Pruning of RPA Decoders for Reed-Muller Codes Based on Projection Effectiveness and Uniqueness

Renate Debets
r.h.debets@student.tue.nl, ID 1014638
Supervisors: Alexios Balatsoukas-Stimming and Marzieh Hashemipour-Nazari

***Abstract*—This paper concerns the recently developed recursive projection-aggregation decoder for short-blocklength Reed-Muller codes and techniques to reduce its high complexity while retaining an acceptable error-correcting performance. A common way to simplify an RPA decoder sufficiently for hardware implementation is to get rid of projection branches at one or more of its recursion levels. This paper investigates how a good performance can be achieved through the method of choosing branches to retain. In particular, decoders for third-order Reed-Muller codes and the contributions of their usually large number of first-order projections are analysed. Analysis and simulation results suggest that individually, all of these projections perform equally. However, contributions of certain pairs that are indirectly constructed very similarly, overlap highly. This suggests that to take full advantage of the decreased number of first order projections, pruning should take into account the uniqueness of remaining branches. We show that a third-order decoder constructed to contain solely unique first-order projections can perform better than decoders constructed using a equally-sized but different selection of first-order projections.**

I. Introduction

WHILE the fifth generation network for mobile telecommunications (5G) is being rolled out globally, visions for the sixth generation (6G) are already taking shape. By the time 6G will be implemented in practice, it will host connections between a wide range of devices which place a variety of demands on the network, as described in [1]. For a large number of these mobile devices, it will be essential to have an up-, down-, or even sidelink with a considerably low latency and high reliability. Examples of such applications can be found in autonomous driving, medical monitoring, or other situations where human lives might depend on the speed and robustness of a communication protocol. The protocols used in these cases are ultra reliable low latency communications (URLLC) protocols, and will require the usage of a strong error correction code (ECC) to ensure high reliability. The current standard for error correction in 5G is the Low Density Parity Check scheme (LDPC), as described in [2]. This code excels in the large blocklength regime, which is convenient for usage in 5G. LDPC codes become less effective for shorter blocklengths, which is shown in [3] to be a common property of all ECC's. However, due to the requirement of low latency for URLLC, transmitted blocks need to be short and an ECC needs to be used which performs well in this short blocklength regime.

A promising candidate for an ECC to be used in URLLC, is the Reed-Muller (RM) family of codes, which has been around for over half a century [4]. RM codes have gained more attention recently due their high effectiveness for short blocklenghts and the recent development of a near-capacity-achieving decoder called recursive projection-aggregation (RPA) [5]. The major drawback of this decoder is the high complexity due to its recursive nature, which makes RPA in its original form not suitable for practical implementation. Multiple attempts to reduce its complexity have already been done, for example by collapsing several levels of recursive projection and aggregation into a single level using a technique called collapsed projection aggregation (CPA) [6], replacing the recursive structure by an iterative structure [7], reducing the number of branches created during projection [8] [9], or by applying a combination of these techniques [10].

The mentioned technique of removing branches is called pruning, and is currently one of the most promising options for simplifying RPA for hardware implementation. Pruning can, in certain cases, reduce the decoder complexity by up to 87% while requiring only a 0.1 dB higher E_b/N_0 to achieve the same frame error rate (FER), as shown in [8]. The selection of branches that is retained in these cases, is either chosen randomly, or spread evenly over the complete set of branches. Although previous research analyses how different amounts of pruning affect performance, there has been little research into the optimal way to choose branches to prune. There might exist specific selections of branches that make an RPA-based decoder perform overall better than other, equal-sized selections of branches. This paper aims to explore whether such selections of branches exist, by mapping the contributions of individual branches and sets of branches.

Section II gives an overview of the mechanics of RM codes and the RPA decoder. Section III describes the different analyses performed to uncover the contributions of the different elements of an RPA decoder. In section IV, a description can be found of how these findings can be utilized in practice, and section V shows the simulation results for the applied methods.

II. Preliminaries

A. Reed-Muller codes

The phrase "code" refers to a finite set of codewords of a fixed length n, used to encode a fixed number of message bits k at a time. RM codes and their properties are commonly

denoted as RM(m, r)[1]. Both of these notations specify a set of properties that uniquely defines an RM code. The blocklength is indicated by n, and it holds that $n = 2^m$. The number of encoded message bits k is equal to the number of basis vectors in the code and depends on the order r of the code in the following way:

$$k = \sum_{i=0}^{r} \binom{m}{i} \tag{1}$$

Moreover, r influences the minimum Hamming distance d between codewords, which is determined by $d = 2^{m-r}$. This last property is relevant, because the maximum number of errors in a corrupt codeword that can be corrected using hard-decision maximum likelihood decoding, is equal to $\lfloor \frac{d-1}{2} \rfloor$.

The RM code where $r = 0$ is a special case, and is equal to a repetition code. On the other hand, when $r = m$, it will hold that $k = n$ according to (1). The latter code will add no redundancy bits during the encoding process, and will thus be unable to correct errors, which is also explained by the fact that $d = 0$ in this case. In conclusion, these two extreme cases are not practical, and therefore it always holds that $0 < r < m$ for RM codes used in practice.

Next, we will explain the process of encoding messages using an RM code. If the information bits of message $\mathbf{x}$ are denoted by the vector $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & ... & x_k \end{bmatrix}$, then the corresponding codeword is given by $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & ... & y_n \end{bmatrix}$ and is generated by $\mathbf{y} = \mathbf{x}\mathbf{G}_{(m,r)}$. Here, $\mathbf{G}_{(m,r)}$ is the k-by-n generator matrix, the rows of which are the basis vectors of RM(r, m).

Any $\mathbf{G}_{(m,r)}$ for a different order or blocklength can be constructed from a lower-blocklength code as follows:

$$\mathbf{G}_{(m,r)} = \begin{bmatrix} \mathbf{G}_{(m-1,r)} & \mathbf{G}_{(m-1,r)} \\ 0 & \mathbf{G}_{(m-1,r-1)} \end{bmatrix} \tag{2}$$

Any $\mathbf{G}_{(m,r)}$ can also be constructed from a higher-order, same-blocklength code by sorting all rows by their Hamming weight, and using only the first k rows. Using these two properties, any $\mathbf{G}_{(m,r)}$ can be recursively constructed from one known generator matrix, $\mathbf{G}_{(1,1)}$, which is defined as:

$$\mathbf{G}_{(1,1)} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{3}$$

B. Decoding of RM codes using RPA

Known methods to decode RM codes make use of majority voting, including the original decoding algorithm by Reed himself [4]. Of these decoders, the recently invented RPA decoder [5] performs significantly better than alternatives in the short blocklength regime and is the subject of this paper.

RPA utilizes the fact that first-order RM codes can be decoded optimally using a Fast Hadamard Transform (FHT) [11]. The global structure of RPA consists of recursively projecting higher-order RM codes onto lower orders until a number of first-order codes is obtained, which can be decoded using FHT. The decoded RM$(m - r + 1, 1)$ codes are then recursively aggregated until a single decoded RM(m, r) code is obtained. How each of these recursive steps works for a codeword sent over a binary symmetric channel (BSC) will be explained next, followed by an overview of the adaptations needed to apply RPA for an additive white gaussian noise (AWGN) channel.

1) BSC: RPA projects a noisy RM(m, r) codeword received over a BSC onto $n-1$ codewords of RM$(m-1, r-1)$ which thus have length $n/2$. Each of those projections will in turn be projected onto $n/2-1$ codewords of RM$(m-2, r-2)$, etc. Each projection $\mathbf{y}_i$ is created according to Algorithm 1 which, for each bit of $\mathbf{y}_{\text{out}}$, combines two bits of $\mathbf{y}_{\text{in}}$ into one, the coordinates of which depend on the projection number $i \in \{1, ..., n\}$. In the case of a BSC, these two bits are combined through a bit-wise XOR operation on line 8. This process will eventually result in a number N_{total} of codewords of RM$(m - r + 1, 1)$ equal to:

$$N_{\text{total}} = \prod_{x=0}^{r-2} (2^{m-x} - 1) \tag{4}$$

Algorithm 1 `Projection`

Input: $\mathbf{y}_{\text{in}}(0 : n-1), n, i$
Output: $\mathbf{y}_{\text{out}}(0 : \frac{n}{2}-1)$

1: **if** $i < \frac{n}{2}$ **then**
2: $\quad \mathbf{y}_{\text{out}}(0 : \frac{n}{4}-1) \leftarrow$ `Projection`$(\mathbf{y}_{\text{in}}(0 : \frac{n}{2}-1), \frac{n}{2}, i)$
3: $\quad \mathbf{y}_{\text{out}}(\frac{n}{4} : \frac{n}{2}-1) \leftarrow$ `Projection`$(\mathbf{y}_{\text{in}}(\frac{n}{2} : n-1), \frac{n}{2}, i)$
4: **else**
5: $\quad$ **for** $j = 0 : \frac{n}{2}-1$ **do**
6: $\quad\quad z_1 \leftarrow \mathbf{y}_{\text{in}}(j)$
7: $\quad\quad z_2 \leftarrow \mathbf{y}_{\text{in}}($`bi2de(de2bi(`$i$`)` $\oplus$ `de2bi(`j`))`$)$
8: $\quad\quad \mathbf{y}_{\text{out}}(j) \leftarrow z_1 \oplus z_2$
9: $\quad$ **end for**
10: **end if**

After decoding all projections of a certain $\mathbf{y}$, the original and decoded projections are aggregated to provide an estimate for the original transmitted value of $\mathbf{y}$, indicated by $\hat{\mathbf{y}}$. This estimate is achieved through majority voting by each of the projections $\mathbf{y}_i$ for the bits of $\mathbf{y}$ through which they were created. If $\hat{\mathbf{y}} = \mathbf{y}$, then the decoder will continue to aggregate the set of projections that $\mathbf{y}$ itself was part of in the previous recursion level. If this is not yet the case, then the decoder will assign $\mathbf{y} \leftarrow \hat{\mathbf{y}}$ and project, decode and aggregate $\hat{\mathbf{y}}$ again until its value converges to $\mathbf{y}$, for at most $N_{\max}$ repetitions.

2) AWGN channel: As mentioned in section II-A, a decoder for a BSC can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. However, this limit can be surpassed with a soft-decision decoder, using the log-likelihood ratio (LLR) values from an AWGN channel as an input. In that case, the `XOR` function on line 8 of algorithm 1 must be replaced by the following assignment:

$$\mathbf{y}_{\text{out}}(j) \leftarrow \ln(\exp(z_1+z_2)+1) - \ln(\exp(z_1)+\ln(\exp(z_2)) \tag{5}$$

For all simulations in this paper, this is replaced by the min-sum-approximation [9] which requires less hardware for its implementation:

$$\mathbf{y}_{\text{out}}(j) \leftarrow \text{sign}(z_1)\text{sign}(z_2)\min(|z_1|, |z_2|) \tag{6}$$

[1]The notation RM(r, m) is also common. However, since $r > m$ in practice, confusion should not be possible.

As for the aggregation, LLR values cannot be used directly for a majority voting mechanism. Instead, the new LLR values of $\hat{\mathbf{y}}$ are determined by taking the averages of the corresponding LLR values of its projections.

III. Analysis of Contribution of Branches

Each projection onto a lower order of either the received codeword or of another projection, is called a branch. The contributions of branches at all recursion levels are assessed by projecting the original codeword alongside the received noisy LLR values in the decoder, and counting the number of corrections done each time the RPA decoder is invoked. In this context, a bit is seen as 'corrected' if, after the decoder has been invoked, the corresponding LLR value's sign equals that of the corresponding LLR value in the corresponding projection of the original codeword and additionally, has been flipped. In the remainder of this section, we analyse the decoding of 10^5 samples of both RM$(5,3)$ and RM$(6,3)$ sent over an AWGN channel with 2.0 dB E_b/N_0.

A. Individual projection contribution

For this analysis, the number of corrections are visualised as a bar graph in in figures 1 and 2. The number of bit corrections, which can be multiple per frame, are shown as bars. The number of times at least one bit correction took place in a frame, also called the frame corrections, are added up per branch and plotted with a red line in figures 1 and 2. In both figures, the top plot represents the corrections made by aggregating first-order projections back into each of the second-order projections they belonged to. The bottom plot shows all the corrections made by each of the first-order decoders applying the FHT.

Figures 1 and 2 show that the distribution of corrections among the branches is notably uniform, and suggest that no particular decoders contribute above or below average to the decoding process. This could imply that the performance of a pruned decoder cannot be improved by a utilizing a selection based on individual performance of each branch.

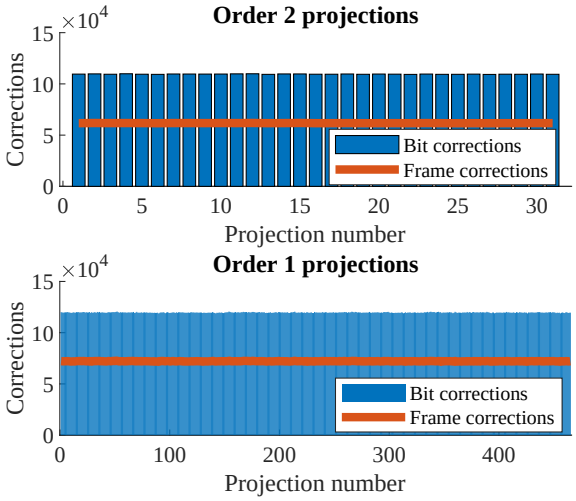

Fig. 1. Corrections per branch in the decoder for RM$(5,3)$

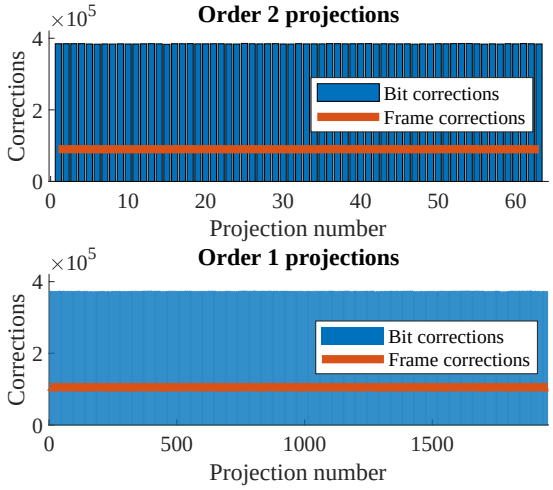

Fig. 2. Corrections per branch in the decoder for RM$(6,3)$

B. Combined projection contribution

However, the results of the previous section do not exclude the possibility that performance is impacted by the choice of implemented branches due to correlation in corrected errors. Hence, the next analysis shows how frequently corrections are done simultaneously by every possible pair of first-order projections. All first-order projections of RM$(5,3)$ are enumerated and represented on both axes in figure 3. Each value depicted in this graph at coordinate (x,y) shows the normalized frequency with which first-order projection x and projection y correct an error simultaneously. This does not necessarily mean that they correct the exact same error. Figure 3 is therefore an indicator of an upper limit to the overlap between each pair of first-order projections.

Above-average figures can be found at the diagonal yellow line, and as scattered orange dots. The diagonal yellow line can be explained by the fact that these values are the result of projections being compared to themselves. The orange dots are caused by pairs of projections that are both very similar, as will be explained in section III-C. Taking into account their uniform individual error correcting performance, it can be reasoned that a certain selection of first-order projections that shows a low overlap will perform better than one that shows a high overlap, and will 'catch' a larger total number of errors. This theory will be further explored in section III-C and tested in section IV.

C. Unique projections

The scattered orange dots in figure 3 are indicators of pairs of projections that are identical with regard to the received third-order codeword, meaning that they are indirectly constructed from the same LLR values. This can be illustrated with the following example, in a hard-decision decoder for simplicity, where y_i^r indicates the i-th projection onto order r. The first second-order projection and its first first-order

TABLE I
DISTRIBUTION OF DUPLICATES AND UNIQUES OF RM(5,3)

second-order proj. index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	30	31
duplicate first-order	0	1	1	3	3	3	3	7	7	7	7	7	7	7	7	15	15	...	15	15
unique first-order	15	14	14	12	12	12	12	8	8	8	8	8	8	8	8	0	0	...	0	0

TABLE II
DISTRIBUTION OF DUPLICATES AND UNIQUES OF RM(6,3)

second-order proj. index	1	2	3	4	5	6	7	8	9	...	15	16	17	...	31	32	33	...	62	63
duplicate first-order	0	1	1	3	3	3	3	7	7	...	7	15	15	...	15	31	31	...	31	31
unique first-order	31	30	30	28	28	28	28	24	24	...	24	16	16	...	16	0	0	...	0	0

TABLE III
DISTRIBUTION OF DUPLICATES AND UNIQUES OF RM(7,3)

second-order proj. index	1	2	3	4	5	6	7	8	...	15	16	...	31	32	...	62	63	...	126	127
duplicate first-order	0	1	1	3	3	3	3	7	...	7	15	...	15	31	...	31	31	...	63	63
unique first-order	63	62	62	60	60	60	60	56	...	56	48	...	48	32	...	32	0	...	0	0

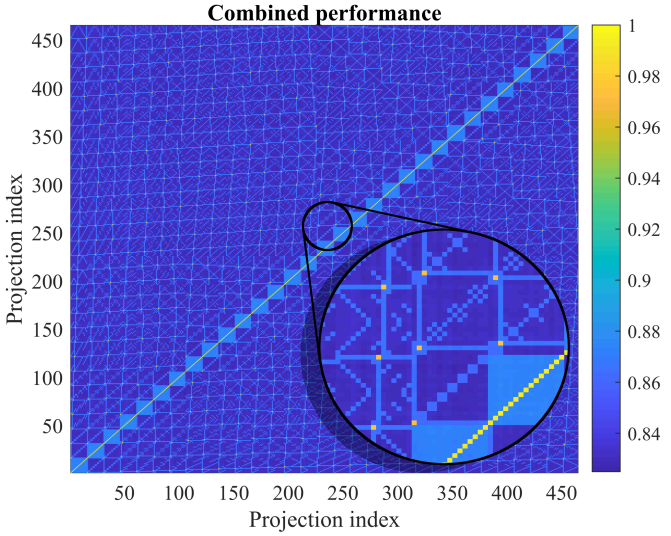

Fig. 3. Combined contribution of branches of RM$(5,3)$

projection look as follows, given that $\mathbf{y}^3$ denotes a received third-order message.

$$\mathbf{y}^3 = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & \dots & y_n \end{bmatrix} \tag{7}$$

$$\mathbf{y}_1^2 = \begin{bmatrix} (y_1 \oplus y_2) & (y_3 \oplus y_4) & \dots \end{bmatrix} \tag{8}$$

$$\mathbf{y}_1^1 = \begin{bmatrix} (y_1 \oplus y_2 \oplus y_3 \oplus y_4) & \dots \end{bmatrix} \tag{9}$$

The second second-order projection and its first first-order projection are as follows:

$$\mathbf{y}_1^2 = \begin{bmatrix} (y_1 \oplus y_3) & (y_2 \oplus y_4) & \dots \end{bmatrix} \tag{10}$$

$$\mathbf{y}_1^1 = \begin{bmatrix} (y_1 \oplus y_3 \oplus y_2 \oplus y_4) & \dots \end{bmatrix} \tag{11}$$

Considering the properties of the XOR function, it can be seen that $\mathbf{y}_1^1$ in (9) and (11) are duplicates. In fact, according to [6], the number of unique projections among all first-order projections of any RM code is equal to:

$$N_{\text{unique}} = \prod_{x=0}^{r-2} \frac{2^{m-x} - 1}{2^{1+x} - 1} \tag{12}$$

which means that, taking into account (4), the number of duplicates is then equal to:

$$N_{\text{duplicate}} = \frac{N_{\text{total}}}{N_{\text{unique}}} = \prod_{x=0}^{r-2} (2^{1+x} - 1) \tag{13}$$

For RM(m,r), each of the $2^m - 1$ second-order projections projects onto $2^{m-1} - 1$ first-order projections. The first few of those $2^{m-1} - 1$ are duplicates of those produced by second-order projections with a lower index. The distribution of duplicates and uniques for RM(m,3) are displayed in table I, II and III. In all three tables, the second row shows how many of the first-order projections are duplicates[2]. The third row shows the number of remaining uniques[2], which will be used as a basis for pruning in the next section.

IV. NEW PRUNING SELECTION

Next, we show that the number of duplicate and unique first-order projections in a pruned decoder influence its error-correcting performance. Figure 3 shows that projection pairs that are not duplicates have a lower chance of correcting the same error. Therefore, a decoder containing less duplicates[2] and more uniques[2] is expected to perform better than a decoder that contains more duplicates and less uniques. Two decoders are simulated to test and show the difference in performance between a decoder pruned with more uniques and a decoder pruned with more duplicates.

One of these two decoders will be pruned in such a way that it contains as many as possible unique first-order projections, and no duplicates. Tables I, II and III show that the second half of the second order-projections produce no unique first-order projections, and are thus pruned away in this decoder. Moreover, of the remaining second-order projections, only the unique first-order projections are retained and the duplicates are left out. As mentioned in section III-C, the former always have a lower projection index than the latter. For example, for

[2]The phrases 'duplicates' and 'uniques' always refer to first-order projections in this paper. For third-order RM codes, second-order projections are always unique with respect to each other.

y_4^2 of RM(5,3), the first 3 first-order projections are duplicates, and the last 12 are uniques. Therefore, the pruned RM(5,3) decoder will contain all 15 first-order projections of its first second-order projection, the last 14 of the second, the last 14 of the third, the last 12 of the fourth, etc. according to the third row of table I. The pruned RM(6,3) and RM(7,3) decoders are constructed in the same way, using table II and III respectively.

The second type of decoder serves the purpose of showing how a pruning selection containing many duplicates can decrease the performance of the decoder. This pruned RM(5,3) decoder contains all 15 first-order projections of its first second-order projection, the first 14 of the second, the first 14 of the third, the first 12 of the fourth, etc. according to the third row of table I. Note that the only difference between both decoders is the fact that the one pruned with uniques contains the *last* few first-order projections of each set, and the one pruned with duplicates contains the *first* few projections of each set.

Previous research has resulted in a different decoder [6] that, like the first of the two decoders described above, only contains unique first-order projections. This CPA decoder does however not have a recursive structure similar to RPA decoders. Because of its equal number of first-order projections, the simulations of two pruned decoders described above are compared to a simulation of a CPA decoder, in addition to the unpruned RPA decoder they are derived from.

V. Results

The four decoders, being RPA, pruned RPA with duplicates, pruned RPA with uniques, and CPA, are simulated for RM$(5,3)$, RM$(6,3)$ and RM$(7,3)$. The frame error rates (FER) for these three codes when applied for several Eb/N_0 values are shown in figure 4, 5 and 6 respectively. It can be observed that with all three RM codes, the pruned RPA decoder that contains more uniques performs better than the pruned RPA decoder with more duplicates, although only marginally. Due the almost equal structure of the two and the equal distribution of number of first-order projections over the second-order projections, it can be stated that the difference in performance is purely the result of the specific choice of first-order projections.

Moreover, for all three RM codes, the efficiently pruned decoder performs almost equal to the unpruned RPA decoder and performs slightly better than the CPA decoder, which can be an advantage. In order to implement CPA, complex hardware is needed to aggregate several decoded first-order projections back into one third-order or higher output message. This hardware is more complex than the hardware needed for aggregation within an RPA decoder. Therefore, it is useful to have a method of pruning to create a decoder that shares with CPA the low number of first-order decoders, but shares with RPA the relatively low complexity of the aggregation structure, which in addition performs better than CPA. One disadvantage however, is the uneven distribution of unique first-order projections, as this might complicate hardware implementation.

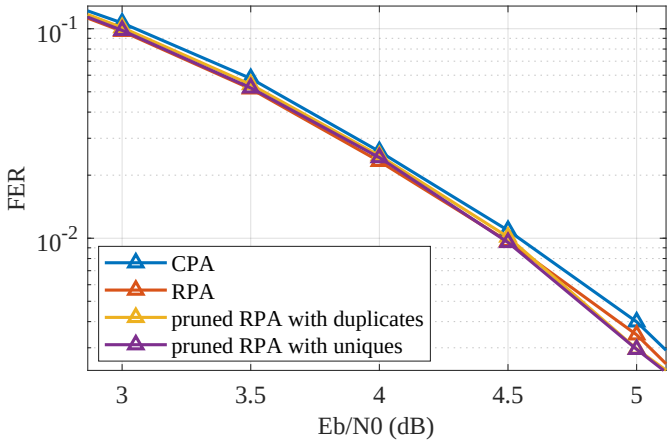

Fig. 4. Simulation results of several decoders for RM(5,3)

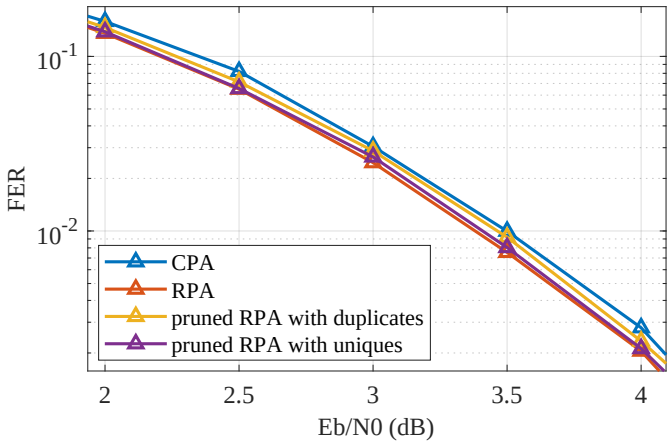

Fig. 5. Simulation results of several decoders for RM(6,3)

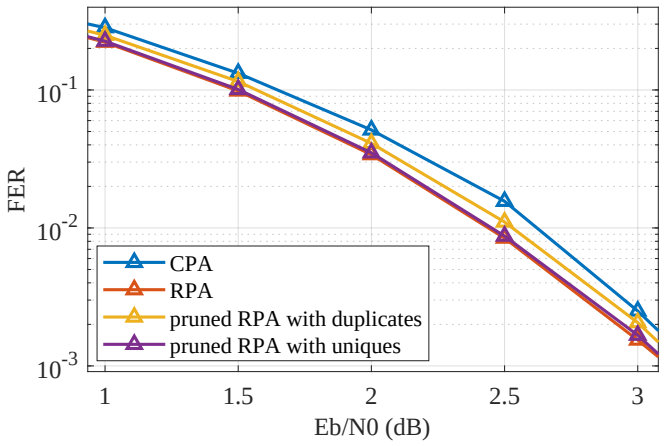

Fig. 6. Simulation results of several decoders for RM(7,3)

VI. Conclusions

In this paper, we have analysed the efficiency of individual first order projections within an RM(m,3) decoder, and evaluated the overlap in their contributions to the overall decoding process. It can be concluded that each first order decoder on its own contributes equally much. However, some first order decoders contribute almost exactly the same as other first order decoders. This paper has shown that choosing first order projections which show little overlap, can contribute to a higher efficiency of a pruned decoder when compared to a very similar decoder where pruned branches are chosen slightly differently. How large this advantage can be exactly has to be assessed further in future work. Lastly, although the pruned decoder is easier to implement in hardware than CPA, future implementations in hardware will also benefit from a more even distribution of unique first order projections.

References

[1] N. H. Mahmood, S. Böcker *et al.*, "White Paper on Critical and Massive Machine Type Communication Towards 6G," no. 11, pp. 1–36, 2020. [Online]. Available: http://arxiv.org/abs/2004.14146

[2] R. G. Gallager, *Low-Density Parity-Check Codes*, 1962, vol. 8, no. 1.

[3] M. C. Coşkun, G. Durisi *et al.*, "Efficient error-correcting codes in the short blocklength regime," *Physical Communication*, vol. 34, pp. 66–79, 2019. [Online]. Available: https://doi.org/10.1016/j.phycom.2019.03.004

[4] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, 1954.

[5] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed-Muller codes," pp. 2064–2068, 2019.

[6] M. Lian, C. Häger, and H. D. Pfister, "Decoding Reed–Muller Codes Using Redundant Code Constraints," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020. [Online]. Available: https://ieeexplore-ieee-org.dianus.libr.tue.nl/document/9174087/

[7] M. Hashemipour-Nazari, K. Goossens, and A. Balatsoukas-Stimming, "Hardware Implementation of Iterative Projection-aggregation Decoding of Reed-muller Codes," vol. 1, no. 1, pp. 8293–8297, 2021.

[8] D. Fathollahi, N. Farsad *et al.*, "Sparse Multi-Decoder Recursive Projection Aggregation for Reed-Muller Codes," *IEEE International Symposium on Information Theory - Proceedings*, vol. 2021-July, pp. 1082–1087, 2021.

[9] J. Li, S. M. Abbas *et al.*, "Reduced Complexity RPA Decoder for Reed-Muller Codes," *2021 11th International Symposium on Topics in Coding, ISTC 2021*, 2021.

[10] Q. Huang and B. Zhang, "Pruned Collapsed Projection-Aggregation Decoding of Reed-Muller Codes," no. 1, 2021. [Online]. Available: http://arxiv.org/abs/2105.11878

[11] F. J. MacWilliams and N. J. a. Sloane, "The Theory of Error-Correcting Codes (North-Holland Mathematical Library)," vol. 16, pp. 419–426, 1988. [Online]. Available: