicity, computational efficiency and error correction capability, the BF decoder is typically the standard choice for cryptosystems based on MDPC codes [3], [4], [11], [14], [15].

The inherent difficulty of predicting the DFR. Given how BIKE parameters are selected, for a security parameter of λ and for circulants of side r, the weight of each circulant is $v \approx \frac{1}{2}\lambda + \frac{1}{2}\log_2(r)$, while the amount of errors is $t \approx \lambda + \frac{1}{2}\log_2(r)$ [16]. Since a double circulant QC-MDPC code has always minimum distance $\leqslant 2v \approx t + \frac{1}{2}\log_2(r)$, the number of errors is greater than half the minimum distance. Thus, there always exist error vectors that exceed the boundaries of decoding spheres and, hence, any decoder must have some non-zero DFR. Through geometrical observations and counting arguments, we can lower bound the DFR of any decoder as $\frac{\binom{2v}{v}\binom{2r-2v}{t-v}}{2\binom{2r}{t}}$ [16, Section 3.1]. However, this bound is not very useful for estimating the actual DFR of a BF decoder, which can be considerably higher than the bound. In fact, this bound assumes an optimal decoding strategy, such as maximum likelihood decoding, whereas BF decoding has low complexity but is inherently suboptimal. Unfortunately, finding a tighter bound, and possibly an upper bound for the DFR of a BF decoder, or even just a good estimate of it, turns out to be an extremely difficult task. While good bounds exist for one-iteration BF decoders [2], [6], [17], when it comes to considering multiple iterations, state-of-the-art models introduce heuristic assumptions and/or consider ad-hoc and somewhat downgraded settings for the decoder [3]–[5], [7]–[11].

Near-codewords for QC-MDPC codes. A significant step towards a better understanding of BF decoding has been done in [9]. In that work, the authors argue that, especially at very low DFR values, the most meaningful events of decoding failures are due to so-called *near-codewords*. A near-codeword is a weight-v error pattern which overlaps precisely with one of the columns of the double-circulant parity-check matrix $\mathbf{H}$. Whenever the error to be corrected has many common positions with a near-codeword, the decoder is likely going to get fooled because counters behave in an unexpected way, e.g., take low values for erred bits and high values for error-free bits. While near-codewords have been first introduced and analyzed in [3], [8], [16], the more recent analysis in [9] explains their impact on the DFR of a BF decoder more quantitatively. In particular, [9] shows that the amount u of intersections between the error vec-

tor and a near-codeword plays a fundamental role in studying the DFR of BF decoders[3].

1.1 Our contribution

In this paper, we show how the knowledge of near-codewords for a specific QC-MDPC code can be leveraged to improve BF decoding.

If we indicate with $h_1, h_2 \in \mathbb{F}_2[x]/(x^r + 1)$ the polynomials associated with the first rows of $\mathbf{H}_1^\top$ and $\mathbf{H}_2^\top$, respectively, then the set of near-codewords is associated with

$$\mathcal{E} = \left\{(x^i \cdot h_1, 0) \;\middle|\; i = 0, \cdots, r-1\right\} \cup \left\{(0, x^i \cdot h_2) \;\middle|\; i = 0, \cdots, r-1\right\}.$$

Notice that $\mathcal{E}$ is easily determined from $\mathbf{H}$.

Our idea exploits the so-called *convergence to near-codewords*, already analyzed in [3], [8], [9], [16]. Basically, when the initial error overlaps with a near-codeword in a large enough number u of positions, at very low DFR values and in the case of a decoding failure, with large probability the decoder is going to flip the bits that are actually error-free but are included in the support of the involved near-codeword. Inheriting the vocabulary settled in [9], we call such bits *suspicious bits*. Moreover, we refer to bits that are in error and overlap with the near-codeword as *bad bits*, while we call *normal bits* those bits in the initial error vector that are not in the support of the near-codeword and *good bits* such bits that are error-free both in the initial error vector and in the associated near-codeword.

We define a weight-t error as a (t, u)-*almost near-codeword*, with $1 \leqslant u \leqslant v$, if the maximum number of overlapping set entries with a near-codeword is u. We refer to a near-codeword having u intersections with a (t, u)-almost near-codeword as a *close near-codeword*. For an error pattern, the close near-codewords are the near-codewords to which the decoder is likely to converge in case of a decoding failure. See Figure 1 for a representation of the four types of bits, for an almost near-codeword with toy parameters.

For a (t, u)-almost near-codeword, counters for bad and suspicious bits can be (roughly) approximated as $v - u + 1$ and u, respectively. Then, when u is large enough (say, $u > v/2$), it is very likely that bad bits are (incorrectly) not flipped and suspicious bits are (incorrectly) flipped. Since the other bits are not involved in the near-codeword, their counters behave normally and those bits are (correctly) flipped by the decoder with large probability. So, we expect that the decoder will converge to the close near-codeword with large probability.

A Markov model for BF-Max. The above reasoning can be made more rigorous through a theoretical model of the decoder as a finite state machine, with the

[3] This value is used, for instance, to model the counters behavior. Moreover, [9] specializes the Markov chain approach introduced in [5] by considering how the value of u affects decoding.

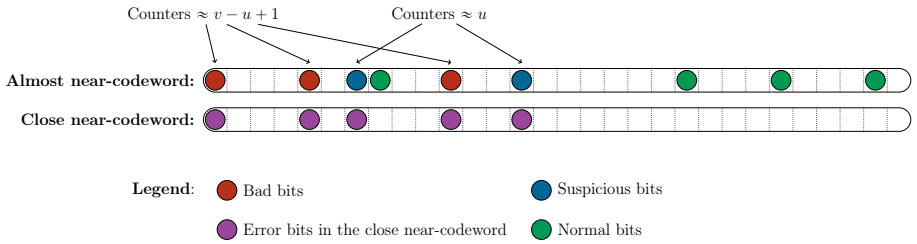


Fig. 1: Example of bad bits, suspicious bits, normal bits and good bits, for a (t, u)-almost near-codeword with $n = 30$, $t = 7$, $u = 3$ and $v = 5$. In the almost near-codeword, the red (bad bits) and green (normal bits) bins represent bits in error, whereas the blue (suspicious bits) and uncolored (good bits) bins represent error-free bits.

state being represented by the syndrome weight, the number of residual errors and the maximum intersection with near-codewords. This has already been done in [9] for a *step-by-step* BF decoder. Inspired by this, we derive a Markov model for BF-Max [18], a recently proposed decoder where a unique bit (one of those with the highest counter value) is flipped in each iteration. We use the Markov model to show that, for BIKE parameters, convergence to near-codewords is expected to be the most relevant event of decoding failures for BF-Max. Even though convergence to near-codewords should be a common behavior for BF decoders, for BF-Max it is even more pronounced. Indeed, since BF-Max performs only one flip in each iteration, it explores states in a more granular way with respect to *out-of-place* decoders (i.e., decoders where multiple bits are potentially flipped in each iteration) and, when failures happen, it progressively moves towards the state corresponding to the near-codeword.

A look-up table to leverage near-codewords. Whenever convergence to a near-codeword happens, then a BF decoder gets stuck in a configuration where the updated syndrome is $\widehat{\mathbf{s}} = (\widehat{\mathbf{e}} + \mathbf{e})\mathbf{H}^\top$, with $\widehat{\mathbf{e}} + \mathbf{e}$ being a near-codeword. We can then recover from this unlucky situation by simply recognizing the associated syndrome and updating the error estimate accordingly. This can be done efficiently by preparing a look-up table where syndromes for near-codewords are stored: at each iteration, it is enough to check whether the current syndrome is included in the table and in that case, flip the bits corresponding to that near-codeword. To assess the effectiveness of this approach, we consider several BF decoders from the literature and show that our technique can indeed reduce the DFR significantly.

Notice that, since all syndromes in the table have weight exactly v, the table must be accessed only when the syndrome weight is exactly equal to v: this observation reduces significantly the amount of table reads. For QC-MDPC codes with cryptographic parameters, having $v = O(\sqrt{r})$, each table read costs $O\big(\sqrt{r} \cdot \log_2^2(r)\big)$ in the worst case. Given that BF decoding comes with a cost of $O\big(r^{1.5} \cdot$

$\log_2(r)\big)$, the modification we propose has a very mild impact on the overall decoder complexity.

As a little drawback, our technique requires $O\big(r^{1.5} \cdot \log_2(r)\big)$ memory and in some contexts this may not be affordable. As a workaround, we can trade memory with time by observing that all syndromes in the table can be generated by cyclic shifts of h_1^2 and h_2^2. Exploiting this, we describe how our technique can be implemented using no memory and (worst case) time for each table access given by $O\big(r \cdot \log_2(r)\big)$ which, again, is significantly lower than the time required for BF decoding.

BF-Max in the rumble of decoders. For BIKE parameters, we compare BF-Max with the BGF [11] and BIKE-Flip [15] decoders, which have been employed within the NIST competition[4]. We show that, among the three considered decoders, BF-Max is the one having the lowest DFR, despite a comparable computational complexity. Remarkably, BF-Max is also inherently constant time, since it flips a unique bit in each iteration.

Simulations show that, for the waterfall region, BF-Max outperforms both BGF and BIKE-Flip. As far as we can tell, BF-Max appears to be also the decoder with the lowest DFR in the floor region. In particular, while for BGF and BIKE-Flip our technique leads to an almost negligible improvement, for BF-Max the reduction in the DFR is much more significant.

The DFR of BIKE. Even though we do not provide a precise estimate for the DFR of BIKE, we are able to give a lower bound relying on a mixture of combinatorial arguments and simulations. For BF-Max and for its improved version, the bound is much lower than the bounds for BGF and BIKE-Flip. Unfortunately (or, perhaps, rather fortunately) the lower bound for the improved BF-Max may be vague since, for this decoder, experiencing decoding failures through numerical simulations has been extremely difficult[5]. Still, all the results we have obtained so far hint at the fact that the improved BF-Max we present in this paper has a remarkably low DFR, much lower than all other decoders from the literature.

Sums of near-codewords. We also consider more convoluted events of failure, such as error patterns where decoding is prone to converge to a sum of near-codewords. We initiate the study of such errors by introducing the notions of *similarity* and *profiles*. The similarity between an error vector and a near-codeword is the number of their overlapping ones, while the profile is the collection of all similarities, given a number of near-codewords. We argue that BF decoding is prone to converge to the near-codewords having a large similarity

[4] BIKE-Flip has replaced BGF in the last round of the competition.

[5] The tightness of the bound depends on the possibility to estimate, through numerical simulations, failure probabilities for the decoding of (t, u)-almost near-codewords. As we will see later on, in all the simulations we have run for the improved BF-Max for BIKE parameters, we have experienced only one event of failure.

with the error vector: if there are multiple such near-codewords, then the decoder likely converges to their sum. With the help of numerical simulations, we show that, for the improved BF-Max, these errors are likely the main responsible for decoding failures in the floor. Then, again, we can leverage the knowledge about near-codewords to recover also from such configurations and, thus, further reduce the DFR. In this case, however, a solution as simple as a look-up table cannot be used in practice, because it would require a huge amount of memory. We sketch some ideas on how to implement such a feature and leave this investigation for future work.

1.2 Paper organization

In Section 2 we settle the notation we use throughout the paper and recall useful concepts about QC-MDPC codes and BF decoding. We also recall the vocabulary from [9] and formally define near-codewords and almost near-codewords. In Section 3, using arguments similar to those presented in [9], we model the behavior of BF-Max and justify its convergence to near-codewords in the events of failure. Then, in Section 4, we revisit the analysis of how counters for almost near-codewords behave: this will be fundamental to motivate our tweaked decoder, which we introduce in the same section. Numerical results about toy codes (QC-MDPC codes with circulant size $r = 2003$ and weight $v \in \{9,\ 11,\ 13,\ 15\}$) are given in the same section. In Section 5 we focus on the floor, describe how the DFR in the floor region can be approximated and evaluate the impact on the floor region for toy codes. In Section 6, we focus on BIKE parameters for NIST category 1 and evaluate the impact of our techniques on the BGF and BIKE-Flip decoders, as well as on the BF-Max. Finally, sums of near-codewords are introduced and studied in Section 7, while Section 8 provides some conclusive remarks.

All the source code we have employed for this paper, as well as raw data from numerical simulations, can be found at the following public repository:

`https://github.com/secomms/BF_near_codewords`

1.3 Relation with previous versions of this work

The present paper significantly extends the conference version [19] with new details and contents, such as novel theoretical results, complete proofs and more details about numerical simulations. Notice that Sections 5 and 7 are completely new compared to the conference paper. Hence, even the DFR lower bounds for the BIKE parameters we provide in Section 6 are missing in the conference paper. Analogously, the time/memory trade-offs we discuss in this paper are not provided in [19].

Also notice that, in the conference version, the Markov model for BF-Max was not present and the convergence to near-codewords was explained with intuitive arguments and approximations. In this paper, we make it more quantitative

through the theoretical characterization of BF-Max, which is a contribution of independent interest.

We remark that, during the preparation of this work, we have repeated numerical simulations for the decoding of (t, u)-almost near-codewords for BIKE parameters. By doing this, we have been able to experience decoding failures for the improved BF-Max; actually, we have just been able to see one decoding failure for all the simulations we have run (10^8 decoding instances for each value of u). Further simulations are still underway, and their results may be used in revised versions of this paper.

2 Preliminaries

In this section, we first settle the notation we will use throughout the paper and then recall background concepts about MDPC codes and the BF decoder.

2.1 Notation

The binary finite field is denoted as $\mathbb{F}_2$. Vectors (resp., matrices) are denoted with bold lowercase (resp., uppercase) letters, e.g., $\mathbf{a}$ is a vector and $\mathbf{A}$ is a matrix. An all-zero vector is indicated as $\mathbf{0}$. For a vector $\mathbf{a}$, $\mathrm{Supp}(\mathbf{a})$ denotes its support (the set of indices of non null coordinates), while $\mathrm{wt}(\mathbf{a})$ stands for the Hamming weight. For a set $\mathcal{A}$ (say, a set of vectors), we write $\mathbf{a} \xleftarrow{\$} \mathcal{A}$ if $\mathbf{a}$ is sampled uniformly at random from $\mathcal{A}$.

Circulant matrices and polynomials. A square matrix is said to be *circulant* if every row can be obtained by applying a cyclic shift to the previous row; the same holds for the columns. Then, an $r \times r$ circulant matrix has the following structure:

$$\mathbf{A} = \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{r-2} & a_{r-1} \\ a_{r-1} & a_0 & a_1 & \cdots & a_{r-3} & a_{r-2} \\ a_{r-2} & a_{r-1} & a_0 & \cdots & a_{r-4} & a_{r-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_{r-1} & a_0 \end{pmatrix}.$$

In a circulant matrix, all rows and columns have the same Hamming weight, which we will sometimes refer to as *circulant weight*.

The set of circulant matrices, equipped with the standard matrix sum and multiplication, forms a ring which is isomorphic to the ring $\mathbb{F}_2[x]/(x^r + 1)$. In other words, let φ indicate the map that, on input a circulant matrix with first row $(a_0, \cdots, a_{r-1})$, returns the polynomial $\sum_{i=0}^{r-1} a_i x^i \in \mathbb{F}_2[x]/(x^r + 1)$. Then, for any two circulant matrices $\mathbf{A}, \mathbf{B}$, it holds that

$$\varphi(\mathbf{A}) \cdot \varphi(\mathbf{B}) = \varphi(\mathbf{A} \cdot \mathbf{B}), \qquad \varphi(\mathbf{A}) + \varphi(\mathbf{B}) = \varphi(\mathbf{A} + \mathbf{B}).$$

Any polynomial $\sum_{i=0}^{r-1} a_i x^i$ can be represented as the vector $(a_0, \cdots, a_{r-1})$. For this reason, we will sometimes be rather relaxed with the notation and interchangeably speak of vectors, circulant matrices and polynomials. For instance, the support of a polynomial $\sum_{i=0}^{r-1} a_i x^i$ is defined naturally by first associating it with the vector $(a_0, \cdots, a_{r-1})$ and then taking its support. In other words, the polynomial with support S is $\sum_{i \in S} x^i$.

2.2 QC-MDPC codes

A binary *linear code* $\mathscr{C} \subseteq \mathbb{F}_2^n$ with redundancy r and length n, with $r \leqslant n$, is a linear subspace of $\mathbb{F}_2^n$ with dimension $k = n - r$. The code rate is $k/n = 1 - r/n$. As a compact representation for a code, one can use the *parity-check matrix*, which is an $r \times n$ matrix with rank r and which serves as a basis for the null space of the code, that is,

$$\mathscr{C} \coloneqq \left\{ \mathbf{c} \in \mathbb{F}_2^n \;\middle|\; \mathbf{H}\mathbf{c}^\top = \mathbf{0} \right\}.$$

Every row of this matrix gives a *parity-check equation*. For a vector $\mathbf{e} \in \mathbb{F}_2^n$, its *syndrome* is $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$. If $\mathbf{s} \neq \mathbf{0}$, then $\mathbf{e} \notin \mathscr{C}$: the only vectors having null syndromes are exactly the codewords.

LDPC codes constitute a family of error correcting codes which can be represented using only sparse parity-check equations. In other words, for an LDPC code, there exists at least one parity-check matrix $\mathbf{H}$ which contains a very small number of ones. MDPC codes are close relatives of LDPC codes, in the sense that they can be thought of as LDPC codes with a somewhat larger density. Apart from this difference, they essentially belong to the same family of codes and, as such, share all relevant properties and features, as well as the same decoding strategies.[6]

A code is said to be QC if it is invariant under block-wise cyclic shifts. In this paper we consider only a special class of QC codes, which we introduce next.

Definition 1 (Double circulant codes). *Let* $\mathbf{H} = (\mathbf{H}_1, \mathbf{H}_2) \in \mathbb{F}_2^{r \times 2r}$*, with* $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *being* $r \times r$ *binary circulant matrices. Let* $\mathscr{C}$ *be the code having* $\mathbf{H}$ *as parity-check matrix. Then, we say that* $\mathscr{C}$ *is a* double circulant code*.*

When both $\mathbf{H}_1$ and $\mathbf{H}_2$ have weight $v \ll r$, then we speak of *double circulant QC-MDPC codes*.

Polynomial view. We will sometimes write $\mathbf{H} = (h_1, h_2)$, where $h_1, h_2 \in \mathbb{F}_2[x]/(x^r + 1)$ are the polynomials associated to the first columns of $\mathbf{H}_1$ and $\mathbf{H}_2$, respectively. This becomes especially useful when one wants to express the syndrome using polynomials. Indeed, for $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^{2r}$ and e_1 and e_2 being the polynomials associated to $\mathbf{e}_1$ and $\mathbf{e}_2$, respectively, we have that the syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$ corresponds to the polynomial

$$s = h_1 \cdot e_1 + h_2 \cdot e_2.$$

[6] Some authors actually distinguish LDPC codes from MDPC codes by saying that parity-check equations for LDPC codes have weight which grows as $O\big(\log_2(n)\big)$, while for MDPC codes the weight grows as $O\big(\sqrt{n}\big)$.

Cryptographic QC-MDPC codes We refer to double circulant QC-MDPC codes employed in BIKE, in short, as cryptographic QC-MDPC codes. Asymptotically, for this family of codes one has $v = O(\sqrt{r})$, with the number of errors to be corrected being $t = O(\sqrt{r})$. The recommended parameters for BIKE are shown in Table 1. Throughout this paper, we focus on the Category 1 parameters.

Table 1: Parameters of the QC-MDPC codes employed in BIKE for the three NIST security levels.

NIST category	r	v	t
1	12 323	71	134
3	24 659	103	199
5	40 973	137	264

2.3 Bit Flipping decoding

The BF is a family of rather simple decoders for LDPC codes. Every BF decoder is based on a basic principle, that is, flip the bits that are more likely to be wrong. This criterion is based on the so-called *counters*.

Definition 2 (Counters). *Let $\mathbf{H} \in \mathbb{F}_2^{r\times n}$ be a parity-check matrix and $\mathbf{s} \in \mathbb{F}_2^r$ a syndrome. Then, for $i \in \{0, \cdots, n-1\}$, the i-th counter is defined as*

$$\sigma_i = |\{j \in \{0, \cdots, r-1\} \mid (h_{j,i} = 1) \wedge (s_j = 1)\}|,$$

where $h_{j,i}$ is the entry of $\mathbf{H}$ in the j-th row and i-th column, and s_j is the j-th entry of $\mathbf{s}$.

In BF decoding, counters are used to determine which positions are affected by errors: discriminating high counters from low counters gives us a (heuristic) very simple criterion to locate error positions. This is at the basis of any BF decoder. More precisely, BF decoding refers to a class of decoders based on the common procedure which is sketched in Algorithm 1.

The algorithm starts with an initially null error estimate $\widehat{\mathbf{e}}$. Counters are used to determine which positions of $\widehat{\mathbf{e}}$ are likely to be wrongly estimated: bits having counters which are large enough are deemed as wrong and get flipped (later on, we will precisely describe how bits to be flipped are selected, for several BF variants). The syndrome is updated accordingly as $\mathbf{s} + \mathbf{H}\widehat{\mathbf{e}}^\top$.

Remark 1. For cryptographic QC-MDPC codes, exploiting sparsity for counters computations and syndrome updates, the cost of BF decoding can be estimated as $O\big(r^{1.5} \cdot \log_2(r)\big)$ (see, e.g., [18, Section II.B]).

Algorithm 1: BF

Data: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, maximum number of iterations `IterMax` $\in \mathbb{N}$
Input: syndrome $\mathbf{s} \in \mathbb{F}_2^r$
Output: decoding failure $\perp$, or vector $\widehat{\mathbf{e}} \in \mathbb{F}_2^n$ such that $\mathbf{s} = \widehat{\mathbf{e}}\mathbf{H}^\top$

1 Set $\widehat{\mathbf{e}} = \mathbf{0}$;// All-zero vector of length n
2 Set `Iter` $= 0$;// Number of performed iterations
3 **while** $(\mathbf{s} \neq \mathbf{0}) \wedge (\texttt{Iter} < \texttt{IterMax})$ **do**
  /* Compute counters */
4 **for** $i = 0, \cdots, n-1$ **do**: compute σ_i;
  /* Find positions that are likely to be wrongly estimated */
5 Determine $J \subseteq \{0, \cdots, n-1\}$ so that, for each $i \in J$, σ_i is large enough;
  /* Update syndrome and error estimate */
6 **for** $i \in J$ **do**
7 $\widehat{e}_i \leftarrow \widehat{e}_i + 1$;
8 $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{h}_i$;
  /* Update number of performed iterations */
9 `Iter` $\leftarrow$ `Iter` $+ 1$;
/* Return failure if syndrome is not null */
10 **if** $\mathbf{s} \neq \mathbf{0}$ return $\perp$, **else** return $\widehat{\mathbf{e}}$;

2.4 BF variants

By *Out-of-Place* decoders, we refer to a family of decoders where, in each iteration, multiple bits can be flipped. In this type of decoders, the indices of the bits to be flipped (line 5 in Algorithm 1) are selected through a threshold criterion: when using a threshold $T \in \{1, \cdots, v\}$, the set J is determined as

$$J = \{i \in \{0, \cdots, n-1\} \mid \sigma_i \geqslant T\}.$$

We consider several BF variants which differ on how the threshold is selected:

- OoP: a standard Out-of-Place decoder where the threshold is a function only of `Iter` (the iteration number);
- MLD: the majority-logic decoder, where all iterations use the same threshold $T = \lceil v/2 \rceil$;
- BGF: the decoder proposed in [11] and used by BIKE in the first rounds of the NIST competition. In this decoder, the threshold selection rule is a function of the syndrome weight $w = \mathrm{wt}(\mathbf{s})$. In particular, for Level 1 BIKE parameters, the threshold is

$$T = \max\lfloor(0.0069722 \cdot w + 13.530\rfloor, 36).$$

  Notice that, during the first iteration, the decoder actually does two extra iterations, looking only at subgroups of bits (which, in[11], are called black

and gray bits) and with thresholds which may vary from the above. We do not enter into further details and refer the interested reader to [11]; in any case, it is easily seen that even this decoder is encompassed by the template in Algorithm 1 (by modeling these extra iterations as standard BF iterations with an ad-hoc rule to determine J). The performs a maximum of 5 iterations;

- BIKE-Flip: described in version 5.2 of the BIKE specification [15], it has been used by BIKE in the last round of the NIST competition. In this decoder, $\texttt{IterMax} = 7$ and the threshold is chosen as a function of both the current syndrome weight w, the weight of the initial syndrome (which we indicate as w_0), and the number of the current iteration `Iter`. Namely,

$$T = \max\big(f(w),\, F_{\texttt{Iter}}(w_0)\big),$$

where for Level 1 BIKE parameters we have

$$\begin{aligned}
f(w) &= 0.006254868353074983 \cdot w + 11.10143233724,\\
F_1(w_0) &= f(w_0) + 3,\\
F_2(w_0) &= \frac{1}{3}\left(2 \cdot f(w_0) + \frac{v+1}{2}\right) + 3,\\
F_3(w_0) &= \frac{1}{3}\left(f(w_0) + 2 \cdot \frac{v+1}{2}\right) + 3,\\
F_{\texttt{Iter}} &= \frac{v+1}{2} + 3, \quad \texttt{Iter} \geqslant 4.
\end{aligned}$$

Among decoders that are not out-of-place, we focus on BF-Max, introduced in [18]. In this BF variant, each iteration flips a unique bit, chosen so that its counter is maximum. The intuition behind this strategy is simple: in fact, by flipping the largest counter bit, we are heuristically reducing to its minimum the probability of flipping a wrong bit. If more than one bit shares the same, highest counter value, we randomly select one of them and flip it. Namely, in this case $J = \{i^*\}$ with

$$i^* \xleftarrow{\$} \{i \in \{0, \cdots, n-1\} \mid \sigma_{i^*} \geqslant \sigma_j,\, \forall j \neq i^* \in \{0, \cdots, n-1\}\}\,.$$

BF-Max must perform many more iterations that out-of-place decoders (obviously, at least t iterations are necessary to decode successfully). This would lead to rather large complexity for a schoolbook implementation. As shown in [18], the large number of iterations can be countered with a clever implementation exploiting sparsity. In particular, for cryptographic parameters, this implementation leads to a complexity of $O\big(r^{1.5} \cdot \log_2(r)\big)$, which is in the same order of the out-of-place BF.

2.5 Near-codewords

Trapping sets are special error vectors that lead to a syndrome and counters having a somewhat unexpected behavior, e.g., counters of error positions having

very low values. An example of trapping sets for double circulant MDPC codes is that of *near-codewords*.

Definition 3 (Near-codewords). *Let* $\mathbf{H} = (h_1, h_2)$ *be the parity-check matrix of a double circulant code; then the set of near-codewords is*

$$\mathcal{E} = \left\{(x^i \cdot h_1, 0) \;\middle|\; i = 0, \cdots, r-1\right\} \cup \left\{(0, x^i \cdot h_2) \;\middle|\; i = 0, \cdots, r-1\right\}.$$

Observe that, for every error vector in $\mathcal{E}$, the syndrome has weight v. For instance, near-codewords of the form $(x^i \cdot h_1, 0)$ have syndrome $x^i \cdot h_1^2 + h_2 \cdot 0 = x^i \cdot h_1^2$; analogously, the syndrome of near-codewords of the form $(0, x^i \cdot h_2)$ is given by $x^i \cdot h_2^2$.

As observed in [9], near-codewords are the culprits for many decoding failures, especially at very low DFR values. Moreover, [9] shows that the decoding failure probability has a strong dependence on the number u of intersections between the initial error vector and near-codewords. We formalize this property by introducing the notion of *almost near-codewords*.

Definition 4 (Almost near-codewords). *For a parity-check matrix* $\mathbf{H}$ *with* $\mathcal{E}$ *being the set of near-codewords, for any pair of integers such that* $t \geqslant u$*, let* $\mathcal{E}_{t,u} \subseteq \mathbb{F}_2^n$ *be the set of weight-*t *vectors such that, for each* $\mathbf{e} \in \mathcal{E}_{t,u}$*,*

$$\max_{\mathbf{m} \in \mathcal{E}} \{|\mathrm{Supp}(\mathbf{e}) \cap \mathrm{Supp}(\mathbf{m})|\} = u.$$

We refer to elements of $\mathcal{E}_{t,u}$ *as* (t, u)*-almost near-codewords.*

Remark 2. For a (t, u)-almost near-codeword $\mathbf{e}$ there exists at least one near-codeword $\mathbf{m} \in \mathcal{E}$ which overlaps with $\mathbf{e}$ in *exactly* u ones; for each of the other near-codewords, the overlapping ones with $\mathbf{e}$ are *at most* u.

Definition 5 (Close near-codeword). *Let* $\mathbf{e}$ *be a* (t, u)*-almost near-codeword and* $\mathbf{m} \in \mathcal{E}$ *such that* $|\mathrm{Supp}(\mathbf{e}) \cap \mathrm{Supp}(\mathbf{m})| = u$*. Then, we say that* $\mathbf{m}$ *is a* close *near-codeword for* $\mathbf{e}$*.*

Remark 3. Notice that for a double circulant QC-MDPC code with column weight v, (t, u)-almost near-codewords exist only for $u \leqslant v$.

We finally recall the notion of *bad bits* and *suspicious bits*, introduced in [9].

Definition 6 (Bad bits, suspicious bits, normal bits and good bits). *Let* $\mathbf{e} \in \mathcal{E}_{t,u}$ *be a* (t, u)*-almost near-codeword. Then, we say that*

- *the* j*-th bit is a bad bit if* $j \in \mathrm{Supp}(\mathbf{e})$ *and there is a close near-codeword* $\mathbf{m}$ *such that* $j \in \mathrm{Supp}(\mathbf{m}) \cap \mathrm{Supp}(\mathbf{e})$*;*
- *the* j*-th bit is a suspicious bit if* $j \notin \mathrm{Supp}(\mathbf{e})$ *and there is a close near-codeword* $\mathbf{m}$ *such that* $j \in \mathrm{Supp}(\mathbf{m}) \backslash \mathrm{Supp}(\mathbf{e})$*.*
- *the* j*-th bit is a normal bit (in error) if* $j \in \mathrm{Supp}(\mathbf{e})$ *and there is a close near-codeword* $\mathbf{m}$ *such that* $j \in \mathrm{Supp}(\mathbf{m}) \backslash \mathrm{Supp}(\mathbf{e})$*.*
- *the* j*-th bit is a good bit if* $j \notin \mathrm{Supp}(\mathbf{e})$ *and there is a close near-codeword* $\mathbf{m}$ *such that* $j \notin \mathrm{Supp}(\mathbf{m})$*.*

Notice that when a (t, u)-almost near-codeword admits only one close near-codeword, then the number of bad bits is exactly u and the number of suspicious bits is exactly $v-u$. Still, it may happen that for an almost near-codeword there is more than one close near-codeword, hence the number of bad and suspicious bits may be higher. However, when u is large enough, we expect that for (t, u)-almost near-codewords there is only one close near-codeword.

Remark 4. From Definition 4 it follows that every error vector of weight t is a (t, u)-almost near-codeword for some $u \geqslant 1$. In other words, the sets of almost near-codewords provide a partition of the Hamming sphere with radius t.

Example 1. Let $r = 251$ and $v = 7$; let $\mathscr{C}$ be the double circulant QC-MDPC code defined by

$$h_1 = x^{50} + x^{55} + x^{65} + x^{68} + x^{107} + x^{153} + x^{203},$$

$$h_2 = x^{6} + x^{17} + x^{26} + x^{48} + x^{91} + x^{97} + x^{187}.$$

Let $t = 9$ and $\mathbf{e}$ with support

$$\{40,\ 70,\ 169,\ 183,\ 186,\ 250,\ 437,\ 447,\ 458\}.$$

In polynomial view, the error vector is

$$\mathbf{e} = \left(x^{40} + x^{70} + x^{169} + x^{183} + x^{186} + x^{250},\ \ x^{186} + x^{196} + x^{207}\right).$$

This is a (t, u)-almost near-codeword for $u = 3$ and there is a unique close near-codeword, corresponding to $\mathbf{m} = (x^{118} \cdot h_1,\ 0)$. The overlapping positions between $\mathbf{m}$ and $\mathbf{e}$ are $\{70,\ 183,\ 186\}$.

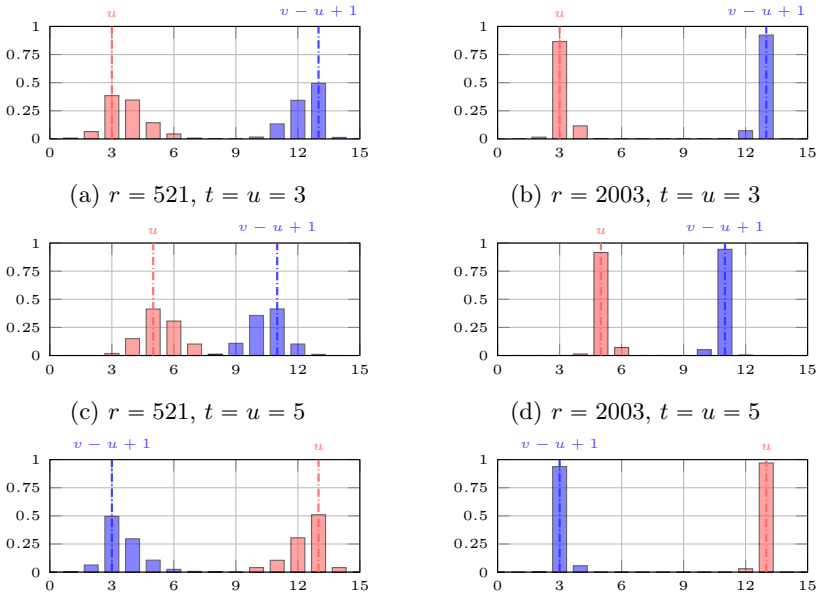
Let us now consider, instead, the vector $\mathbf{e}$ with support

$$\{46,\ 86,\ 255,\ 308,\ 314,\ 329,\ 350,\ 474,\ 489\}.$$

In this case, there are two close near-codewords for $u = 3$: $(0,\ x^{302} \cdot h_2)$ and $(0,\ x^{468} \cdot h_2)$.

3 Convergence to near-codewords for BF-Max

In this section, we model the behavior of BF-Max as a Markov process. Namely, we take the statistical distribution of counters already introduced in [9] and compute the flipping probabilities for the BF-Max decoder. As in [9], we model the decoder as a finite state machine, with the state being identified by the syndrome weight, the number of residual errors and the maximum intersection with near-codewords. Starting from a particular state, the reachable states depend on which type of bit is flipped. Using this model, we give a theoretical explanation of why, when a decoding failure happens, the BF-Max decoder has with high probability converged to a near-codeword. Even though this modeling is general, we only consider in our numerical experiments the case of BIKE parameters.



(a) $r = 521$, $t = u = 3$ (b) $r = 2003$, $t = u = 3$

(c) $r = 521$, $t = u = 5$ (d) $r = 2003$, $t = u = 5$

(e) $r = 521$, $t = u = 13$ (f) $r = 2003$, $t = u = 13$

Fig. 2: Experimental distribution of counters for bad bits (in blue) and suspicious bits (in red), for codes with $v = 15$ and different circulant sides. For each configuration, the distribution is estimated by generating 100 uniformly random (t, u)-almost near-codewords.

3.1 Intuitive arguments

Recall that, as shown in [19], for (t, u)-almost near-codewords with $t = u$ with extremely large r (i.e., very sparse parity-check matrices) we have

$$\sigma_i \approx \begin{cases} v - u + 1 & \text{if } i \text{ is a bad bit,} \\ u & \text{if } i \text{ is a suspicious bit.} \end{cases}$$

In practice, bad and suspicious bits have some statistical distributions with mean values which we expect to be close to $v - u + 1$ and u, respectively. As matrices get sparser, the distributions shall become more and more concentrated around the mean values. See Figure 3 for a validation of this fact through numerical simulations.

When u is large enough (say, $u > v/2$), we expect that with high probability many suspicious bits have counters which are larger than those of bad bits. Then, it is rather likely that a BF decoder flips such suspicious bits: this starts a sort of avalanche effect so that, in the next iterations, the other suspicious bits will have, with larger probability, high counters and they will be flipped, as well. If $\mathbf{m}$ was the unique close near-codeword for the initial error vector, then $\widehat{\mathbf{e}} + \mathbf{e}$ converges to $\mathbf{m}$.

When considering a (t,u)-almost near-codewords with $t > u$, with respect to the case of $t = u$ that we have analyzed before, in this case we have more errors, so overall the decoder behavior should not be better. We expect that the decoder will, with large probability, flip first all the normal bits until the vector $\widehat{\mathbf{e}} + \mathbf{e}$ of residual errors form a (u,u)-almost near-codeword. Then, decoding will proceed as we have already explained. Obviously, in such a case, things may even get worse and the decoder may end up making wrong flips before hitting a (u,u)-almost near-codewords: this will increase the number of residual errors and, with high probability, also the maximum intersection with near-codewords. As we shall see later on, this is further confirmed by both the theoretical model and numerical simulations.

3.2 Statistical distribution of counters

We recall the model and heuristics from [9, Section 5].

Proposition 1. *Let $\mathbf{e} \in \mathbb{F}_2^n$ be a random (t,u)-almost near-codeword. Let $w = \mathrm{wt}\left(\mathbf{e}\mathbf{H}^\top\right)$ and $(\sigma_1, \cdots, \sigma_n)$ indicate the counters associated with $\mathbf{e}$. Moreover, let $X = \left(\sum_{j \in \mathrm{Supp}(\mathbf{e})} \sigma_j\right) - w$. Then, the expected value of X is*

$$\mathbb{E}[X \mid w,t,u] = w \cdot \frac{\sum_\ell 2\ell \cdot \rho_{2\ell+1}}{\sum_\ell \rho_{2\ell+1}},$$

where ρ_x is defined as

$$\sum_{j=0}^{x} \binom{t-u}{j} \binom{u}{x-j} \left(\frac{v}{r}\right)^j \left(\frac{r-v}{r}\right)^{t-u-j} \cdot \left(\frac{v+1-u}{r}\right)^{x-j} \left(\frac{r-v-1+u}{r}\right)^{u-x+j}.$$

Heuristic 1 *For a syndrome with weight w let,*

$$\pi_0 = \frac{w}{r}, \qquad \pi_1 = \frac{w + \xi \cdot \mathbb{E}[X \mid w,t,u]}{w_{\max}}, \qquad w_{\max} = v \cdot t - 2\binom{u}{2},$$

where $\xi = 0.955$. Let $\mathcal{B}_{z,p}$ be the binomial distribution with parameters z and p:

$$\mathcal{B}_{z,p}(x) = \binom{z}{x} p^x (1-p)^{z-x}.$$

Then, the probability that a counter σ_j has value σ is

$$\begin{cases} \mathcal{D}_{\mathrm{g}}(\sigma) \coloneqq \mathcal{B}_{v,\pi_0}(\sigma) & \text{if } j \text{ is a good bit,} \\ \mathcal{D}_{\mathrm{n}}(\sigma) \coloneqq \mathcal{B}_{v,\pi_1}(\sigma) & \text{if } j \text{ is a normal bit,} \\ \mathcal{D}_{\mathrm{s}}(\sigma) \coloneqq \sum_{c=0}^{\sigma} \mathcal{B}_{v-u,\pi_0}(c) + \mathcal{B}_{u,\pi_1}(\sigma - c) & \text{if } j \text{ is a suspicious bit,} \\ \mathcal{D}_{\mathrm{b}}(\sigma) \coloneqq \sum_{c=0}^{\sigma} \mathcal{B}_{v-u+1,\pi_1}(c) + \mathcal{B}_{u-1,\pi_0}(\sigma - c) & \text{if } j \text{ is a bad bit.} \end{cases}$$

Let:

- $N_{\mathrm{g}} = 2r - v - (t-u)$*: number of good bits;*
- $N_{\mathrm{b}} = u$*: number of bad bits;*
- $N_{\mathrm{s}} = v - u$*: number of suspicious bits;*
- $N_{\mathrm{n}} = t - u$*: number of normal bits.*

An example of the resulting distributions for BIKE parameters, for different values of u, is shown in Figure 3. We see that, for normal bits, the distribution is always biased towards high values: with large probability, normal bits are flipped by the decoder when $t > u$. As u gets larger, the distribution for bad bits shifts towards lower values while that for suspicious bits shifts towards high values. For instance, for $u = 45$, suspicious bits tend to have larger values than bad bits but, still, lower than those of normal bits.

To get the syndrome weights for the theoretical distributions, we have sampled (t, u)-almost near-codewords. Even though this leads to some variation in the distributions, we see that the fluctuations are very mild: the behavior of the distributions depends strongly in the value of u and only marginally on w.

3.3 Flipping probabilities

For the BF-Max decoder, we want to compute the probabilities that the decoder flips a bit belonging to one of the four groups. We just give such a probability for the case of a normal bit, for the other bits the expressions are analogous.

Proposition 2. *Under Heuristic 1, the probability that BF-Max flips a normal bit with counter value σ, when decoding a (t, u)-almost near-codeword with syndrome weight w, is*

$$p_{\mathrm{n}}^{(t,w,u)}(\sigma) = q_{\mathrm{n}}(\sigma) + d_{\mathrm{n}}(\sigma),$$

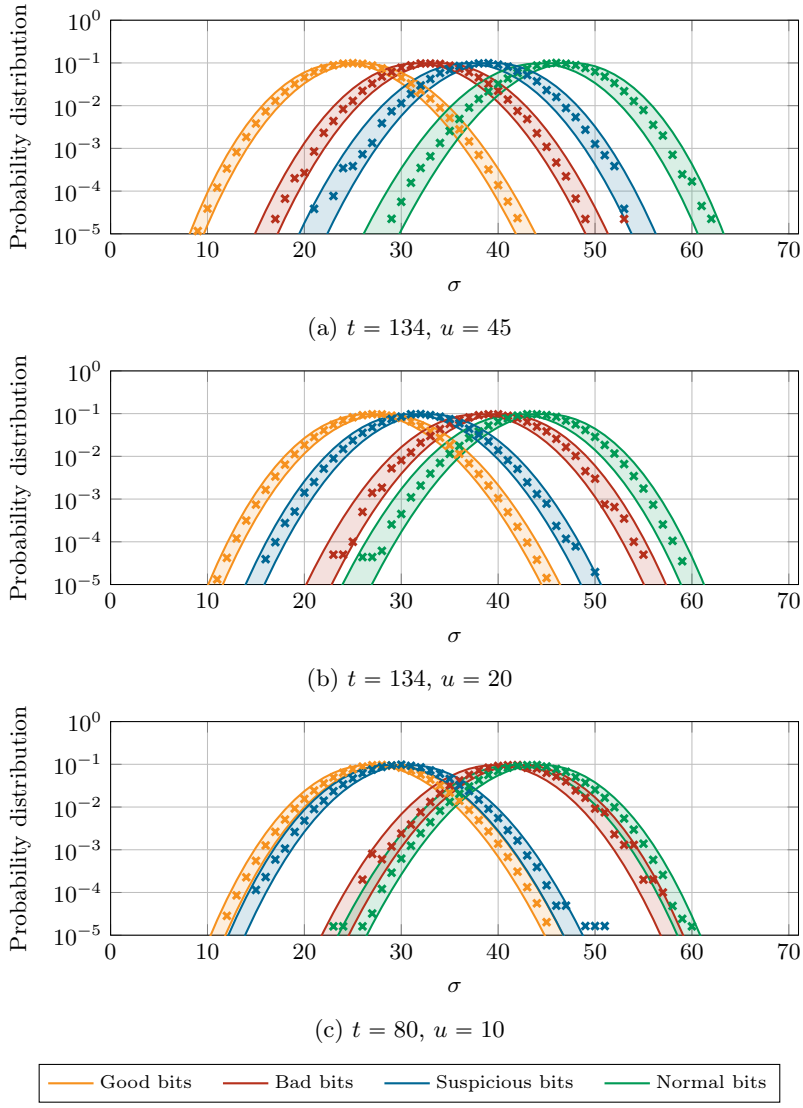
where

$$q_{\mathrm{n}}(\sigma) = \left(\left(\sum_{c=0}^{\sigma} \mathcal{D}_{\mathrm{n}}(c) \right)^{N_{\mathrm{n}}} - \left(\sum_{c=0}^{\sigma-1} \mathcal{D}_{\mathrm{n}}(c) \right)^{N_{\mathrm{n}}} \right) \cdot \left(\sum_{c=0}^{\sigma-1} \mathcal{D}_{\mathrm{g}}(c) \right)^{N_{\mathrm{g}}} \cdot \left(\sum_{c=0}^{\sigma-1} \mathcal{D}_{\mathrm{s}}(c) \right)^{N_{\mathrm{s}}} \cdot \left(\sum_{c=0}^{\sigma-1} \mathcal{D}_{\mathrm{b}}(c) \right)^{N_{\mathrm{b}}},$$

$$d_{\mathrm{n}}(\sigma) = \sum_{\substack{1 \leqslant z_{\mathrm{n}} \leqslant N_{\mathrm{n}} \\ 0 \leqslant z_{\mathrm{g}} \leqslant N_{\mathrm{g}} \\ 0 \leqslant z_{\mathrm{b}} \leqslant N_{\mathrm{b}} \\ 0 \leqslant z_{\mathrm{s}} \leqslant N_{\mathrm{s}} \\ z_{\mathrm{g}} + z_{\mathrm{b}} + z_{\mathrm{s}} \geqslant 1}} \frac{z_{\mathrm{n}}}{z_{\mathrm{n}} + z_{\mathrm{g}} + z_{\mathrm{b}} + z_{\mathrm{s}}} \cdot g_{\mathrm{n}}(\sigma, z_{\mathrm{n}}) \cdot g_{\mathrm{g}}(\sigma, z_{\mathrm{g}}) \cdot g_{\mathrm{b}}(\sigma, z_{\mathrm{b}}) \cdot g_{\mathrm{s}}(\sigma, z_{\mathrm{s}}),$$

with $g_{\mathrm{n}}(\sigma, z_{\mathrm{n}})$ being (analogous definitions hold for the other terms)

$$g_{\mathrm{n}}(\sigma, z_{\mathrm{n}}) = \binom{N_{\mathrm{n}}}{z_{\mathrm{n}}} \left(\mathcal{D}_{\mathrm{n}}(\sigma)\right)^{z_{\mathrm{n}}} \left(\sum_{c=0}^{\sigma-1} \mathcal{D}_{\mathrm{n}}(c) \right)^{N_{\mathrm{n}} - z_{\mathrm{n}}}.$$



(a) $t = 134$, $u = 45$

(b) $t = 134$, $u = 20$

(c) $t = 80$, $u = 10$

Fig. 3: Comparison between the distribution for counters arising from Heuristic 1 and empirical data. The empirical distributions have been generated by sampling, for each parameter set, 10^3 random (t, u)-almost near-codewords. These almost near-codewords have also been used to compute the syndrome weights which are required for the modeling in Heuristic 1. For the resulting theoretical distributions, we show the minimum and maximum values we have found (corresponding to the boundaries of the colored areas).

Proof. We flip a normal bit with counter σ if:

- there is at least a normal bit with counter σ, while the counters for all other types of bits have value $< \sigma$. We indicate with $q_{\mathrm{n}}(\sigma)$ the probability that such an event happens;
- there is at least a normal bit with counter σ, there is at least one bit from the other types of bits with counter σ (but not bit with counter $> \sigma$), and the decoder chooses to flip one of the normal bits with counter σ. We indicate with $d_{\mathrm{n}}(\sigma)$ the probability that such an event happens.

$q_{\mathrm{n}}(\sigma)$ is the product of two terms: $\left(\sum_{c=0}^{\sigma}\mathcal{D}_{\mathrm{n}}(c)\right)^{N_{\mathrm{n}}} - \left(\sum_{c=0}^{\sigma-1}\mathcal{D}_{\mathrm{n}}(c)\right)^{N_{\mathrm{n}}}$, the probability that the normal bits have all counters $\leqslant \sigma$ and there is at least a counter with value σ, and $\left(\sum_{c=0}^{\sigma-1}\mathcal{D}_{\mathrm{g}}(c)\right)^{N_{\mathrm{g}}} \cdot \left(\sum_{c=0}^{\sigma-1}\mathcal{D}_{\mathrm{s}}(c)\right)^{N_{\mathrm{s}}} \cdot \left(\sum_{c=0}^{\sigma-1}\mathcal{D}_{\mathrm{b}}(c)\right)^{N_{\mathrm{b}}}$, which is the probability that for all good, bad and suspicious bits all counters have value lower than σ.

To get d_{n}, we consider all configurations in which the number of counters with value equal to σ is $z_{\mathrm{n}}+z_{\mathrm{g}}+z_{\mathrm{b}}+z_{\mathrm{s}}$, with $z_{\mathrm{n}} \geqslant 1$ and $z_{\mathrm{g}}+z_{\mathrm{b}}+z_{\mathrm{s}} \geqslant 1$. The probability that this configuration happens is $g_{\mathrm{n}}(\sigma, z_{\mathrm{n}})\cdot g_{\mathrm{g}}(\sigma, z_{\mathrm{g}})\cdot g_{\mathrm{b}}(\sigma, z_{\mathrm{b}})\cdot g_{\mathrm{s}}(\sigma, z_{\mathrm{s}})$. Then, we multiply this by the probability that the decoder selects one of the normal bits, which is $\frac{z_{\mathrm{n}}}{z_{\mathrm{n}}+z_{\mathrm{g}}+z_{\mathrm{b}}+z_{\mathrm{s}}}$. Observe tat the configurations where $z_{\mathrm{g}} + z_{\mathrm{b}} + z_{\mathrm{s}} = 0$ are already counted by $q_{\mathrm{n}}(\sigma)$, which is why we are summing over all configurations where at least one of terms z_{g}, z_{b}, z_{s} is not null. □

The probability that a normal bit is flipped follows immediately.

Proposition 3. *Under Heuristic 1, the probability that* BF-Max *flips a normal bit, when decoding a (t, u)-almost near-codeword with syndrome weight w, is*

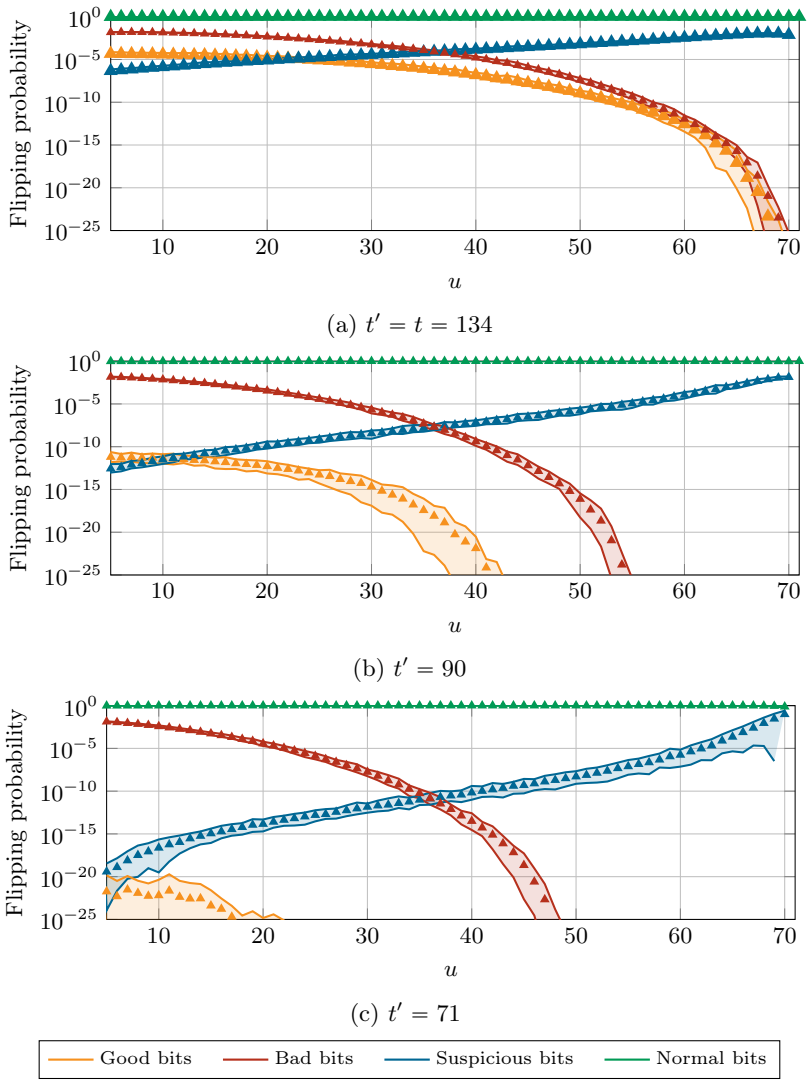
$$\sum_{\sigma=1}^{v} p_{\mathrm{n}}^{(w,t,u)}(\sigma).$$

For the other types of bit, analogous relations (with analogous notation) hold.

3.4 Evidences of convergence

As argued in [18], when the decoder makes only correct flips (in our case, this means it flips normal bits), the residual error vector can be studied as it is drawn uniformly at random. Quantitatively: if the initial error vector is a (t, u)-almost near-codeword, after t^* iterations with t^* low enough (later on, we make this more quantitative), then the vector of residual errors can be treated as a random (t', u)-almost near-codeword with weight $t' = t - t^*$.

In Figure 4 we compare the flipping probabilities for the four types of bits, for several values of t' and each admissible value of u. The syndrome weights have been obtained by sampling (t', u)-almost near-codewords and computing the corresponding syndromes. We have then computed the flipping probabilities as in Proposition 3.



(a) $t' = t = 134$

(b) $t' = 90$

(c) $t' = 71$

Fig. 4: Flipping probabilities for the four types of bit. To get the syndrome weights, we have sampled 100 random (t', u)-almost near-codewords, for each value of u. For the resulting theoretical distributions, we show the minimum and maximum values we have found (corresponding to the boundaries of the colored areas). Marks are associated with the average probabilities.

As we can see in Figure 4, the bits with the highest probability of being flipped are always normal bits. This confirms, once again, that in the first iterations the decoder flips with extremely large probability only normal bits. For low values of u (say, $u < v/2$) bad bits have always a higher chance of being flipped,

with respect to good and suspicious bits. For higher values of u, instead, the probability to flip a suspicious bit gets higher. We notice that, for good bits, the probability of being flipped is always rather low. In particular, this probability decays as u gets higher. This motivates the convergence to near-codewords: when the decoder makes a wrong flip, the probability that this flip is associated with a suspicious bit is higher than the probability that it is associated with a good bit. When a suspicious bit is flipped, the (t', u)-almost near-codeword becomes a $(t'+1, u+1)$-almost near-codeword. Therefore, since the value of u has increased, in the upcoming iterations flipping of suspicious bits becomes more likely, while flipping of good bits becomes less likely. This triggers the avalanche effect leading to the convergence of residual errors to a near-codeword.

In the next section, we show how such considerations can be incorporated into a Markov model to study the behavior of the BF-Max decoder.

3.5 A Markov chain model for BF-Max

Analogously to [9], we denote the state of the decoder with the parameters (w, t, u), where w is the syndrome weight, t the number of errors to be corrected and u the number of common positions between the error vector and the close near-codeword.

Observe that all of these parameters may change during decoding; first, when the decoder flips a bit with counter σ, the syndrome weight is updated as

$$w \quad \to \quad w + v - 2\sigma.$$

The updates on the pair (t, u) depend on which bit is flipped:

$$\begin{aligned}
&\text{good bit:} && (t,u) \longrightarrow (t+1, u),\\
&\text{bad bit:} && (t,u) \longrightarrow (t-1, u-1),\\
&\text{suspicious bit:} && (t,u) \longrightarrow (t+1, u+1),\\
&\text{normal bit:} && (t,u) \longrightarrow (t-1, u).
\end{aligned}$$

The flipping probabilities are given by $p_{\mathrm{g}}(\sigma)$, $p_{\mathrm{b}}(\sigma)$, $p_{\mathrm{s}}(\sigma)$, $p_{\mathrm{n}}(\sigma)$, respectively, computed as in Proposition 3.

Using these probabilities, one can define a Markov model as in [9]. Observe that the flipping probabilities are a function of the starting state $(w,\ t,\ u)$. For instance, the decoder moves from $(w,\ t,\ u)$ to $(w + v - 2\sigma,\ t - 1, u - 1)$ with probability $p_{\mathrm{b}}^{(t,w,u)}(\sigma)$. Then, another bad bit with counter σ is flipped with probability $p_{\mathrm{b}}^{(w+v-2\sigma,t-1,u-1)}(\sigma)$.

Using the Markov chain modeling, we can estimate the probabilities with which the decoder reaches certain states.

Remark 5. We observed that, when both t and u are extremely small (say, $t \leqslant 10$), some of the transition probabilities may have a strange behavior. Analogous issues arise when $t = u = v-1$. Interestingly, these issues happen only for certain syndrome weights. We believe the model in Heuristic 1 is not precise enough to

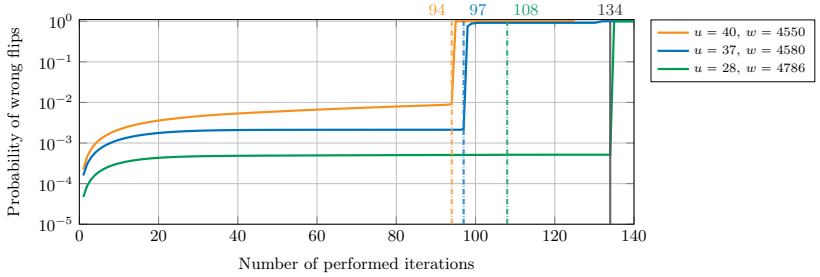


Fig. 5: Probability that the decoder performed only correct flips (i.e., has only flipped bad and/or normal bits), as a function of the number of performed iterations.

deal with such extreme cases, but investigating this further requires a specific in-depth analysis, which we leave for future work. In any case, the behavior of the decoder in such cases is determined: for very small t, it will always correct the errors, while for $t = u = v - 1$ it will converge to the near-codeword. In our implementation of the Markov model, these transitions have been forced.

Remark 6. Observe that we are simplifying the relations between the error vector and the near-codewords, since we are not considering the cases in which there are two or more close near-codewords (this is expected to happen with extremely low probability). The current modeling does not allow for analyzing these events as, for instance, the distributions from 1 take into account only the intersection with a single near-codeword.

Results In our implementation of the model, to avoid storing too many branches, we have pruned paths whenever their probabilities were too low[7]

In Figure 5 we report the probability that the decoder makes wrong flips, as a function of the number of performed iterations. In the figures we also highlight the values of $t-u$, which correspond to the number of normal bits. The syndrome weights have been obtained by sampling actual (t, u)-almost near-codewords for a double circulant QC-MDPC code. Notice that, running the model for different syndrome weights (but same value of u), we get similar results.

As we have already said, we expect that the decoder will first flip the normal bits and then, when the number of normal bits is very low, will start with high probability flipping either bad bits or suspicious bits. The figure confirms this behavior. Indeed, the probability to have wrong flips is always extremely small for the first iterations. For $u = 37$ and $u = 40$, the probability gets very close to 1 as soon as the number of performed iterations exceeds $t - u$. Indeed, after so many iterations, the decoder has (with extremely large probability) flipped all

[7] For large values of u, we have cut paths with probabilities $< 2^{-25}$. For lower values of u (say, $u < v/2$), we tolerated higher probabilities and filtered only paths with probabilities $< 2^{-40}$.

Table 2: Configurations (t, u) reached by the Markov model, for $u = 37$ and $w = 4580$.

After 50 iterations		After $t - u$ iterations		After $t + v - 2u$ iterations	
(t', u')	Pr.	(t', u')	Pr.	(t', u')	Pr.
$(84, 36)$	1.20×10^{-3}	$(37, 36)$	1.20×10^{-3}	$(3, 3)$	5.07×10^{-2}
$(84, 37)$	9.97×10^{-1}	$(37, 37)$	9.97×10^{-1}	$(5, 5)$	3.43×10^{-2}
$(86, 37)$	2.25×10^{-6}	$(39, 37)$	2.26×10^{-7}	$(7, 7)$	1.17×10^{-2}
$(86, 38)$	2.09×10^{-3}	$(39, 38)$	2.13×10^{-3}	$(9, 9)$	3.84×10^{-3}
$(88, 39)$	2.04×10^{-7}			$(11, 11)$	1.26×10^{-3}
				$(13, 13)$	4.18×10^{-4}
				$(15, 15)$	1.39×10^{-4}
				$(17, 17)$	4.60×10^{-5}
				$(19, 19)$	1.51×10^{-5}
				$(21, 21)$	4.75×10^{-6}
				$(23, 23)$	1.34×10^{-6}
				$(25, 25)$	2.54×10^{-7}
				$(47, 47)$	2.40×10^{-7}
				$(49, 49)$	1.28×10^{-6}
				$(51, 51)$	4.59×10^{-6}
				$(53, 53)$	1.46×10^{-5}
				$(55, 55)$	4.46×10^{-5}
				$(57, 57)$	1.35×10^{-4}
				$(59, 59)$	4.06×10^{-4}
				$(61, 61)$	1.23×10^{-3}
				$(63, 63)$	3.78×10^{-3}
				$(65, 65)$	1.18×10^{-2}
				$(67, 67)$	3.88×10^{-2}
				$(69, 69)$	1.43×10^{-1}
				$(71, 71)$	6.63×10^{-1}

normal bits: among the remaining bits in error, suspicious bits are more biased towards high values, with respect to bad bits, and therefore the decoder will flip a suspicious bit with large probability.

For $u = 28$, instead, the probability is always very small, even after $t - u$ iterations have been performed. Indeed, in such a case, bad bits are flipped with probability greater than that of suspicious bits, therefore the more likely flips regard normal and bad bits. Observe that this means also that, with probability very close to 1, the decoder succeeds after exactly 134 iterations.

To look more closely into the behavior of the Markov model, in Tables 2 and 3 we report the probabilities that the decoder ends up in a state where residual errors form a (t', u')-almost near-codeword, after a certain number of iterations. Remember that, in our model, the state is identified by the parameters (w, t, u): we have aggregated the probabilities for all states with the same pair of values (t, u). We give the resulting states in three cases:

- after 50 iterations, to confirm that, in the initial iterations, the decoder performs only correct flips with large probability;
- after $t - u$ iterations, which is the minimum amount of iterations needed for correcting all normal bits;

Table 3: Configurations (t, u) reached by the Markov model, for $u = 28$ and $w = 4786$.

After 50 iterations		After $t-u$ iterations		After $t+v-2u$ iterations	
(t', u')	Pr.	(t', u')	Pr.	(t', u')	Pr.
$(84, 24)$	1.27×10^{-7}	$(28, 24)$	1.52×10^{-7}	$(0, 0)$	≈ 1
$(84, 25)$	8.06×10^{-6}	$(28, 25)$	9.14×10^{-6}		
$(84, 26)$	4.89×10^{-4}	$(28, 26)$	5.24×10^{-4}		
$(84, 27)$	2.65×10^{-2}	$(28, 27)$	2.89×10^{-2}		
$(84, 28)$	9.73×10^{-1}	$(28, 28)$	9.70×10^{-1}		
$(86, 26)$	2.31×10^{-8}	$(30, 26)$	2.08×10^{-8}		
$(86, 27)$	1.36×10^{-6}	$(30, 27)$	1.39×10^{-6}		
$(86, 28)$	4.69×10^{-5}	$(30, 28)$	4.71×10^{-5}		
$(86, 29)$	4.22×10^{-4}	$(30, 29)$	4.22×10^{-4}		
$(88, 29)$	1.95×10^{-8}	$(32, 29)$	1.63×10^{-8}		
$(88, 30)$	1.43×10^{-7}	$(32, 30)$	1.40×10^{-7}		

- after $t+v-2u$ iterations, which is the minimum amount of iterations needed for having the convergence to a near-codeword.

We notice that, after 50 iterations, for all cases and regardless of the initial value of u, the most likely state is the one with $t' = t - 50 = 84$ errors. We also notice that, in the most likely state, we also have $u' = u$: the decoder has flipped only normal bits. Remarkably, we see that other likely states are those with $u' < u$ (implying that the decoder has flipped also bad bits), while states having $t' > 84$ are rather unlikely.

Looking at the states after $t - u$ iterations, we see that (again) the most likely state is always the one with $t' = u$ and $u' = u$: in the most likely scenario, the decoder has flipped all the normal bits. By looking at the probabilities with which the other states occur, we observe that the decoder will flip also bad bits with high probability. For instance, for $u = 28$, the decoder is in state $(28, 27)$ (i.e., it has flipped one bad bit) with probability 2.89×10^{-2}. We notice that there is also one state which can only be reached if the decoder has flipped one good bit (namely, the state $(30, 28)$), but this is much less likely than the state $(30, 29)$, which can only be reached when the decoder flips one suspicious bit.

We notice that, for $u = 28$, there is only one final state, having $t' = 0$ and $u' = 0$: this is associated with a decoding success. Observe that the associated probability is ≈ 1: this means that this state is the only one that has not been filtered by the Markov model computation (for this simulation, we have filtered all paths with probability less than 2^{-40}). Therefore, we can only say that all the other states are happening with probability less than 2^{-40}.

For $u = 37$, as we expected, the decoder is most likely converging to a near-codeword (with probability ≈ 0.66, the final state is $(71, 71)$). All the other states are (t', t')-almost near-codewords with t' far from $v/2$: the decoder has either taken the path of convergence to a near-codeword (t' is getting closer to v) or that of successful decoding (t' is getting closer to 0).

4 Correcting near-codewords by syndrome recognition

The previous section shows that BF-Max, in the unlikely case of decoding failures, is expected to converge to a near-codeword. We now describe how this behavior can be exploited to improve decoding. Since our modification is general and can be applied to any BF decoder, we describe it using the template of BF from Algorithm 1. We verify its effectiveness through numerical simulations on toy codes.

4.1 Recognizing syndromes of near-codewords

Let us consider the following list:

$$\texttt{NC_Syndromes} = \left\{ \left(\mathbf{m}_i \mathbf{H}^\top \;\middle|\; i = 0, 1, \cdots, 2r-1 \right\} \right.$$

which stores the syndromes of the $2r$ near-codewords. Notice that, because of the polynomial view, we have

$$\mathbf{s}_i = \mathbf{m}_i \mathbf{H}^\top = \begin{cases} x^i \cdot h_1^2 & \text{if } i < r, \\ x^{i-r} \cdot h_2^2 & \text{if } i \geqslant r. \end{cases} \tag{1}$$

Algorithm 2 describes how `NC_Syndromes` is employed to tweak BF decoding. At the end of every iteration, the decoder checks whether the current syndrome is one of the syndromes stored in `NC_Syndromes`. When this is true, the decoder is able to acknowledge that it got stuck in a near-codeword and can consequently recover from this situation by summing the corresponding near-codeword to the current error estimate. Since this implies that the updated syndrome is null, the decoder returns the updated error estimate.

The if condition in line 9 of Algorithm 2 is not mandatory, but helps in avoiding unnecessary searches in the table. Indeed, the syndrome of each near-codeword has weight exactly v: when the syndrome weight differs from this value, we know for sure that the decoder is not stuck in a near-codeword.

With a clever implementation of the table, both the occupied memory and the time to access the table are rather small.

Proposition 4. *The table* `NC_Syndromes` *can be set so that the occupied memory is* $O\big(r \cdot v \cdot \log_2(r)\big)$ *and each access takes time* $O\big(v \cdot \log_2^2(r)\big)$.

Proof. Since each syndrome $\mathbf{s}_i$ has support size v, we can store it using the elements of its support, using $v \cdot \log_2(r)$ bits for each syndrome. The syndromes can then be sorted, so that accessing the table can be done with a binary search algorithm. This, in the worst case, would require to make $\log_2(2r)$ comparisons between objects having binary length $v \cdot \log_2(r)$. □

Remark 7. We observe that we are not considering extra memory for storing the near-codewords, as they correspond essentially to columns of $\mathbf{H}$.

Algorithm 2: BF with knowledge of near-codewords

Data: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$, maximum number of iterations $\texttt{IterMax} \in \mathbb{N}$, table `NC_Syndromes`, near-codewords $\mathbf{m}_0, \cdots, \mathbf{m}_{2r-1} \in \mathbb{F}_2^{2r}$
Input: syndrome $\mathbf{s} \in \mathbb{F}_2^r$
Output: decoding failure $\perp$, or vector $\widehat{\mathbf{e}} \in \mathbb{F}_2^n$ such that $\mathbf{s} = \widehat{\mathbf{e}}\mathbf{H}^\top$

Set $\widehat{\mathbf{e}} = \mathbf{0}$;// Initial error estimate
Set $\texttt{Iter} = 0$;// Number of performed iterations
while $\big(\mathbf{s} \neq \mathbf{0}\big) \wedge \big(\texttt{Iter} < \texttt{IterMax}\big)$ **do**
    /* Compute counters */
    for $i = 0, 1, \cdots, n-1$ **do**: compute σ_i;
    /* Find positions that are likely to be wrongly estimated */
    Determine $J \subseteq \{0, \cdots, n-1\}$ so that, for each $i \in J$, σ_i is large enough;
    /* Update syndrome and error estimate */
    for $i \in J$ **do**
        $\widehat{e}_i \leftarrow \widehat{e}_i + 1$;
        $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{h}_i$;
    /* Compare with syndromes of near-codewords */
    if $\mathrm{wt}(\mathbf{s}) = v$ **then**
        if `NC_Syndromes` contains $\mathbf{s}$ at position i **then**
            Update $\widehat{\mathbf{e}}$ as $\widehat{\mathbf{e}} + \mathbf{m}_i$;
            return $\widehat{\mathbf{e}}$;
    /* Update number of performed iterations */
    $\texttt{Iter} \leftarrow \texttt{Iter} + 1$;
/* Return failure if syndrome is not null */
if $\mathbf{s} \neq \mathbf{0}$ return $\perp$, **else** return $\widehat{\mathbf{e}}$;

Remark 8. We observe that other implementations for `NC_Syndromes`, achieving better performance, may be possible. Still, even the simple solution of using a table with sorted entries is enough to yield a very small complexity. Indeed, remember that the complexity of BF decoding grows at least as $O(r \cdot v)$, which is greater than $O\big(\log_2(r) \cdot v\big)$ when v is small. For instance, for cryptographic QC-MDPC codes, one has that $O\big(\log_2^2(r) \cdot v\big)$ is in $O\big(\sqrt{r} \cdot \log_2^2(r)\big)$, while the complexity of BF grows at least as $O(r^{1.5})$.

4.2 Numerical results on toy parameters

To confirm the effectiveness of the approach we propose, we have run intensive numerical simulations on toy double circulant QC-MDPC codes with $r = 2003$ and weight $v \in \{9, 11, 13, 15\}$. We have considered three BF decoders: BF-Max, MLD and OoP. For the latter, we have set $\texttt{IterMax} = 15$ and have used the

threshold values reported in Table 4. These values have been chosen with the following criterion: the first iterations start with large thresholds (aiming to guarantee that wrong flips occur with small probability), while the subsequent iterations use smaller thresholds (since, in this stage, we expect that the majority of errors have been corrected, so wrong flips occur with small probability in any case). Results of numerical simulations are reported in Figure 6.

Table 4: Thresholds used for the simulations of the OoP decoder, reported in Figure 6c.

v	Thresholds
9	$(9, 8, 8, 8, 8, 7, 7, 7, 6, 6, 6, 6, 5, 5, 5)$
11	$(11, 10, 10, 9, 9, 8, 8, 8, 7, 7, 7, 7, 6, 6, 6)$
13	$(13, 12, 12, 11, 11, 10, 10, 9, 9, 8, 8, 8, 7, 7, 7)$
15	$(15, 14, 14, 13, 13, 12, 12, 11, 11, 10, 10, 9, 9, 8, 8)$

As we can see, for BF-Max, the impact of our modification is rather significant while, for the other two decoders, the reduction in the DFR appears to be less relevant. This is due to the fact that, by design, BF-Max must flip one bit in each iteration: this makes convergence to a near-codeword inevitable. For instance, consider a configuration where counters for suspicious bits are relatively small, but large enough to be among the counters with the greatest values. In such a case, BF decoders using a threshold criterion (such as the MLD and the OoP) may not flip anything. Instead, with large probability, the BF-Max will flip one of these suspicious bits and trigger the avalanche effect leading to the convergence to a near-codeword. When this happens, however, our modification comes into play and the decoder can recover from such a situation.

In any case, even for MLD and OoP, our technique allows to achieve some reduction in the DFR. We also observe that, for all decoders, the impact in the waterfall (i.e., the region where the DFR starts decreasing) is rather mild: this was expected since, for high DFR values, almost near-codewords are not the most meaningful causes of failures [9].

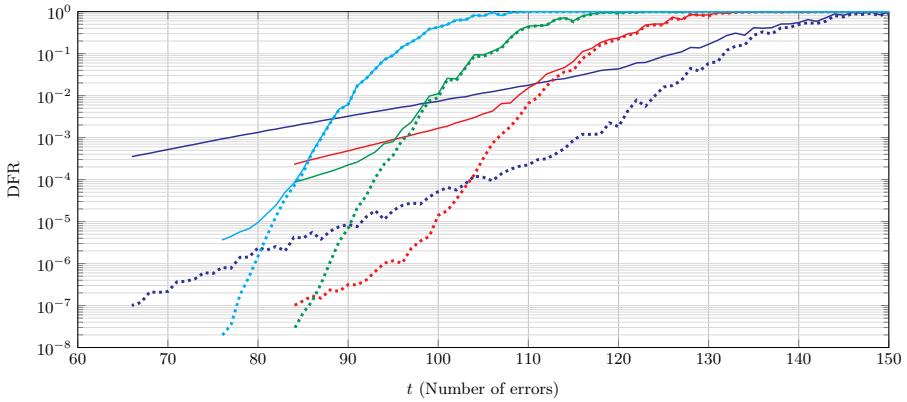
Finally, we point out that we have run simulations considering also other choices for the OoP thresholds and have obtained similar results.

4.3 A time-memory trade-off

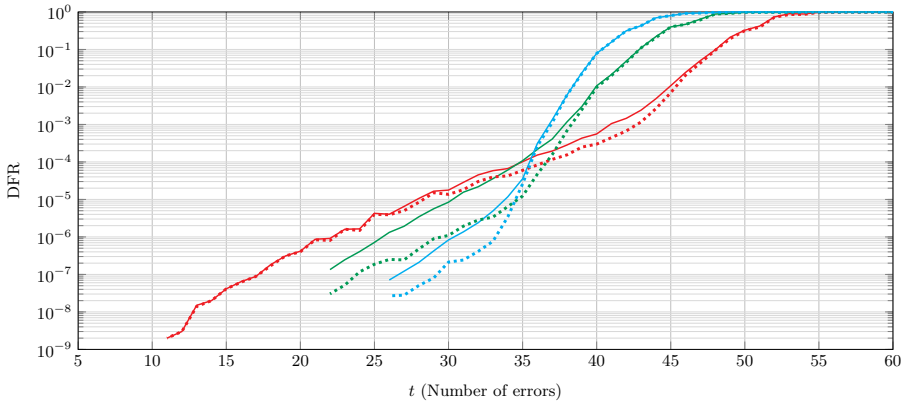
We now show how, by exploiting quasi-cyclicity, one can reduce the space required for storing `NC_Syndromes` , at the cost of a moderate increase in the cost of each table access.

Observe that all syndromes for near-codewords in the look-up table are of the form

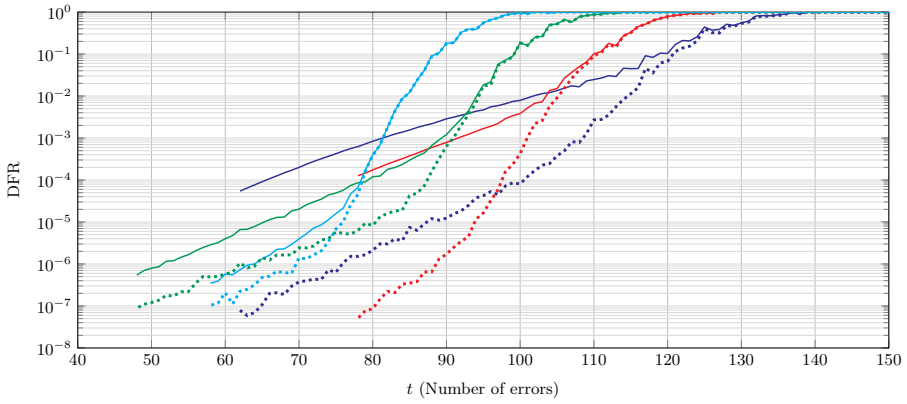
$$\underbrace{\left\{x^i \cdot h_1^2 \;\middle|\; i = 0, \cdots, r-1\right\}}_{\mathcal{Z}_1} \cup \underbrace{\left\{x^i \cdot h_2^2 \;\middle|\; i = 0, \cdots, r-1\right\}}_{\mathcal{Z}_2}.$$



(a) BF-Max, `IterMax` $= 2t$



(b) MLD, `IterMax` $= 50$



(c) OoP, `IterMax` $= 15$, thresholds as in Table 4



Fig. 6: Simulated DFR for double circulant codes with $r = 2003$, for several decoders and values of v. Continuous lines refer to the standard decoder, while dotted lines refer to the improved decoder. Each point has been estimated with at least 30 decoding failures.

Thus, we can just store a representative for each set, for instance, the syndromes $s_0 = h_1^2$ and $s_r = h_2^2$ (we are using the same ordering for near-codewords as (1)). During decoding, whenever the updated syndrome s has weight v, we start comparing cyclic shifts of s with s_0 and s_r. For instance, if $s = s_0$, then we know that s is the syndrome of the near-codeword $\mathbf{m}_0$. If, instead, we find a match for some $i > 0$, then from $x^i \cdot s = s_0$ we get $s = x^{-i} \cdot s_0 = x^{r-i} \cdot s_0$, which implies that s is the syndrome of the near-codeword $\mathbf{m}_{r-i}$.

This approach would reduce down the required memory to $2v \cdot \log_2(r)$, since we just need to store s_0 and s_r (each requiring $v \log_2(r)$ bits), but requires $2r$ comparisons in the worst case, since we need to compare $x^i \cdot s$ with s_0 and s_r, for $i = 0, \cdots, r-1$. Observe that, to compute each $x^i \cdot s$, we need v sums over $\mathbb{Z}_r$[8], each taking time $O\big(\log_2(r)\big)$. So, overall, this naive approach requires $2r$ polynomial comparisons and rv sums over $\mathbb{Z}_r$, in the worst case.

Exploiting sparsity, we can actually do better. Indeed, as representatives for $\mathcal{Z}_1$ and $\mathcal{Z}_2$, we can more conveniently use cyclic shifts of both s_0 and s_r so that, for both, the first coefficient is set. Namely, for $j \in \{0\,;\, r\}$, let

$$s_j = \sum_{\ell \in \mathrm{Supp}(s_j)} x^\ell, \quad \mathrm{Supp}(s_j) = \{\ell_1^{(j)}, \cdots, \ell_v^{(j)}\}.$$

We define $\widetilde{s}_j = x^{-\ell_1^{(j)}} \cdot s_j$, leading to

$$\widetilde{s}_0 = x^{-\ell_1^{(0)}} \cdot h_1^2, \qquad \widetilde{s}_r = x^{-\ell_1^{(r)}} \cdot h_2^2.$$

Now, either we store the two pairs

$$\big(\ell_1^{(0)},\ \widetilde{s}_0\big), \qquad \big(\ell_1^{(r)},\ \widetilde{s}_r\big), \tag{2}$$

or we compute them on the run. In the first case, the required memory is $2v \log_2(r)$ and there is no computational overhead; in the second case, we do not need extra memory but, at each iteration (if the syndrome weight is v) we need to recompute $\widetilde{s}_0$ and $\widetilde{s}_r$, which takes time $O\big(v \cdot \log_2(r)\big)$.

To compare the updated syndrome s with syndromes of near-codewords, we compute $x^{-i} \cdot s$, for each $i \in \mathrm{Supp}(s)$ (since we want the rotated polynomial to start with a set coefficient) and then check it against $\widetilde{s}_0$ and $\widetilde{s}_r$. When there is a match, then we easily get the corresponding near-codeword. For instance, if

$$\widetilde{s}_0 = x^{-i} \cdot s \qquad \Longrightarrow \qquad s = \big(x^{i-\ell_1^{(0)}}\big) \cdot h_1^2.$$

This tells us that the associated near-codeword is the one with index $(i - \ell_1^{(0)} \bmod r)$. Analogously, if $\widetilde{s}_r = x^{-i} \cdot s$, we know that

$$\widetilde{s}_r = x^{-i} \cdot s \qquad \Longrightarrow \qquad s = \big(x^{i-\ell_1^{(r)}}\big) \cdot h_2^2,$$

[8] Since s has weight v (otherwise, there is no need to access the table), we compute $x^i \cdot s$ by summing i to each index in the support of s and then reducing modulo r.

so the associated near-codeword is the one with index $r + \left(i - \ell_1^{(r)} \mod r\right)$.

With this approach, we just need to consider v rotations of s: the amount of sums in $\mathbb{Z}_r$ is $v(v-1)$, so overall the required time grows as $O\left(v^2 \cdot \log_2(r)\right)$.

Remark 9. For cryptographic QC-MDPC codes, each table access takes time $O\left(r \cdot \log_2(r)\right)$.

5 Approximating the decryption failure rate

In this section, we consider a method to obtain a good approximation of the DFR in the floor region or, at least, a lower bound on the DFR of a BF decoder. The methodology is very similar to the one in [16, Proposition 5] and the same as [9, Section 6.3], but applied to the modified decoder we propose in this work. For some of the toy parameters we have already considered in the previous section, we show how our theoretical estimates for the floor coincide with the numerical estimates. This confirms that the modification we propose improves decoding of (t, u)-almost near-codewords with large values of u.

The possibility to get some estimate for the DFR in the floor region becomes rather handy when the actual DFR is far beyond values that can be evaluated experimentally (as it is the case for cryptographic QC-MDPC codes).

5.1 The contribution of (t, u)-almost near-codewords

Recall that, for all values of u, the sets of (t, u)-almost near-codewords form a partition of the Hamming sphere with radius t. In other words, each error pattern with weight t must be a (t, u)-almost near-codeword for a unique value of u. Then, we get[9]

$$\mathrm{DFR} = \sum_{u=1}^{v} \Pr\left[\mathbf{e} \in \mathcal{E}_{t,u} \;\middle|\; \mathbf{e} \xleftarrow{\$} \mathcal{S}_t\right] \cdot \Pr\left[\mathsf{BF}(\mathbf{e}\mathbf{H}^\top) = \perp \;\middle|\; \mathbf{e} \xleftarrow{\$} \mathcal{E}_{t,u}\right], \qquad (3)$$

where $\mathcal{S}_t$ is the Hamming sphere with radius t. Notice that the above formula is simply the sum, over all values of u, of the probability to get a (t, u)-almost near-codeword multiplied by the average decoding failure probability for a (t, u)-almost near-codeword. The formula holds since the sets of almost near-codewords form a disjoint partition of $\mathcal{S}_t$; even though obtaining this formula is quite simple, for the sake of completeness, we prove it in Appendix A.

Let us indicate with $\varepsilon(t, u)$ an estimate for $\Pr\left[\mathsf{BF}(\mathbf{e}\mathbf{H}^\top) = \perp \;\middle|\; \mathbf{e} \xleftarrow{\$} \mathcal{E}_{t,u}\right]$; for instance, $\varepsilon(t, u)$ can be obtained with numerical Monte Carlo simulations, testing decoding only for (t, u)-almost near-codewords. The probability that a random weight-t error vector is a (t, u)-almost near-codeword, which we indicate as $f(u)$, is derived in Appendix B. We remark that the computations we provide are essentially the same as the ones in [9, Appendix A] (but, again, we give them

[9] This is the same formula given in [9, Eq. 6.6].

to make this paper self-contained). Moreover, we remark that our closed-form formula for $f(u)$ holds only under the assumption that columns of $\mathbf{H}$ behave as random and independent vectors with length r and weight v, although this is obviously not true because of quasi-cyclicity. For instance, we know that every vector must be a (t, u)-almost near-codeword for some $u \geqslant 1$, while the closed-form formula for $f(u)$ is such that $f(0) \neq 0$. Still, for the parameters we are interested in, $f(u)$ yields a pretty good approximation of the true probability; see Appendix B for a comparison between empirical and theoretical probabilities.

All in all, we get a good approximation for the DFR, for any value of t, as

$$\mathrm{DFR} \approx \sum_{u=1}^{v} f(u) \cdot \varepsilon(t, u). \tag{4}$$

A numerical confirmation. To confirm the validity of the approach described above, in Figure 7 we compare the simulated DFR curves for some of the codes from Figure 6 with the approximation in (4). As we can see, there is a pretty good match for both codes, for all considered values of t.

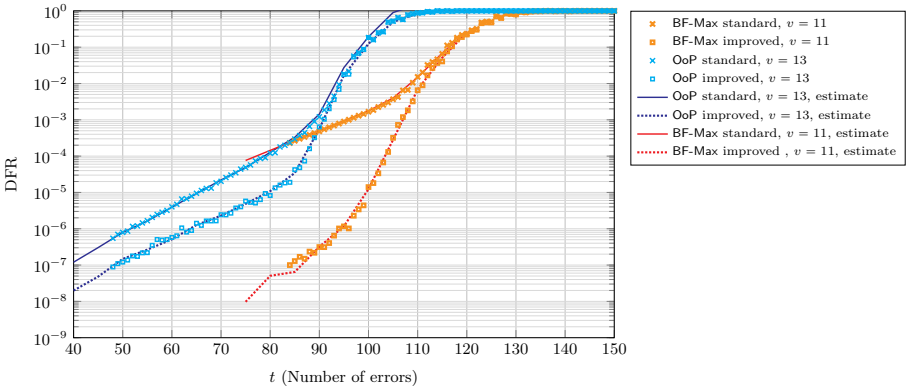


Fig. 7: Comparison between simulated DFR values and the ones resulting from (4), for double circulant QC-MDPC codes with $r = 2003$. For the approximated DFR, we have only considered values of t which are multiples of 5. For each value of t, we have used Monte Carlo simulations to get $\varepsilon(t, u)$, for each value of u.

Remark 10. Let us clarify the meaning of the approximation in (4). To get a reliable approximation, one needs to get very good estimates for $\varepsilon(t, u)$, for relevant values of u. However, this may be rather time consuming, especially for moderately low values of u. Hence, there is a limit on which failure rates can be estimated: this means that, for certain values of u, we will get with overwhelming probability $\varepsilon(t, u) = 0$ even though the failure probability may

be higher. Plugging such values into (4), we actually get a lower bound of the actual DFR. If more simulations are performed, then the bound should become tighter.

5.2 Failure rates for almost near-codewords

To get more insight on how our modification impacts the DFR, let us consider some failure rates for the decoding of almost near-codewords. We focus, again, on the same codes and decoders considered in the previous section; this time, however, we consider only values of t for which we already are in the floor region. Namely, for codes having $v = 11$ and decoded using BF-Max, we have considered $t = 80$, while for codes with $v = 13$ and the OoP decoder, we chose $t = 55$.

The obtained values of $\varepsilon(t, u)$ are shown in Table 5. Notice that when an entry in the table is equal to 0, this does not imply that the associated failure rate is exactly 0, but that we have not been able to see failures in our experiments.

We see that, for both decoders, our tweak leads to a much smaller failure probability for all values of u. In particular, almost near-codewords with large values of u become, actually, easier to correct than those having moderate values of u. This was somewhat expected since, when u is very large, the convergence to the close near-codeword becomes more likely: in such a case, our modification kicks in and allows for a successful correction.

Table 5: Values of $\varepsilon(t, u)$, for codes with circulant side $r = 2003$. Each entry of the table has been estimated by simulating at least 40 decoding failures or up to 10^7 decoding instances.

u	OoP, $v = 13$, $t = 55$		BF-Max, $v = 11$, $t = 80$	
	Standard	Improved	Standard	Improved
2	0	0	0	0
3	0	0	$1 \cdot 10^{-6}$	0
4	$7 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$1.42 \cdot 10^{-4}$	$2 \cdot 10^{-7}$
5	$3.17 \cdot 10^{-4}$	$6.68 \cdot 10^{-5}$	$1.75 \cdot 10^{-2}$	$2.80 \cdot 10^{-6}$
6	$2.56 \cdot 10^{-2}$	$3.90 \cdot 10^{-3}$	$4.36 \cdot 10^{-1}$	$7.17 \cdot 10^{-5}$
7	$3.79 \cdot 10^{-1}$	$1.72 \cdot 10^{-2}$	$9.76 \cdot 10^{-1}$	$1.11 \cdot 10^{-4}$
8	$9.71 \cdot 10^{-1}$	$4.53 \cdot 10^{-3}$	$9.99 \cdot 10^{-1}$	$7.12 \cdot 10^{-5}$
9	$9.99 \cdot 10^{-1}$	$4.20 \cdot 10^{-5}$	1	$9.16 \cdot 10^{-5}$
10	1	$6 \cdot 10^{-7}$	1	$4.89 \cdot 10^{-5}$
11	1	$4 \cdot 10^{-7}$	1	$3.82 \cdot 10^{-5}$
12	1	$5 \cdot 10^{-7}$	-	-
13	1	$2 \cdot 10^{-7}$	-	-

6 Impact on BIKE

Let us now consider the impact of the proposed approach on codes that have exactly the parameters proposed for the BIKE cryptosystem. In this section, we compare the BF-Max decoder with BGF and BIKE-Flip.

We first compare the decoders in terms of time complexity and then look at their DFR.

6.1 Time complexity comparison

Recall that in a standard out-of-place BF decoder, in each iteration, computing counters from scratch takes approximately $n \cdot v \cdot \log_2(v)$ binary operations.

For each flipped bit, the number of elementary operations to update the error estimate and the syndrome is $1 + v$. Assuming that the number of flipped bits is exactly t (i.e., assuming that the decoder never flips a wrong bit), the cost of an out-of-place BF decoder performing $\texttt{IterMax}$ iterations can be estimated as

$$T_{\mathsf{OoP}}(\texttt{IterMax}) = \texttt{IterMax} \cdot n \cdot v \cdot \log_2(v) + t \cdot (1 + v). \tag{5}$$

For what concerns BGF, using (5) with $\texttt{IterMax} = 5$ results in a lower bound, since the decoder does some extra operations, with respect to a basic out-of-place BF decoder. Still, for the sake of simplicity, we do not consider such extra operations. BIKE-Flip has $\texttt{IterMax} = 7$ and, in this case, (5) shall give a rather accurate complexity estimate, since the decoder does not perform extra time consuming operations.

For the time complexity of BF-Max, we rely on the analysis from [18][10]. Since the decoder flips a unique bit in each iteration, there is no need to recompute counters from scratch in each iteration and, instead, one can just update the amount of counters that are affected by the flip. As [18] shows, this requires only $v \cdot w$ operations, with w being the row weight of $\mathbf{H}$. For double-circulant QC-MDPC codes, we have $w = 2v$ hence updating counters in each iteration costs $2v^2 \cdot \log_2(v)$ operations.

Considering all the other costs, we obtain the following overall cost for BF-Max:

$$T_{\mathsf{BF\text{-}Max}}(\texttt{IterMax}) = n \cdot v \cdot \log_2(v) + \texttt{IterMax} \cdot \big(n \cdot \log_2(v) + 1 + v + 2v^2 \cdot \log_2(v)\big). \tag{6}$$

As values for $\texttt{IterMax}$, we will consider $2t$, $1.25 \cdot t$ and $1.15 \cdot t$ (rounded up to the nearest integer). A comparison between the resulting costs, considering BIKE parameters for NIST category 1, is given in Table 6. As we can see, the time complexity of BF-Max is in the same ballpark as BGF and BIKE-Flip and actually, slightly lower for $\texttt{IterMax} = 1.25 \cdot t$ and $\texttt{IterMax} = 1.15 \cdot t$. Obviously, these estimates shall be taken with a grain of salt (we are not counting the cost of memory, etc.). In any case, they show that BF-Max runs in time which, at the very least, is comparable with that of other decoders. So, its performance in terms of DFR is worth looking into.

[10] Notice that there is a little typo in [18, Prop. 3.1]: the time complexity misses a factor $\log_2(v)$.

Decoder	`IterMax`	Complexity
BGF	5	$2^{25.70}$
BIKE-Flip	7	$2^{26.19}$
BF-Max	268	$2^{26.02}$
	168	$2^{25.48}$
	155	$2^{25.39}$

Table 6: Time complexity estimates for the three considered decoders, for BIKE parameters for NIST category 1.

Remark 11. For all decoders, we will evaluate the DFR with and without our proposed enhancement. For simplicity, we neglect the computational overhead due to our modification since, as we have already argued, we expect it to be very mild.

Moreover, we expect that the time complexity of BF-Max can be further reduced by considering a non-naive implementation of the search for the maximum counter, which we are now estimating as $O\big(n \cdot \log_2(v)\big)$. Observe that, for each iteration, this is the most time-consuming operation. We can reduce significantly this cost with an appropriate data structure (e.g., a hash table).

Remark 12. Notice that, for all considered decoders, the average number of required iterations may be less than `IterMax`. However, as shown in [20], [21], both the average number of performed iterations as well as the number of performed flips can be used to mount side-channel attacks. Thus, to obtain a constant time implementation of BF, the decoder should never abort before the maximum number of iterations has been performed. Analogously, the number of syndrome bits which are flipped, in each iteration, can leak information about the secret key. In many implementations where an out-of-place decoder is employed, such as that described in [22], updating the syndrome and avoiding leaking the number of flipped bits requires to access all memory locations where the syndrome is stored. This leads to some additional overhead. For the sake of simplicity, we neglect the above facts and rely on (5) (which, thus, we expect to give a lower bound on the actual complexity). We remark that, for BF-Max, the above types of leakage should not be an issue since, in each iteration, the decoder performs a unique flip in the error estimate and exactly v flips of syndrome bits. However, there may be other sources of information leakage which we are not considering here, and rely on (6) for simplicity.

6.2 DFR comparison

We now consider how these three decoders compare in terms of DFR. First, we consider several values of t and keep all the other parameters unchanged: this allows to see the waterfall behavior. Then, we look at the value $t = 134$, as per the BIKE specification, and focus on decoding of (t, u)-almost near-codewords.

Case $t > 134$. Figure 8 shows the DFR values obtained with numerical simulations, for several values of t. For the BF-Max decoder, we have considered the same values of `IterMax` as in Table 6. Notice that, for the decoders we are considering, the time complexities are in the same ballpark. As we can see, with respect to BGF and BIKE-Flip, the BF-Max decoder has a significantly lower DFR for $\texttt{IterMax} = 1.25 \cdot t$ and $\texttt{IterMax} = 2t$.

We point out that, in all the experiments we made, for the three decoders, our modified decoder has never successfully accessed the look-up table. This means that, in this regime, convergence to a near-codewords never happened and we are still pretty far from the floor (this is also evident by the behavior of the curves). In other words, in this regime, almost near-codewords are not the main reason for decoding failures.

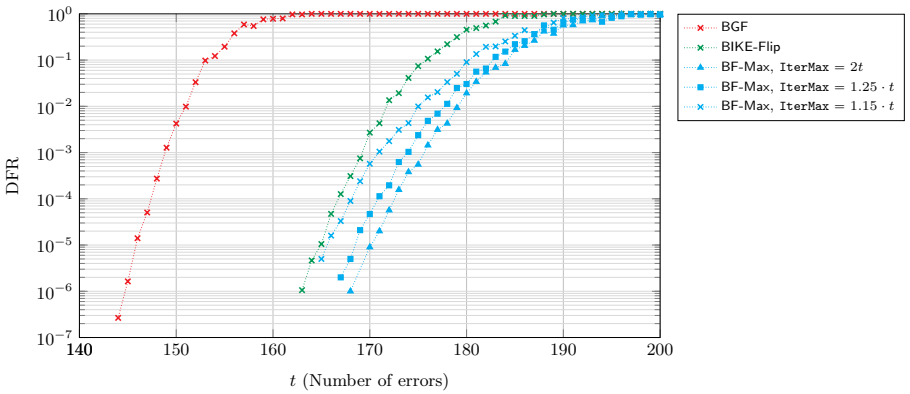


Fig. 8: DFR estimates with Monte Carlo simulations, for NIST category 1 BIKE parameters, as a function of the number of errors t.

Case $t = 134$. For $t = 134$, we have simulated decoding only for (t, u)-almost near-codewords. In this regime, all decoders start hitting the table and, consequently, our enhancement leads to some reduction in the DFR. For this reason, we believe the code is already in the floor regime. For each decoder, we have estimated $\varepsilon(t, u)$ through Monte Carlo simulations. For each value of u, the failure probability has been estimated using up to 10^8 decoding instances for all three decoders (BIKE-Flip, BF-Max and BGF). The range of simulated values for u has been restricted to $[28\ ;\ 45]$, considering that:

– values of u that are too large have not been simulated since, at some point, the decoding failure probability becomes constant and either 1 (for the standard versions) or 0 (when our improvement is employed);

– values of u that are too small have not been simulated as well, since in this case $\varepsilon(t,u)$ is rather small and simulations would have been too time consuming.

The resulting values of $\varepsilon(t,u)$ are shown in Figure 9.

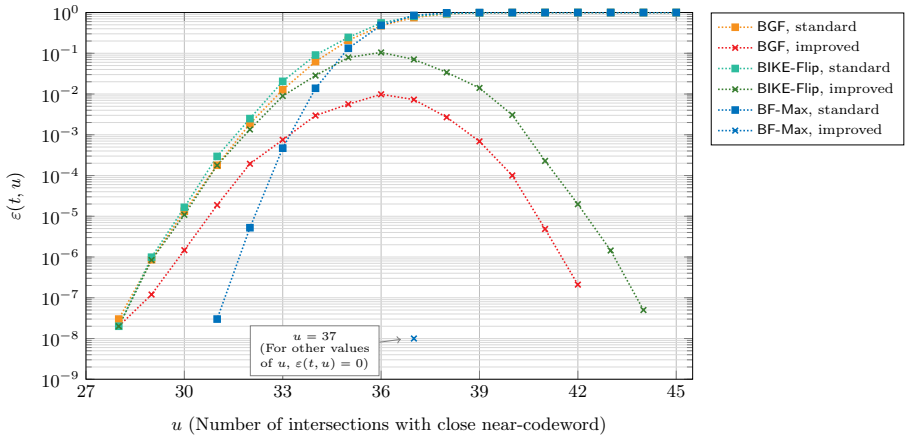


Fig. 9: Values of $\varepsilon(t,u)$, estimated through Monte Carlo simulations, for NIST category 1 BIKE parameters. For BF-Max, we have considered $\texttt{IterMax} = 2t = 268$.

Observe that the figure reports only a single value for the improved BF-Max. This is due to the fact that, for this decoder, we have actually been able to see only one decoding failure, for $u = 37$, out of 10^8 decoding attempts. We have thus considered

$$\varepsilon(t,u) = \begin{cases} 0 & \text{if } u \neq 37, \\ 10^{-8} & \text{if } u = 37. \end{cases}$$

We know that this estimate is obviously rather vague and may change if more intensive numerical simulations are performed[11].

The corresponding DFR estimates, computed using (4), are shown in Table 7. More details about the data from numerical simulations can be found in Appendix D. We remark that the values in the table shall be interpreted as lower bounds on the true DFR, instead of approximations. Still, they can be used to draw some comparison on the behavior of these decoders.

We observe that the DFR of both BGF and BIKE-Flip is only marginally affected by the look-up table tweak. This is due to the fact that, for both decoders,

[11] As we have already said, such simulations are already running. In particular, we are now simulating up to 10^9 for many values of u. At the moment of writing, we have concluded simulations for $u = 38$ and, again, found $\varepsilon(t,38) = 0$. We plan to update the paper as soon as relevant results are obtained.

Table 7: DFR approximations for NIST category 1 BIKE parameters. For BF-Max, we have considered $\texttt{IterMax} = 2t = 268$, using the values of $\varepsilon(t, u)$ from Fig. 9.

Decoder	Basic	Improved
BGF	$2^{-160.08}$	$2^{-160.87}$
BIKE-Flip	$2^{-160.50}$	$2^{-160.56}$
BF-Max	$2^{-181.23}$	$2^{-230.92}$

our enhancement reduces $\varepsilon(t, u)$ significantly only for large values of u. Instead, for smaller values of u, for the basic and improved decoders the values of $\varepsilon(t, u)$ are approximately the same. Since for NIST category 1 BIKE parameters the most likely almost near-codewords are those with low values of u (the expected value for u is ≈ 4.7), for these decoders our modification works only for very unlikely events of errors.

Instead, for BF-Max, the situation seems to be the opposite: even for low values of u, our enhancement brings to a drastic reduction in the DFR. Also in this scenario, the BF-Max decoder appears to have the lowest DFR among all decoders we have tested. Remarkably, the DFR for the basic BF-Max is already lower than that of both BGF and BIKE-Flip.

The BGF and BIKE-Flip perform at most 5 and 7 iterations, respectively, so they explore a few states. Moreover, in each iteration, the decoder potentially flips a large number of bits. So, transitions to one state from the other typically require a significant change in the parameters identifying the state. Our modification works only when the residual errors converge perfectly to the near-codeword. Even if only one bit is missing, then the table will not be read and the decoder will not recognize the proximity with a near-codeword.

For BF-Max, instead, convergence to a near-codeword happens more progressively. Indeed, the decoder explores several states and the transitions from one state to the other are more granular: once during decoding the decoder is close to a near-codeword, it is rather likely that in the next iterations the decoder will converge to it. This particular behavior has already been motivated in Section 3.5.

7 Profile of error vectors and sums of near-codewords

So far, we have only discussed how to tackle errors that, when decoded, converge to a single close near-codeword. However, this is not the only way near-codewords can affect decoding. In this section we make some observations about more general error patterns, which we refer to as *sums of near-codewords*. Looking ahead, we generalize the technique we have introduced in the previous sections and describe some ideas to handle these more general error structures. The purpose of this section is to provide some ideas for showing that these cases can be addressed, even if their deeper investigation is left for future work.

7.1 The profile of an error vector

We start with a simple intuition: if the initial error vector $\mathbf{e}$ has a sufficiently large number of overlapping ones with multiple near-codewords, then it is likely that a BF decoder makes $\widehat{\mathbf{e}} + \mathbf{e}$ converge to the sum of such near-codewords (or to a vector being very close, in terms of Hamming distance, to such a sum). Later on, we corroborate this intuition with numerical simulations; for now, we try to formalize the reasoning as much as possible by introducing the notion of profile of an error vector.

Definition 7 (Profile of error vectors). *Let* $\mathbf{e}$ *with weight* $t > v$*. The profile of* $\mathbf{e}$ *is the tuple* $\mathcal{P}(\mathbf{e}) = (a_0, a_1, \cdots, a_{2r-1}) \in \{0\ ;\ v\}^{2r}$*, where each* a_i *corresponds to the number of overlapping ones between* $\mathbf{e}$ *and the* i*-th near-codeword* $\mathbf{m}_i$*, i.e.,*

$$a_i = \left|\mathrm{Supp}(\mathbf{m}_i) \cap \mathrm{Supp}(\mathbf{e})\right|.$$

We refer to a_i *as the similarity between* $\mathbf{e}$ *and* $\mathbf{m}_i$*; so, the profile of an error vector is the ordered list of all its similarities.*

Observe that if an error vector is a (t, u)-almost near-codeword, then the maximum value in its profile $\mathcal{P}(\mathbf{e})$ is u and if the i-th near-codeword is a close near-codeword, then $a_i = u$. In other words, its maximum similarity with near-codewords is exactly u.

Profiles of error vectors carry relevant information about how BF decoding behaves. The main intuition is very simple: BF decoding of an error vector $\mathbf{e}$ is more prone to converge to the sum of the near-codewords with the largest similarities. Let us better formalize the intuition with which we started this section: if $\mathbf{e}$ has sufficiently large similarity with the z near-codewords $\mathbf{m}_{i_1}, \mathbf{m}_{i_2}, \cdots, \mathbf{m}_{i_z}$, then it is rather likely that, at some point during BF decoding, $\widehat{\mathbf{e}} + \mathbf{e}$ corresponds to $\mathbf{m}_{i_1} + \mathbf{m}_{i_2} + \cdots + \mathbf{m}_{i_z}$ (or, to a vector which has small Hamming distance from $\mathbf{m}_{i_1} + \mathbf{m}_{i_2} + \cdots + \mathbf{m}_{i_z}$).

Quantifying how large should the similarities be to make BF decoding converge to the associated near-codewords looks like a very complicated task and is out of scope for this paper. As a rule of thumb, we can expect that a similarity close to (or larger than) $v/2$ is enough to trigger the convergence to the associated near-codeword, with high probability. In other words, the more a profile is biased towards high values, the higher the probability that BF will fail by converging to the sum of the near-codewords with the largest similarities.

Observe that, when decoding converges to a sum of near-codewords, the profile of $\widehat{\mathbf{e}} + \mathbf{e}$ is rather correlated with $\mathcal{P}(\mathbf{e})$. Indeed, the decoder operations have the effect of increasing similarities that are high enough in $\mathcal{P}(\mathbf{e})$ and decreasing all the other ones. In other words, the similarities in $\mathcal{P}(\mathbf{e})$ which are sufficiently large are expected to become very close to v in $\mathcal{P}(\widehat{\mathbf{e}} + \mathbf{e})$. Simultaneously, low similarities in $\mathcal{P}(\mathbf{e})$ are expected to become close to 0.

7.2 Recognizing syndromes of sums of near-codewords

The above reasoning gives us a way to generalize our technique from Section 4, in order to correct more general hard-to-decode errors. Observe that, in practice,

this means determining which near-codewords have a large similarity in $\mathcal{P}(\widehat{\mathbf{e}}+\mathbf{e})$, knowing the syndrome $(\widehat{\mathbf{e}}+\mathbf{e})\mathbf{H}^\top$.

In principle, one can now set up a look-up table with the syndromes of all sums of z near-codewords, together with the indices of the associated near-codewords. At each BF iteration, the updated syndrome is compared against all the syndromes in the table: if a match occurs, it is enough to update $\widehat{\mathbf{e}}$ by summing the involved near-codewords and return the new error estimate. Notice that the technique introduced in Section 4 is just a special case of this formulation (corresponding to $z = 1$).

Observe that, from a rigorous point of view, the size of the table grows polynomially with r for any constant z. Indeed, the worst case size is given by $L = z \cdot v \cdot \log_2(r) \cdot \binom{2r}{z}$, since:

- there are $\binom{2r}{z}$ sums of z near-codewords;
- for each sum, the syndrome has maximum weight $z \cdot v$;
- for each syndrome, we store only the elements of the support, each taking $\log_2(r)$ bits.

For $z = O(1)$, we have $L = O\big(r^z \cdot v \cdot \log_2(r)\big)$, which is bounded by a polynomial of r. Accessing the table takes time which is bounded by a polynomial of r as well.

So, in principle, our technique scales efficiently for constant z and allows us to handle the sums of any constant number of near-codewords in polynomial time and space. However, in practice, for cryptographic QC-MDPC codes even the case $z = 2$ is already unfeasible. In particular, for BIKE parameters for NIST category 1, we have that the size of the table would be ≈ 73 GB.

However, there may be ways to circumvent such an unrealistic memory requirement. Below we sketch some ideas which, we believe, may be worth looking into also as improvements/optimizations for the basic case of $z = 1$.

Exploiting quasi-cyclicity. With a time-memory trade-off, the number of elements in the table can be reduced down by a factor $\approx r$. For instance, for the case $z = 2$, all sums are quasi-cyclic rotations of the following two types of sums:

- $\mathbf{m}_0 + \mathbf{m}_i$, for $i = 1, \cdots, 2r-1$
- $\mathbf{m}_r + \mathbf{m}_i$, for $i = r+1, \cdots, 2r-1$

Analogously, all syndromes can be obtained by cyclic rotations of the syndromes of the above vectors. Applying ideas similar to those in Section 4.3, this subset of syndromes would be enough to compare the updated syndrome against all syndromes of sums of pairs of near-codewords.

BF decoding over another QC-MDPC code. The syndromes of near-codewords are rotations of h_1^2 or h_2^2. Let $\mathbf{H}'$ be the double circulant matrix which, in polynomial form, is represented by $(h_1^2\ ,\ h_2^2)$ (i.e., the matrix so that the first circulant block has h_1^2 as first column, and the second circulant block has

h_2^2 as first column). Then, whenever $\widehat{\mathbf{e}} + \mathbf{e}$ is the sum of z near-codewords, this means the updated syndrome is the sum of z columns of $\mathbf{H}'$. Suppose that such columns of $\mathbf{H}'$ are those at indexes $\{i_1, \ldots, i_z\}$, that is, $\widehat{\mathbf{e}} + \mathbf{e} = \mathbf{m}_{i_1} + \cdots + \mathbf{m}_{i_z}$. Then, for the updated syndrome, we have the following

$$(\widehat{\mathbf{e}} + \mathbf{e})\mathbf{H}^\top = \mathbf{b}\mathbf{H}'^\top,$$

where $\mathrm{Supp}(\mathbf{b}) = \{i_1, \cdots, i_z\}$. Notice that $\mathbf{H}'$ is, by construction, the parity-check matrix of a double circulant QC-MDPC code with the same parameters as the initial code (same circulant side, same circulant weight). So, it should have a pretty good error correction capability, which implies that the vector $\mathbf{b}$ can be easily obtained by, e.g., calling some BF decoder, on input the updated syndrome and $\mathbf{H}'$. Once $\mathbf{b}$ has been estimated, its support tells us which near-codewords need to be summed.

Notice that, with this observation, one gets rid entirely of the need for the look-up table, but needs to invoke some additional decoding procedure: again, memory is saved at the cost of some computational overhead.

Recognition of noisy syndromes. Finally, we may be able to identify which near-codewords have large similarities with $\widehat{\mathbf{e}} + \mathbf{e}$, even when $\widehat{\mathbf{e}} + \mathbf{e}$ is not exactly the sum of near-codewords. This should allow exploiting the knowledge about near-codewords even when convergence to a sum of near-codewords is not full (e.g., some suspicious bits are not flipped). There may be multiple ways to do this.

For instance, this can be seen as the problem of decoding a *noisy syndrome*. Indeed, assume $\widehat{\mathbf{e}} + \mathbf{e}$ is at Hamming distance d from a sum of z near-codewords: then, $\mathbf{s}' = (\widehat{\mathbf{e}} + \mathbf{e})\mathbf{H}^\top$ can be written as the sum between z columns of $\mathbf{H}'$ and an error vector of length r and weight d. When d is small enough, a BF decoder (eventually tweaked to take into account the noisiness of the syndrome) should do the job.

7.3 A numerical confirmation

We conclude this section by backing up our intuition with some numerical results. We have considered a code with $r = 2003$ and $v = 15$, decoded with BF-Max with `IterMax` $= 2t$ and using our technique from Section 4. We have set $t = 80$ which, according to Fig. 6, implies we are in the floor regime. More details about the code can be found in Appendix C.

In other words, this code should operate in a regime where (t, u)-almost near-codewords with large values of u are corrected by our enhancement. So, residual events of failure should be given by anything that goes beyond the convergence to a single near-codeword: we expect that the most relevant events of failure are associated with the convergence to a sum of two near-codewords.

To verify this, we have run intensive numerical simulations and have collected the errors leading to decoding failures, being able to collect 44 errors. For each of them, we have computed the profile of the updated error vector $\widehat{\mathbf{e}} + \mathbf{e}$ and

considered the two near-codewords with the largest similarities. In 41 out of 44 cases, $\widehat{\mathbf{e}} + \mathbf{e}$ was indeed the sum of these near-codewords: using the ideas we have described above, we would have been able to correctly decode also these patterns. For such errors, the profiles of $\mathbf{e}$ and $\widehat{\mathbf{e}} + \mathbf{e}$ were strongly correlated: the two near-codewords with the highest similarities in both profiles were always the same.

Interestingly, the three errors that were not apparently associated with a sum of two near-codewords were (t, u)-almost near-codewords with rather small values of u (5 and 4). Moreover, for such errors, the profiles of $\mathcal{P}(\mathbf{e})$ and $\mathcal{P}(\widehat{\mathbf{e}}+\mathbf{e})$ were not correlated significantly: for instance, the largest similarities in $\mathcal{P}(\mathbf{e})$ had not been preserved in $\mathcal{P}(\widehat{\mathbf{e}} + \mathbf{e})$.

We believe these errors are due to peculiar error patterns that are not associated with near-codewords and lead to a very bad counters configuration. Understanding the properties of such patterns (e.g., whether they are expected to exist also for cryptographic QC-MDPC codes, their structure, etc.) is a valuable research direction that we leave for future works.

In Appendix C, we give some more comments about this experiment; full details and data about this experiment can be found in the public repository that contains all the source code associated with this paper.

8 Conclusions

We have proposed a technique to exploit the convergence to near-codewords in order to reduce the DFR of BF decoders working on double circulant QC-MDPC codes. This technique can be applied to any BF decoder and comes with a very mild computational overhead. We have verified that, for several decoders, our approach allows achieving significant reductions in the DFR. In particular, for BIKE parameters, this becomes especially true when considering BF-Max. Despite a running time which is comparable with that of decoders already employed by BIKE within the NIST competition, the basic version of BF-Max already outperforms the other decoders (even when improved with the techniques we describe in this paper). Using our enhancement, the DFR of BF-Max is reduced further.

We have analyzed the behavior of BF-Max from an intuitive standpoint, using simple and crude approximations for counters, as well as through a more rigorous theoretical approach relying on an adaptation of the modeling from [9]. Both approaches show that, in the event of failure, BF-Max is prone to converge to a near-codeword, that is, a configuration that can be corrected through our approach. In particular, for BF-Max, this behavior turns out to be more pronounced with respect to what happens for other decoders.

Finally, we have initiated the study of more general events of failure by putting forward the notions of profiles and similarities for error vectors. This allowed us identifying more general events of failures: instead of converging to a single near-codeword, the decoder converges to a sum of near-codewords. Preliminary results show that, for the enhanced BF-Max, such error vectors are rather

meaningful at very low DFR values. Understanding how such types of errors can be decoded can reduce further the DFR of a BF decoder, but we leave this investigation for future work.

Acknowledgements

This work was partially supported by the Italian Ministry of University and Research under the Italian Fund for Applied Science (FISA 2022), Call for tender No. 1405 published on 13-09-2022 - project title "Quantum-safe cryptographic tools for the protection of national data and information technology assets" (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001), Grant Assignment Decree no. 15461 adopted on 02.08.2024. The work of Alessio Baldelli was partially supported by Agenzia per la Cybersicurezza Nazionale (ACN) under the programme for promotion of XL cycle PhD research in cybersecurity (CUP I32B24001750005).

References

[1] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the Fujisaki-Okamoto transformation," in *Theory of Cryptography Conference*, Springer, 2017, pp. 341–371.

[2] P. Santini, M. Battaglioni, M. Baldi, and F. Chiaraluce, "Analysis of the error correction capability of LDPC and MDPC codes under parallel bit-flipping decoding and application to cryptography," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4648–4660, 2020.

[3] V. Vasseur, "Post-quantum cryptography: A study of the decoding of QC-MDPC codes," Ph.D. dissertation, Université Paris Cité, 2021.

[4] N. Sendrier and V. Vasseur, "On the decoding failure rate of QC-MDPC bit-flipping decoders," in *Proc. Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019*, Springer, 2019, pp. 404–416.

[5] N. Sendrier and V. Vasseur, "About low DFR for QC-MDPC decoding," in *Proc. International Conference on Post-Quantum Cryptography*, Springer, 2020, pp. 20–34.

[6] J.-P. Tillich, "The decoding failure probability of MDPC codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 941–945.

[7] G. L. Matthews and E. McMillon, "A combinatorial approach to avoiding weak keys in the BIKE cryptosystem," *Designs, Codes and Cryptography*, pp. 1–24, 2025.

[8] S. Arpin, T. R. Billingsley, D. R. Hast, J. B. Lau, R. Perlner, and A. Robinson, "A study of error floor behavior in QC-MDPC codes," in *Proc. International Conference on Post-Quantum Cryptography*, Springer, 2022, pp. 89–103.

[9] S. Arpin, J. B. Lau, A. Mesnard, *et al.*, "Error floor prediction with Markov models for QC-MDPC codes," in *Advances in Cryptology – CRYPTO*, 2025, pp. 221–252.

[10] A. Annechini, A. Barenghi, and G. Pelosi, "Bit-flipping decoder failure rate estimation for (v,w)-regular codes," in *Proc. 2024 IEEE International Symposium on Information Theory (ISIT)*, 2024, pp. 3374–3379.

[11] N. Drucker, S. Gueron, and D. Kostic, "QC-MDPC decoders with several shades of gray," in *Proc. International Conference on Post-Quantum Cryptography*, Springer, 2020, pp. 35–50.
[12] G. Alagic, M. Bros, P. Ciadoux, *et al.*, *Status report on the fourth round of the NIST post-quantum cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology, 2025.
[13] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.
[14] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt: QC-LDPC code-based cryptosystems with bounded decryption failure rate," in *Code-Based Cryptography*, M. Baldi, E. Persichetti, and P. Santini, Eds., Springer International Publishing, 2019, pp. 11–43.
[15] N. Aragon, P. Barreto, S. Bettaieb, *et al.*, "BIKE," *NIST PQC Submission*, 2020.
[16] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "Performance bounds for QC-MDPC codes decoders," in *Code-Based Cryptography Workshop*, Springer, 2021, pp. 95–122.
[17] P. Santini, M. Battaglioni, M. Baldi, and F. Chiaraluce, "Hard-decision iterative decoding of LDPC codes with bounded error rate," in *Proc. IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
[18] A. Baldelli, M. Baldi, F. Chiaraluce, and P. Santini, "BF-Max: An efficient bit flipping decoder with predictable decoding failure rate," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2025, pp. 1–6.
[19] A. Baldelli, M. Baldi, D. De Zuane, and P. Santini, "Near-codewords aware bit flipping decoding of QC-MDPC codes," in *Proc. 2026 IEEE International Symposium on Information Theory (ISIT)*, 2026.
[20] P. Santini, M. Battaglioni, F. Chiaraluce, and M. Baldi, "Analysis of reaction and timing attacks against cryptosystems based on sparse parity-check codes," in *Code-Based Cryptography, 2019*, Springer, 2019, pp. 115–136.
[21] E. Eaton, M. Lequesne, A. Parent, and N. Sendrier, "QC-MDPC: A timing attack and a CCA2 KEM," in *International Conference on Post-Quantum Cryptography*, Springer, 2018, pp. 47–76.
[22] T. Chou, "QcBits: Constant-time small-key code-based cryptography," in *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, 2016, pp. 280–300.

A DFR and partitions of the Hamming sphere

For a given parity-check matrix $\mathbf{H}$ and a given decoder Dec, the DFR corresponds to

$$\mathrm{DFR} = \sum_{\mathbf{e}\in\mathcal{S}_t} \Pr\left[\mathbf{e}\right] \cdot \Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp\right], \tag{7}$$

where $\Pr\left[\mathbf{e}\right]$ indicates the probability that the error vector $\mathbf{e}$ is sampled during encapsulation, while $\Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp\right]$ is the probability that its syndrome cannot be decoded. Observe that, for a deterministic decoder, this can only be either 0 or 1. Since we are interested in the more general case of a probabilistic decoder, such a probability can instead take any value in the range $[0\ ;\ 1]$. Indeed, the $\mathsf{BF\text{-}Max}$ decoder is probabilistic since, when the maximum counter

value corresponds to two or more bits, the bit to be flipped is selected uniformly at random among them. The other decoders considered in this paper are, instead, deterministic.

For the vector $\mathbf{e}$ we consider the uniform distribution over $\mathcal{S}_t$, hence

$$\Pr\left[\mathbf{e}\right] = \frac{1}{|\mathcal{S}_t|} = \frac{1}{\binom{n}{t}}.$$

So, we rewrite (7) as

$$\text{DFR} = \frac{1}{|\mathcal{S}_t|} \cdot \sum_{\mathbf{e}\in\mathcal{S}_t} \Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp\right]. \tag{8}$$

Let us now consider a partition of $\mathcal{S}_t$ given by the subsets $\mathcal{A}_1, \cdots, \mathcal{A}_z \subseteq \mathcal{S}_t$; remember that, by definition of partition, the sets $\mathcal{A}_1, \cdots, \mathcal{A}_z$ are disjoint and their union corresponds to $\mathcal{S}_t$. Then, we manipulate (8) as follows:

$$\begin{aligned}\text{DFR} &= \frac{1}{|\mathcal{S}_t|} \cdot \sum_{i=1}^{z} \sum_{\mathbf{e}\in\mathcal{A}_i} \Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp\right] \\ &= \sum_{i=1}^{z} \underbrace{\frac{|\mathcal{A}_i|}{|\mathcal{S}_t|}}_{\Pr\left[\mathbf{e}\in\mathcal{A}_i \;\middle|\; \mathbf{e}\xleftarrow{\$}\mathcal{S}_t\right]} \cdot \sum_{\mathbf{e}\in\mathcal{A}_i} \frac{1}{|\mathcal{A}_i|} \cdot \Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp\right].\end{aligned}$$

We observe that $1/|\mathcal{A}_i|$ corresponds to the probability that a vector $\mathbf{e}$ is sampled uniformly at random from $\mathcal{A}_i$, so $\sum_{\mathbf{e}\in\mathcal{A}_i} \frac{1}{|\mathcal{A}_i|} \cdot \Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp\right]$ is the probability that an error sampled uniformly at random from $\mathcal{A}_i$ leads to a decoding failure. Then, we rewrite the above expression as

$$\text{DFR} = \sum_{i=1}^{z} \Pr\left[\mathbf{e}\in\mathcal{A}_i \;\middle|\; \mathbf{e}\xleftarrow{\$}\mathcal{S}_t\right] \cdot \Pr\left[\mathsf{Dec}(\mathbf{e}\mathbf{H}^\top) = \perp \;\middle|\; \mathbf{e}\xleftarrow{\$}\mathcal{A}_i\right].$$

In Section 5, we partition $\mathcal{S}_t$ using the sets of (t,u)-almost near-codewords.

B Probability to sample a (t,u)-almost near-codeword

We start with a result that, modulo some differences in the notation, is identical to the one from [9].

Proposition 5. *Let $\mathbf{e} \xleftarrow{\$} \mathcal{S}_t$ and let $\mathbf{a}_1, \cdots, \mathbf{a}_{2r} \in \mathbb{F}_2^{2r}$ be vectors with length $2r$ and such that:*

- *for $i = 1, \cdots, r$, each vector $\mathbf{a}_i$ is of the form $(\mathbf{a}'_i\ ,\ \mathbf{0})$, with $\mathbf{a}'_i$ sampled uniformly at random with length r and weight v;*
- *for $i = r+1, \cdots, 2r$, each vector $\mathbf{a}_i$ is of the form $(\mathbf{0}\ ,\ \mathbf{a}'_i)$, with $\mathbf{a}'_i$ sampled uniformly at random with length r and weight v.*

Then,

$$\Pr\left[\max_{i\in\{1,\cdots,2r\}}\{|\mathrm{Supp}(\mathbf{e})\cap\mathrm{Supp}(\mathbf{a}_i)|\}=u\right]=f(u),$$

where

$$f(u)=\sum_{t_1=0}^{t}p(t_1)\cdot\left(f_2(t-t_1,u)\sum_{u_1=0}^{u-1}f_1(t_1,u_1)\right.$$
$$\left.+f_1(t_1,u)f_2(t-t_1,u)+f_1(t_1,u)\sum_{u_2=0}^{u-1}f_2(t-t_1,u_2)\right),$$

where, for $j=1,2$*,*

$$f_j(t_j,u_j)=\left(\sum_{i=0}^{u_j}\frac{\binom{t_j}{i}\binom{r-t_j}{v-i}}{\binom{r}{v}}\right)^r-\left(\sum_{i=0}^{u_j-1}\frac{\binom{t_j}{i}\binom{r-t_j}{v-i}}{\binom{r}{v}}\right)^r.$$

Proof. We first consider the probability that $\mathbf{e}$ has weight t_1 in the first half and weight $t_2=t-t_1$ in the second half. This is given by $p(t_1)=\binom{r}{t_1}\binom{r}{t-t_1}/\binom{2r}{t}$. Let $f_1(t_1,u_1)$ be the probability that the maximum intersection between $\mathbf{e}$ and any of the vectors $\mathbf{a}_1,\cdots,\mathbf{a}_r$ is u_1, when the first half of $\mathbf{e}$ has weigh t_1. We first observe that, for any of such vectors, the probability to have exactly i overlapping positions with $\mathbf{e}$ is $\binom{t_1}{i}\binom{r-t_1}{v-i}/\binom{r}{v}$. Then,

$$f_1(t_1,u_1)=\left(\sum_{i=0}^{u_1}\frac{\binom{t_1}{i}\binom{r-t_1}{v-i}}{\binom{r}{v}}\right)^r-\left(\sum_{i=0}^{u_1-1}\frac{\binom{t_1}{i}\binom{r-t_1}{v-i}}{\binom{r}{v}}\right)^r.$$

With analogous meaning and notation, let[12]

$$f_2(t_2,u_2)=\left(\sum_{i=0}^{u_2}\frac{\binom{t_2}{i}\binom{r-t_2}{v-i}}{\binom{r}{v}}\right)^r-\left(\sum_{i=0}^{u_2-1}\frac{\binom{t_2}{i}\binom{r-t_2}{v-i}}{\binom{r}{v}}\right)^r.$$

Now, for a vector $\mathbf{e}$ with weight t_1 in the first half, the maximum intersection with all the vectors $a_1,\cdots,a_{2r}$ is u only if one of the following three events happen:

- $u_2=u$ and $u_1<u$, which happens with probability

$$f_2(t-t_1,u)\sum_{u_1=1}^{u-1}f_1(t_1,u_1);$$

- $u_1=u_2=u$, which happens with probability

$$f_2(t-t_1,u)f_1(t_1,u);$$

[12] Observe that the expression of f_2 is formally the same as f_1; since we need to call it with different inputs, in order to avoid confusion, we wrote two different full expressions anyway.

- $u_1 = u$ and $u_2 < u$, which happens with probability

$$f_1(t_1, u) \sum_{u_2=1}^{u-1} f_2(t - t_1, u_2).$$

Summing over all the probabilities for the events, we get the probability in the thesis. □

If we model the columns of a parity-check matrix as the vectors $\mathbf{a}_1, \cdots, \mathbf{a}_{2r}$ from the previous proposition, then we have that $f(u)$ yields an estimate for the probability that a random vector with weight t is a (t, u)-almost near-codeword. In Figure 10 we compare the resulting theoretical distribution of the value of u with the one estimated through numerical simulations; as we can see, the two distributions match pretty well.

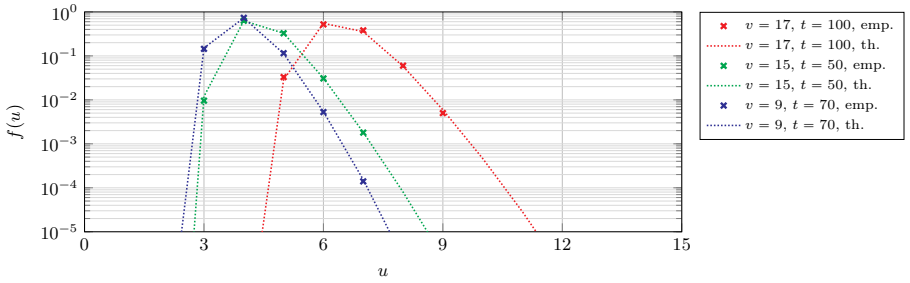


Fig. 10: Comparison between theory and experiments, for the probability that a random vector with weight t is a (t, u)-almost near-codeword. For all experiments, we have used a QC-MDPC code with $r = 521$.

C Details about experiments for sums of near-codewords

The code we have considered has the following parameters $r = 2003$, $v = 15$. We have considered $t = 80$ and BF-Max with `IterMax` $= 2t = 160$.

The support of h_1 is {16, 393, 757, 813, 863, 932, 1004, 1012, 1177, 1216, 1328, 1399, 1560, 1572, 1634}.

The support of h_2 is {147, 312, 783, 1077, 1127, 1135, 1152, 1177, 1406, 1483, 1724, 1839, 1842, 1877, 1935}.

One of the errors leading to a failure has support {107, 119, 172, 179, 181, 227, 337, 453, 544, 566, 675, 690, 700, 728, 734, 763, 839, 891, 910, 912, 920, 965, 1019, 1081, 1082, 1095, 1175, 1307, 1310, 1353, 1379, 1385, 1482, 1530, 1535, 1554, 1567, 1656, 1723, 1766, 1778, 1796, 1797, 1800, 1835, 1862, 1878, 1930, 1949, 2073, 2090, 2108, 2160, 2165, 2209, 2224, 2260, 2365, 2403, 2411, 2466,

2589, 2668, 2721, 2810, 2978, 3058, 3062, 3084, 3093, 3207, 3392, 3442, 3479, 3533, 3596, 3737, 3857, 3860, 3930}.

Computing its profile and sorting positions according to similarity scores, we obtain:

- position: 550, score: 10
- position: 163, score: 7
- position: 1699, score: 4
- position: 2750, score: 3
- position: 619, score: 3

So, $\mathbf{e}$ is a (t, u)-almost near-codeword with $u = 10$. After decoding, we have that $\widehat{\mathbf{e}} + \mathbf{e}$ has the following support: {107, 119, 179, 181, 556, 566, 920, 943, 976, 1026, 1095, 1167, 1175, 1307, 1340, 1363, 1379, 1413, 1482, 1491, 1554, 1723, 1727, 1735, 1766, 1797, 1878, 1949}.

Computing its profile and sorting positions according to similarity scores, we obtain:

- position: 550, score: 14
- position: 163, score: 14
- position: 91, score: 3
- position: 723, score: 3
- position: 622, score: 3

Another error leading to a failure had support {1, 43, 91, 140, 142, 151, 214, 278, 333, 387, 426, 441, 514, 538, 597, 655, 657, 666, 713, 770, 782, 842, 844, 1046, 1061, 1143, 1180, 1234, 1329, 1345, 1360, 1361, 1448, 1494, 1533, 1549, 1590, 1603, 1614, 1644, 1645, 1716, 1801, 1889, 1895, 1941, 1948, 2159, 2178, 2209, 2410, 2413, 2470, 2534, 2610, 2655, 2670, 2680, 2718, 2744, 2754, 2766, 2815, 2827, 2869, 3183, 3277, 3521, 3522, 3523, 3541, 3592, 3596, 3638, 3668, 3679, 3695, 3729, 3864, 3964}.

The positions with the largest similarity scores were as follows:

- position: 317, score: 8
- position: 1213, score: 8
- position: 2602, score: 3
- position: 1386, score: 3
- position: 712, score: 3

After decoding, we have that $\widehat{\mathbf{e}} + \mathbf{e}$ has the following support: {23, 73, 142, 214, 222, 333, 387, 426, 538, 609, 710, 770, 782, 844, 1074, 1130, 1180, 1229, 1249, 1321, 1329, 1494, 1533, 1606, 1645, 1716, 1877, 1889, 1951, 1970}.

Computing its profile and sorting positions according to similarity scores, we obtain:

- position: 317, score: 14
- position: 1213, score: 14
- position: 2602, score: 2
- position: 1386, score: 2
- position: 712, score: 2

D Details about experiments for the floor

In Tables 8, 9 and 10, we report the data about the simulations for decoding of (t, u)-almost near-codewords. Remember that raw data can be found at `https://github.com/secomms/BF_near_codewords`. The values of $f(u)$ have been computed theoretically, while the values of $\varepsilon(t, u)$ have been estimated with Monte Carlo simulations, considering up to 10^8 decoding instances for all three decoders.

		$\varepsilon(t,u)$		$f(u)\cdot\varepsilon(t,u)$	
u	$f(u)$	Std.	Impr.	Std.	Impr.
45	$2^{-272.17}$	1	0	$2^{-272.17}$	0
44	$2^{-263.35}$	1	0	$2^{-263.35}$	0
43	$2^{-254.62}$	1	0	$2^{-254.62}$	0
42	$2^{-246.0}$	1	$2^{-22.18}$	$2^{-246.0}$	$2^{-268.18}$
41	$2^{-237.47}$	1	$2^{-17.65}$	$2^{-237.47}$	$2^{-255.12}$
40	$2^{-229.05}$	1	$2^{-13.29}$	$2^{-229.05}$	$2^{-242.33}$
39	$2^{-220.71}$	$2^{-0.02}$	$2^{-10.5}$	$2^{-220.73}$	$2^{-231.22}$
38	$2^{-212.48}$	$2^{-0.11}$	$2^{-8.54}$	$2^{-212.59}$	$2^{-221.02}$
37	$2^{-204.34}$	$2^{-0.41}$	$2^{-7.1}$	$2^{-204.75}$	$2^{-211.44}$
36	$2^{-196.3}$	$2^{-1.07}$	$2^{-6.67}$	$2^{-197.37}$	$2^{-202.96}$
35	$2^{-188.35}$	$2^{-2.27}$	$2^{-7.47}$	$2^{-190.61}$	$2^{-195.82}$
34	$2^{-180.49}$	$2^{-4.0}$	$2^{-8.4}$	$2^{-184.49}$	$2^{-188.89}$
33	$2^{-172.73}$	$2^{-6.3}$	$2^{-10.38}$	$2^{-179.03}$	$2^{-183.11}$
32	$2^{-165.06}$	$2^{-9.06}$	$2^{-12.33}$	$2^{-174.13}$	$2^{-177.4}$
31	$2^{-157.49}$	$2^{-12.45}$	$2^{-15.7}$	$2^{-169.94}$	$2^{-173.19}$
30	$2^{-150.02}$	$2^{-16.2}$	$2^{-19.38}$	$2^{-166.22}$	$2^{-169.39}$
29	$2^{-142.64}$	$2^{-20.15}$	$2^{-22.99}$	$2^{-162.79}$	$2^{-165.63}$
28	$2^{-135.36}$	$2^{-24.99}$	$2^{-25.58}$	$2^{-160.35}$	$2^{-160.93}$

Table 8: Data for the BGF decoder.

		$\varepsilon(t,u)$		$f(u)\cdot\varepsilon(t,u)$	
u	$f(u)$	Std.	Impr.	Std.	Impr.
45	$2^{-272.17}$	1	0	$2^{-272.17}$	0
44	$2^{-263.35}$	1	$2^{-24.25}$	$2^{-263.35}$	$2^{-287.6}$
43	$2^{-254.62}$	1	$2^{-19.4}$	$2^{-254.62}$	$2^{-274.03}$
42	$2^{-246.0}$	1	$2^{-15.63}$	$2^{-246.0}$	$2^{-261.63}$
41	$2^{-237.47}$	1	$2^{-12.1}$	$2^{-237.47}$	$2^{-249.57}$
40	$2^{-229.05}$	$2^{-0.01}$	$2^{-8.35}$	$2^{-229.05}$	$2^{-237.4}$
39	$2^{-220.71}$	$2^{-0.02}$	$2^{-6.15}$	$2^{-220.74}$	$2^{-226.86}$
38	$2^{-212.48}$	$2^{-0.09}$	$2^{-4.88}$	$2^{-212.57}$	$2^{-217.36}$
37	$2^{-204.34}$	$2^{-0.31}$	$2^{-3.82}$	$2^{-204.65}$	$2^{-208.16}$
36	$2^{-196.3}$	$2^{-0.85}$	$2^{-3.25}$	$2^{-197.14}$	$2^{-199.55}$
35	$2^{-188.35}$	$2^{-2.03}$	$2^{-3.66}$	$2^{-190.37}$	$2^{-192.01}$
34	$2^{-180.49}$	$2^{-3.47}$	$2^{-5.14}$	$2^{-183.96}$	$2^{-185.63}$
33	$2^{-172.73}$	$2^{-5.61}$	$2^{-6.8}$	$2^{-178.34}$	$2^{-179.53}$
32	$2^{-165.06}$	$2^{-8.65}$	$2^{-9.54}$	$2^{-173.71}$	$2^{-174.6}$
31	$2^{-157.49}$	$2^{-11.73}$	$2^{-12.45}$	$2^{-169.22}$	$2^{-169.95}$
30	$2^{-150.02}$	$2^{-15.88}$	$2^{-16.49}$	$2^{-165.9}$	$2^{-166.51}$
29	$2^{-142.64}$	$2^{-19.95}$	$2^{-20.18}$	$2^{-162.59}$	$2^{-162.82}$
28	$2^{-135.36}$	$2^{-25.58}$	$2^{-25.58}$	$2^{-160.93}$	$2^{-160.93}$

Table 9: Data for the BIKE-Flip decoder.

		$\varepsilon(t,u)$		$f(u)\cdot\varepsilon(t,u)$	
u	$f(u)$	Std.	Impr.	Std.	Impr.
45	$2^{-272.17}$	1	0	$2^{-272.17}$	0
44	$2^{-263.35}$	1	0	$2^{-263.35}$	0
43	$2^{-254.62}$	1	0	$2^{-254.62}$	0
42	$2^{-246.0}$	1	0	$2^{-246.0}$	0
41	$2^{-237.47}$	1	0	$2^{-237.47}$	0
40	$2^{-229.05}$	1	0	$2^{-229.05}$	0
39	$2^{-220.71}$	1	0	$2^{-220.72}$	0
38	$2^{-212.48}$	$2^{-0.02}$	0	$2^{-212.5}$	0
37	$2^{-204.34}$	$2^{-0.23}$	$2^{-26.58}$	$2^{-204.57}$	$2^{-230.92}$
36	$2^{-196.3}$	$2^{-1.04}$	0	$2^{-197.34}$	0
35	$2^{-188.35}$	$2^{-2.91}$	0	$2^{-191.26}$	0
34	$2^{-180.49}$	$2^{-6.18}$	0	$2^{-186.67}$	0
33	$2^{-172.73}$	$2^{-11.06}$	0	$2^{-183.79}$	0
32	$2^{-165.06}$	$2^{-17.54}$	0	$2^{-182.61}$	0
31	$2^{-157.49}$	$2^{-24.99}$	0	$2^{-182.48}$	0
30	$2^{-150.02}$	0	0	0	0
29	$2^{-142.64}$	0	0	0	0
28	$2^{-135.36}$	0	0	0	0

Table 10: Data for the BF-Max decoder.