Decoding Reed–Muller Codes Using Redundant Code Constraints

Mengke Lian*, Christian Häger‡, and Henry D. Pfister*
*Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina
‡Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

***Abstract*—The recursive projection–aggregation (RPA) decoding algorithm for Reed–Muller (RM) codes was recently introduced by Ye and Abbe. We show that the RPA algorithm is closely related to (weighted) belief-propagation (BP) decoding by interpreting it as a message-passing algorithm on a factor graph with redundant code constraints. We use this observation to introduce a novel decoder tailored to high-rate RM codes. The new algorithm relies on puncturing rather than projections and is called recursive puncturing–aggregation (RXA). We also investigate collapsed (i.e., non-recursive) versions of RPA and RXA and show some examples where they achieve similar performance with lower decoding complexity.**

I. Introduction

Reed–Muller (RM) codes were introduced by Muller [1] and a bounded-distance decoding algorithm was given by Reed [2]. Under maximum-likelihood (ML) decoding, RM codes achieve capacity on the binary erasure channel [3] and they also provide excellent performance at low to medium block lengths over the additive white Gaussian noise (AWGN) channel [4], [5]. Due to the intractability of ML decoding, an important question is how to approach ML performance with reasonable decoding complexity in practice. Many recent approaches exploit the symmetry of RM codes by making use of their large automorphism group[1] [4], [5], [7], [8]. This typically results in code representations with many redundant code constraints, e.g., overcomplete parity-check matrices where the number of rows is much larger than the rank [7], [9]. Other options for decoding RM codes include the recursive list decoder introduced by Dumer and Shabunov [16] and the successive-cancellation list decoder for polar codes [8], [10]. These will be discussed further in Section V.

This paper focuses on the recursive projection–aggregation (RPA) algorithm that was recently introduced in [4], [5]. RPA decoding achieves excellent performance on low-rate (2nd- and 3rd-order) RM codes. It also significantly outperforms comparable polar codes under successive-cancellation list decoding. In this work, we show that the RPA algorithm has a natural interpretation on a factor graph with generalized check constraints and can be seen as a version of (weighted) belief-propagation (BP) decoding using many redundant code constraints. We use this observation to introduce a new decoding approach tailored to high-rate codes. For a detailed comparison between RPA and previous decoding approaches, such as [11]–[13] and [14]–[16], see [5, Sec. V-B].

To begin, we provide a simple overview of RM codes that illustrates how their recursive structure and their large automorphism group together imply that they satisfy a very large set of code constraints. We then give a high-level description of the algorithms and contributions in this paper with the help of the RM code table shown in Fig. 1. To that end, let $\mathrm{RM}(r,m) \subseteq \mathbb{F}_2^N$ denote the set of codewords in the r-th order RM code of length $N = 2^m$. The well-known recursive definition of $\mathrm{RM}(r,m)$ [17, p. 374] is given by

$$\{(\mathbf{u},\mathbf{u}+\mathbf{v}) \,|\, \mathbf{u}{\in}\mathrm{RM}(r,m-1), \mathbf{v}{\in}\mathrm{RM}(r-1,m-1)\}\,, \quad (1)$$

where $(\mathbf{u},\mathbf{w})$ denotes vector concatenation. For $\mathbf{u},\mathbf{w} \in \mathbb{F}_2^{N/2}$ with $(\mathbf{u},\mathbf{w}) \in \mathrm{RM}(r,m)$, this implies that (i) $\mathbf{u} \in \mathrm{RM}(r,m-1)$, i.e., *puncturing* the second half of the codeword gives a shorter RM codeword of the same order, and (ii) $\mathbf{v} = \mathbf{u}+\mathbf{w} \in \mathrm{RM}(r-1,m-1)$, i.e., summing the two codeword halves *projects* onto a shorter RM code with reduced order. Of course, these statements remain true even after reordering the code bits using a permutation in the code's automorphism group. Thus, there are in fact many different puncturing and projection patterns that result in RM subcodes. In RPA decoding, these subcodes are decoded recursively until one reaches a 1st-order (i.e., augmented Hadamard) code, for which there exist efficient decoders based on the fast Hadamard transform (FHT) [17]. The schematic decoding path taken by RPA is illustrated by the solid red line in Fig. 1.

The complexity of RPA increases significantly with each additional recursion stage and RPA decoding is thus limited to low-rate codes in practice. To alleviate this problem, we propose a new algorithm, which is similar in spirit to RPA, that relies on puncturing instead of projection. The new algorithm is called recursive puncturing–aggregation (RXA). It uses all possible $\mathrm{RM}(r,m-1)$ subcodes and traverses upwards in the RM tableau, as illustrated by the solid blue line in Fig. 1. The base case for RXA is the $\mathrm{RM}(r,r+2)$ (i.e., extended Hamming) code which also has a FHT-based decoder [18].

We further investigate collapsed, i.e., non-recursive, versions of RPA and RXA. We provide a theoretical justifica-

The work of M. Lian and H. D. Pfister was supported in part by the National Science Foundation (NSF) under Grant No. 1718494. The work of C. Häger was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant No. 749798. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors. Please send correspondence to henry.pfister@duke.edu.

[1]The automorphism group of a code $\mathcal{C}$ is defined as $\{\pi \in \mathcal{S}_N \,|\, \mathbf{x}^\pi \in \mathcal{C}, \forall \mathbf{x} \in \mathcal{C}\}$, where $\mathcal{S}_N$ is the symmetric group on N elements, i.e., $\pi \in \mathcal{S}_N$ is a bijective mapping (or permutation) from $[N]$ to itself, and $\mathbf{x}^\pi$ denotes a permuted vector, i.e., $x_i^\pi = x_{\pi(i)}$.

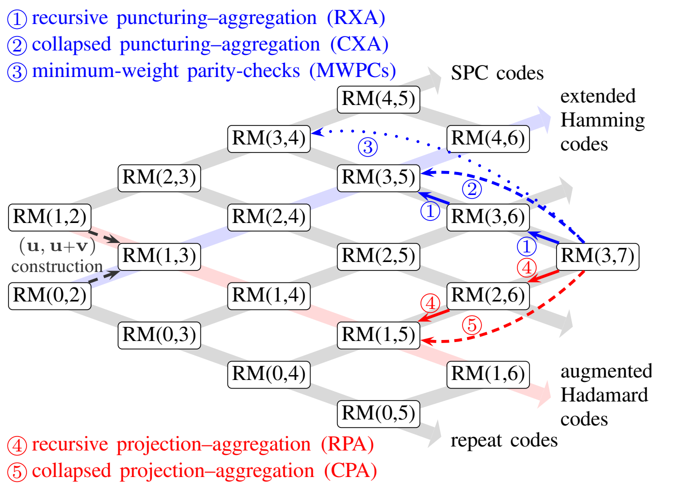


Fig. 1: Standard tableau of Reed–Muller (RM) codes and schematic illustration of all algorithms considered in this paper. (SPC: single parity-check)

tion for using collapsed algorithms by showing that multiple recursive stages result in reuse of the same subcodes. We then propose two new algorithms, called collapsed projection–aggregation (CPA) and collapsed puncturing–aggregation (CXA), and show that they can achieve similar performance as their recursive counterparts with lower complexity. CPA and CXA correspond to the dashed lines in Fig. 1 and directly project or puncture onto the base codes.

Lastly, we connect the decoding algorithms in this paper to previous approaches based on overcomplete parity-check matrices that contain all minimum-weight dual codewords as rows [7], [9]. In particular, we show that the factor graph for CXA with one additional stage of puncturing (i.e., to single parity-checks instead of extended Hamming codes, see the blue dotted line in Fig. 1) is equivalent to the factor graph that contains all minimum-weight parity-checks (MWPCs).

In summary, the contributions in this paper are as follows:

- We show that RPA is closely related to BP decoding by interpreting it as a message-passing algorithm on a highly redundant factor graph representation of the code.
- We propose a new algorithm, called RXA, which allows for efficient decoding of high-rate RM codes.
- We propose collapsed versions of RPA and RXA, called CPA and CXA, and show that they sometimes achieve similar performance with lower complexity[2].
- We highlight the connections between this work, RPA, and previous approaches based on using all MWPCs.

II. Factor Graphs for Reed–Muller Codes

In order to compare RPA and BP decoding, we start by reviewing a few factor graph representations of $\mathrm{RM}(r,m)$. For an introduction to factor graphs, see [19].

In general, the factor graphs in this paper contain four distinct node types, which are collected in the sets $\mathcal{V}, \mathcal{C}, \mathcal{V}_{\mathrm{h}}, \mathcal{C}_{\mathrm{g}}$:

- $\mathcal{V}$: variable nodes (VNs), corresponding to the code bits of $\mathrm{RM}(r,m)$, where $|\mathcal{V}| = 2^m$,
- $\mathcal{C}$: check nodes (CNs), corresponding to projections, i.e., the summation of code bits,
- $\mathcal{V}_{\mathrm{h}}$: hidden VNs (of degree 2), corresponding to the code bits of $\mathrm{RM}(r-1,m-1)$ subcodes,
- $\mathcal{C}_{\mathrm{g}}$: generalized CNs, corresponding to $\mathrm{RM}(r,m-d)$ or $\mathrm{RM}(r-d,m-d)$ subcode constraints for $1 \le d \le r-1$.

The number of nodes of each type and the graph connectivity (including the node degrees) depend on the particular code representation. As an example, the factor graph that can be inferred from the $(\mathbf{u},\mathbf{u}+\mathbf{v})$ construction in (1) is shown in Fig. 2(a). It consists of one generalized CN corresponding to $\mathrm{RM}(r,m-1)$ for the first codeword half and one generalized CN corresponding to $\mathrm{RM}(r-1,m-1)$ that constrains the sum of the two codeword halves. Moreover, one may use the fact that $\mathrm{RM}(r-1,m) \subset \mathrm{RM}(r,m)$ [17, p. 377] to see that the second half of the codeword also forms a valid codeword in $\mathrm{RM}(r,m-1)$, leading to one additional subcode constraint.

A. Redundant Factor Graphs

Redundant code constraints can be obtained by exploiting the code's automorphism group. Fig. 2(a) shows the implicit factor graph for RPA decoding, which is based on projecting onto the $2^m - 1$ different $\mathrm{RM}(r-1,m-1)$ subcodes. Note that, for factor graphs with generalized CNs, the ordering of edges corresponding to the subcode bits is important. Here, we neglect this issue to allow for a concise high-level description of all decoding algorithms. A precise definition of the factor graph connectivity, including proper indexing of subcode bits, can be found in the extended version of this paper [20].

B. Belief-Propagation Decoding

Once the factor graph is defined, many decoding algorithms are defined automatically by standard variations of BP update rules. For example, assume VN messages are updated with

$$\lambda^{(t)}_{v\to c} = \ell_v + \sum_{c' \in \partial v \setminus c} \hat{\lambda}^{(t)}_{c' \to v}, \tag{2}$$

and outgoing CN messages are updated with

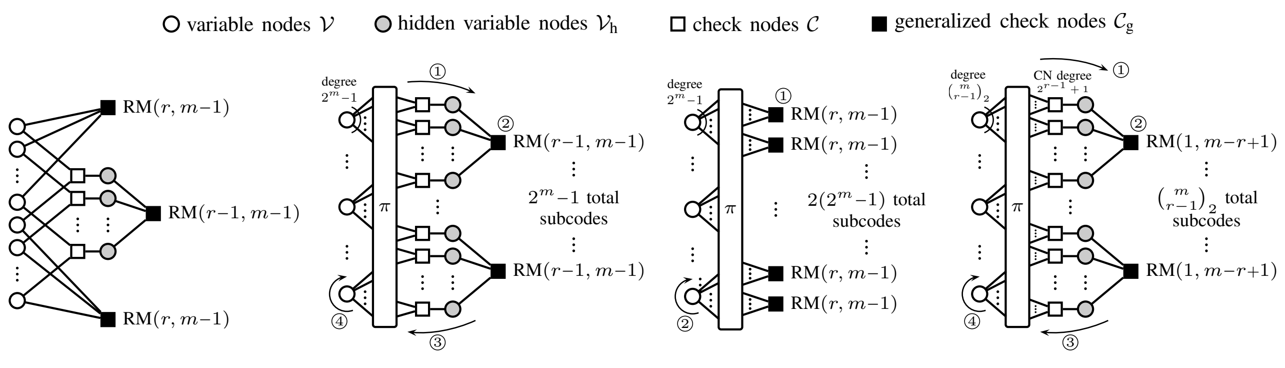
$$\hat{\lambda}^{(t)}_{c\to v} = 2\tanh^{-1}\left(\prod_{v' \in \partial c \setminus v} \tanh\left(\frac{\lambda^{(t)}_{v'\to c}}{2}\right)\right), \tag{3}$$

where t refers to the iteration number and ℓ_v corresponds to the channel log-likelihood ratio (LLR). For hidden (degree-2) VNs in $\mathcal{V}_{\mathrm{h}}$, we have $\ell_v = 0$ and (2) corresponds to a simple message forwarding. Outgoing messages for generalized CNs are updated by computing the corresponding extrinsic bit-wise posterior LLRs. This is generally intractable for $r > 1$, which motivates the use of recursive approaches.

III. Recursive Decoding Algorithms

In this section, we revisit RPA decoding as a message-passing algorithm and highlight the differences compared to standard BP decoding. We then describe the new RXA decoding algorithm.

[2]The journal version [5] of [4] appeared after this paper was submitted and also considers a simplified decoder that partially collapses the RPA recursion.



(a) $(\mathbf{u}, \mathbf{u}+\mathbf{v})$ construction (b) RPA (c) RXA (d) CPA

Fig. 2: Factor graphs for RM(r,m). Circled numbers and arrows indicate message-passing schedules for the decoding algorithms.

Algorithm 1: RPA [4] and CPA

```
   Input: LLRs $\boldsymbol{\ell}$, params $r, m$, flags $E, C \in \{0,1\}$
   Output: (binary) output LLRs $\hat{\boldsymbol{\ell}}$ where $\ell_v \in \{\pm\infty\}$
 1 if $r = 1$ then
 2     $\hat{\boldsymbol{\ell}}$ = FHT decode $\boldsymbol{\ell}$: hard-ML to $\pm\infty$ ($E$=0) or extrinsic ($E$=1)
 3 else
 4     construct the RPA or, if $C = 1$, the CPA factor graph for
         RM$(r,m)$ with node sets $\mathcal{V}, \mathcal{V}_\mathrm{h}, \mathcal{C}, \mathcal{C}_\mathrm{g}$ (see Figs. 2(b) and 2(d))
 5     initialize $\lambda^{(0)}_{v\to c} = \ell_v \; \forall v \in \mathcal{V}$
 6     for $t = 1, \dots, T_\mathrm{max}$ do
           // projection, step (1) in Fig. 2(b)
 7         update $\hat{\lambda}^{(t)}_{c\to v}$ via (3) $\forall c \in \mathcal{C}$ and $\lambda^{(t)}_{v\to c}$ via (2) $\forall v \in \mathcal{V}_\mathrm{h}$
           // subcode decoding, step (2)
 8         if $C = 0$ then
 9             $(\hat{\lambda}^{(t)}_{c\to v})_{v\in\partial c} = \mathrm{RPA}\left((\lambda^{(t)}_{v\to c})_{v\in\partial c}, r-1, m-1\right) \; \forall c \in \mathcal{C}_\mathrm{g}$
10         else  // jump to line 2 and apply FHT decoding
11             $(\hat{\lambda}^{(t)}_{c\to v})_{v\in\partial c} = \mathrm{RPA}\left((\lambda^{(t)}_{v\to c})_{v\in\partial c}, 1, m-r+1\right) \; \forall c \in \mathcal{C}_\mathrm{g}$
           // backward update, step (3)
12         update $\lambda^{(t)}_{v\to c}$ via (2) $\forall v \in \mathcal{V}_\mathrm{h}$ and $\hat{\lambda}^{(t)}_{c\to v}$ via (3) $\forall c \in \mathcal{C}$
           // VN update, step (4)
13         update $\lambda^{(t)}_{v\to c}$ via (4) ($E$=0) or (6) ($E$=1) for all $v \in \mathcal{V}$
14     compute $\hat{\ell}_v$ via (5) and map to $\pm\infty \; \forall v \in \mathcal{V}$
15 return $\hat{\boldsymbol{\ell}}$
```

A. Recursive Projection–Aggregation

The message-passing version of RPA is defined in Algorithm 1. The inputs to the algorithm are the channel LLRs $\boldsymbol{\ell}$, the RM parameters r and m, and the flags $E, C \in \{0,1\}$. The flag E indicates if intrinsic ($E = 0$) or extrinsic updates ($E = 1$) should be used and the flag C indicates if the original ($C = 0$) or the collapsed version ($C = 1$) should be used. On line 2, the base codes ($r = 1$) are decoded via FHTs using either ($E = 0$) intrinsic hard-ML decoding with LLR decisions mapped to $\pm\infty$ or ($E = 1$) extrinsic soft decoding [21]. If $r \neq 1$, then the decoding starts by constructing the factor graph shown in Fig. 2(b) (line 4) and initializing all outgoing VN messages to the channel LLRs (line 5). The algorithm then iterates T_max times over the following four steps:

1) Projection: All CNs update their outgoing messages according to the standard CN update rule (3). Afterwards, the hidden VNs forward these messages to the generalized CNs. One can also show that (3) is equivalent to the projection-step update equation in the original RPA algorithm because $\ln(e^{a+b}+1) - \ln(e^a + e^b) = 2\tanh^{-1}\left(\tanh\frac{a}{2}\tanh\frac{b}{2}\right)$ [20].

2) Subcode decoding: The incoming messages for each generalized CN $c \in \mathcal{C}_\mathrm{g}$ are collected into a vector $(\lambda^{(t)}_{v\to c})_{v\in\partial c}$ which serves as the input for the recursive RPA decoding of the subcodes (line 9). The output vector is used to update the outgoing messages of the generalized CNs. Note that, if $r \neq 1$ or $E = 0$, then the outgoing messages are binary ($\pm\infty$).

3) Backward update: The hidden VNs forward the messages from the generalized CNs and the standard CNs are updated according to (3). If $r \neq 1$ or $E = 0$, then the messages from generalized CN are $\pm\infty$ and the message received by a given VN equals the LLR from the other connected VN, but perhaps with a flipped sign. In this case, the result matches the aggregation function in the original RPA algorithm [4, Alg. 4].

4) VN update: If $E = 0$, then the outgoing VN messages for $v \in \mathcal{V}$ are computed like the original RPA algorithm with

$$\lambda^{(t)}_{v\to c} = \frac{1}{2^m - 1} \sum_{c' \in \partial v \setminus c} \hat{\lambda}^{(t)}_{c'\to v}, \tag{4}$$

where the If $E = 1$, then (6) is used instead. Then, after T_max iterations, the hard-decision outputs are formed by computing

$$\hat{\ell}_v = \frac{1}{2^m - 1} \sum_{c \in \partial v} \hat{\lambda}^{(t)}_{c\to v}, \tag{5}$$

for all $v \in \mathcal{V}$, and then mapping the output to $\pm\infty$.

The VN update rule (4) can be seen as an intrinsic version of a *weighted* BP update rule where the channel message has zero weight and the 2^m-1 incoming messages are simply averaged. Indeed, a known problem with the standard BP decoder is that it only tends to work well for sparse factor graphs with tree-like neighborhoods. However, redundant factor graphs generally have many cycles. A standard technique in this case is to add weights to reduce the overconfidence induced by correlated messages [22]. For example, weighted BP decoding using a single weighting factor for scaling incoming VN message gives good performance when decoding RM codes based on overcomplete parity-check matrices, even though the factor graph has many cycles [7]. More generally, it can be beneficial to introduce and optimize general weighting factors for each edge in the factor graph [23], [24].

Algorithm 2: RXA and CXA

```
Input: input LLRs $\boldsymbol{\ell}$, RM parameters $r, m$, collapsed $C \in \{0,1\}$
Output: output LLRs $\hat{\boldsymbol{\ell}}$
1  if $r = m-2$ then
2      $\hat{\boldsymbol{\ell}}$ = decode $\boldsymbol{\ell}$ using extended Hamming FHT decoder [18]
3  else
4      construct the RXA or, if $C = 1$, the CXA factor graph for
       RM$(r,m)$ with node sets $\mathcal{V}, \mathcal{C}_{\mathrm{g}}$
5      initialize $\lambda^{(0)}_{v\to c} = \ell_v \ \forall v \in \mathcal{V}$
6      for $t = 1, \dots, T_{\max}$ do
           // subcode decoding, step (1) in Fig. 2(c)
7          if $C = 0$ then
8              $(\hat{\lambda}^{(t)}_{c\to v})_{v\in\partial c} = \mathtt{RXA}\left((\lambda^{(t)}_{v\to c})_{v\in\partial c}, r, m-1\right) \ \forall c \in \mathcal{C}_{\mathrm{g}}$
9          else // jump to line 2 and apply FHT decoding
10             $(\hat{\lambda}^{(t)}_{c\to v})_{v\in\partial c} = \mathtt{RXA}\left((\lambda^{(t)}_{v\to c})_{v\in\partial c}, r, r+2\right) \ \forall c \in \mathcal{C}_{\mathrm{g}}$
           // VN update, step (2)
11         update $\lambda^{(t)}_{v\to c}$ via (6) $\forall v \in \mathcal{V}$
12     compute $\hat{\ell}_v$ via (5) $\forall v \in \mathcal{V}$
13 return $\hat{\boldsymbol{\ell}}$;
```

B. Recursive Puncturing–Aggregation

The factor graph for the $(\mathbf{u}, \mathbf{u}+\mathbf{v})$ construction in Fig. 2(a) has two RM$(r, m-1)$ constraints based on puncturing the two codeword halves. Similar to RPA, additional RM$(r, m-1)$ constraints can be obtained by exploiting the code's automorphism group. Fig. 2(c) illustrates the resulting factor graph that is used for RXA decoding. A precise definition of the factor graph connectivity[3] is given in [20]. Since there are no projections, the factor graph consists only of standard VNs (contained in $\mathcal{V}$) and generalized CNs (contained in $\mathcal{C}_{\mathrm{g}}$). There are $|\mathcal{C}_{\mathrm{g}}| = 2(2^m - 1)$ subcode constraints in total—twice as many as for RPA.

The RXA algorithm is shown in Algorithm 2. The base codes are decoded using an extended Hamming FHT decoder [18] that produces bit-wise posterior LLRs. Thus, RXA works entirely with extrinsic (i.e., $E = 1$) message passing. Otherwise, the algorithm follows a simple "flooding" scheduling, alternating between recursive decoding (line 8) and a VN update (line 11). For RXA, we observed that the update rule in (4) does not give good results and we instead use the weighted extrinsic VN update rule given by

$$\lambda^{(t)}_{v\to c} = \ell_v + w_{r,m} \sum_{c' \in \partial v \setminus c} \hat{\lambda}^{(t)}_{c'\to v}, \tag{6}$$

where $w_{r,m}$ can be optimized separately for each algorithm (i.e., RPA, RXA, CPA, and CXA) and each channel condition.

IV. Non-Recursive (Collapsed) Algorithms

To decode RM(r, m) using RPA, the number of RM$(1, m-r+1)$ base codes is $\prod_{d=0}^{r-2}(2^{m-d} - 1)$. On the other hand, any RM$(1, m-r+1)$ subcode can be obtained by projecting RM(r, m) via an $(r-1)$-dimensional subspace in $\mathbb{F}_2^m$ [4, Lem. 1]. The number of distinct $(r-1)$-dimensional subspaces in $\mathbb{F}_2^m$ is given by the Gaussian binomial coefficient $\binom{m}{r-1}_2 \triangleq \prod_{l=0}^{d-1} \frac{2^{m-l}-1}{2^{d-l}-1}$, where $r-1 \le m$. When $r-1 \ge 2$, the ratio between $\prod_{d=0}^{r-2}(2^{m-d} - 1)$ and $\binom{m}{r-1}_2$ is $\prod_{d=1}^{r-1}(2^d - 1) > 1$. This means that RPA with more than one stage of recursion reuses the same RM$(1, m-r+1)$ base code multiple times. A similar argument can be made for RXA decoding.

To avoid this reuse of base codes, we investigate collapsed algorithms based on factor graphs whose generalized CNs are only for the base codes RM$(1, m-r+1)$ or RM$(r, r+2)$. We note that the idea of using other RM subcodes was mentioned briefly in [6, Sec. VI] but no results were provided. The journal version [5] (which appeared after this paper was submitted) does present results for a simplified decoder using this idea.

A. Collapsed Projection–Aggregation

The factor graph for CPA is shown in Fig. 2(d). There are $B = \binom{m}{r-1}_2$ different RM$(1, m-r+1)$ subcodes and each VN in $\mathcal{V}$ has degree B. The CNs correspond to projections involving 2^{r-1} code bits, i.e., their degree is $2^{r-1}+1$. The CPA algorithm is obtained by calling Algorithm 1 with the $C = 1$ flag. The corresponding steps mimic those of RPA except that the recursion in line 9 is replaced with the decoding of the RM$(1, m-r+1)$ subcodes (line 11) and the combining weight in (6) is $w_{r,m} = \alpha/B$ for some optimized $\alpha \in (0, 1]$.

B. Collapsed Puncturing–Aggregation

Similar to CPA, the CXA algorithm collapses the recursion in RXA and directly decodes RM$(r, r+2)$ subcodes. The factor graph is a bipartite graph similar to Fig. 2(c). The VN degree is $B = \binom{m}{m-r-2}_2$ and there are $2^{m-r-2}B$ different subcodes based on the distinct $(m-r-2)$-dimensional subspaces in $\mathbb{F}_2^m$. The RXA algorithm uses $2^{m-r-2}\prod_{d=0}^{m-r-2}(2^{m-d} - 1)$ base codes and, in comparison, the number of base codes used by CXA is divided by $\prod_{d=1}^{m-r-2}(2^d - 1)$. The CXA algorithm is obtained by calling Algorithm 2 with the $C = 1$ flag. It follows the same steps as RXA except that the recursion in line 7 is replaced with the extended Hamming FHT decoder which performs the base decoding. Also, the combining weight in (6) is chosen to be $w_{r,m} = \alpha/B$ for some optimized $\alpha \in (0, 1]$.

C. Connection to Minimum-Weight Parity Checks

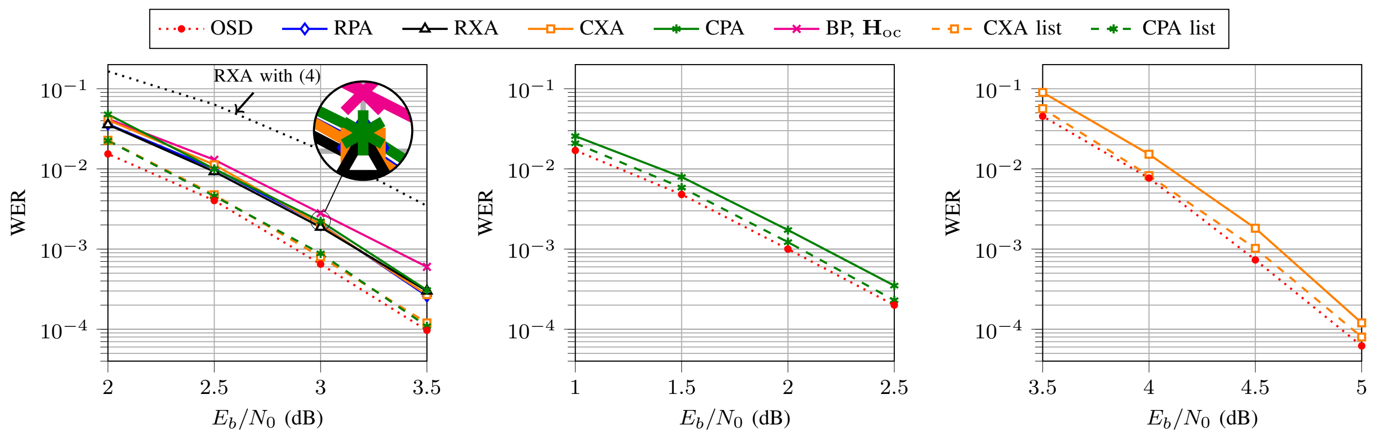
In [7] (see also [9]), weighted BP decoding of RM codes is considered based on parity-check matrices that contain all minimum-weight dual codewords as rows. This can be seen as CXA decoding with one additional stage of puncturing. In particular, puncturing RM$(r, r+2)$ leads to RM$(r, r+1)$ which are single parity-check codes, see Fig. 1. The number of subcodes for CXA assuming an additional puncturing stage is $2^{m-r-1}\binom{m}{m-r-1}$ which is exactly the number of minimum weight codewords of RM$(m-r-1, m)$ [17, p. 381]. Thus, there is a common thread between these new methods and prior work on overcomplete decoding.

V. Simulation Results

A. Performance

The described decoding algorithms are tested on some RM codes with lengths $N \in \{128, 256\}$ over the AWGN channel. The iteration number is set to $T_{\max} = 15$ in all

[3]RXA exploits the same $2^m - 1$ distinct one-dimensional subspaces in $\mathbb{F}_2^m$ as RPA, but each subspace is used to partition $\mathcal{V}$ into two ordered sets.



(a) RM$(3,7)$, $[128,64]$, $R = 0.5$ (b) RM$(2,8)$, $[256,37]$, $R = 0.145$ (c) RM$(5,8)$, $[256,219]$, $R = 0.856$

Fig. 3: Simulation results

cases. A convergence-based stopping criterion according to $\|\hat{\boldsymbol{\ell}}_{\text{old}} - \hat{\boldsymbol{\ell}}_{\text{new}}\| < 0.01\|\hat{\boldsymbol{\ell}}_{\text{old}}\|$, where $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{N} x_i^2}$, was implemented to reduce simulation time, similar to [4]. Ordered statistics decoding (OSD) is used as a benchmark, whose performance is close to ML [25].

The self-dual $\mathrm{RM}(3,7)$ code is used to compare the performance between the different projection and puncturing approaches. The results in Fig. 3(a) show that RPA, RXA, CPA, and CXA all perform similarly and the gap to OSD in all cases is around 0.3 dB at a word error rate (WER) of 10^{-3}. The performance degradation of RXA when using the RPA-style VN update (4) instead of (6) is also shown. As a reference, we compare to weighted BP decoding assuming an overcomplete parity-check matrix $\mathbf{H}_{\text{oc}}$ that contains all $94\,488$ minimum-weight dual codewords as rows, where the weighting factor for incoming VN messages is 0.05 [7, Fig. 1(b)]. Weighted BP has higher WER compared to the considered algorithms which highlights the advantage of using more powerful base codes (i.e., extended Hamming or augmented Hadamard codes) compared to single parity-checks. Following [4], we also consider list decoding [26] in combination with the proposed algorithms. For a list size of 16, the resulting performance of both CPA and CXA approaches the OSD curve within < 0.1 dB at a WER of 10^{-3}.

Lastly, we consider the codes $\mathrm{RM}(2,8)$ and $\mathrm{RM}(5,8)$. For $\mathrm{RM}(2,8)$, the results in Fig 3(b) show that CPA performs within 0.2 dB of OSD at a WER of 10^{-3}, and this gap becomes almost negligible once the list decoder is used. Note that CPA and RPA are equivalent for second-order RM codes. For CXA on $\mathrm{RM}(5,8)$, Fig 3(c) shows a similar 0.2 dB gap from OSD, and a reduced gap of < 0.1 dB once the list decoder is used. Note that CXA is equivalent to RXA for this code.

B. Complexity

In terms of computational complexity, the same number of projection/puncturing stages are required for self-dual codes. In general, however, approaches based on projections have fewer subcodes than those based on puncturing. They also work well with hard-decision decoding of base codes, whereas soft-decisions are required for RXA and CXA. On the other hand, projections require evaluating the standard CN update equation, which is not required for puncturing.

Comparing the recursive and collapsed approaches, both CPA and CXA reduce the number of base codes for RM$(3,7)$ by a factor of 3. Moreover, the number of base decodings is limited to at most T_{max} for collapsed algorithms, whereas the recursion entail a potentially much higher number of base decodings: up to T_{max}^2 in the case of RM$(3,7)$.

While all of these algorithms are interesting from a theoretical perspective, their computational complexity is actually quite high. For example, the CPA algorithm for RM$(3,7)$ decodes 2667 different RM$(1,5)$ codes per iteration and has a complexity of roughly $2667{\cdot}32{\cdot}(5{+}4) \approx 768K$ operations per iteration. The simplified RPA approach in [5] reduces this but it is unclear if these methods can match list-based approaches (e.g., [8], [10], [16]) in performance versus complexity.

For example, it is known that SCL decoding with a list size of 16 already achieves very good performance for RM$(3,7)$ at high SNR [8]. Also, the simulation code for the method in [16] was recently posted to GitHub (see https://github.com/kshabunov/ecclab). We tested this code and found that a list size of 32 was sufficient to achieve near-ML performance for RM$(3,7)$. The decoding speed, with list size 32, is also much faster than our simulation code for RPA, RXA, CPA, and CXA without a list. Of course, these algorithms are quite new and it remains to be seen if program optimization and algorithm development can make this approach competitive in practice.

VI. Conclusions

This paper connects the RPA decoding of RM codes to message-passing and BP decoding on a redundant factor graph. Based on this, the RXA algorithm is introduced to decode high-rate RM codes. It is analagous to RPA decoding but it is based on puncturing up to RM$(m-2,m)$ codes rather than projection down to RM$(1,m)$ codes. To reduce complexity, we also propose collapsing the recursions and projecting (or puncturing) directly to the base codes. More work is needed to fully understand the complexity trade-off between these algorithms and previous approaches based on lists.

References

[1] D. Muller, “Application of Boolean algebra to switching circuit design and to error detection,” *IRE Tran. on Electronic Computers*, vol. EC-3, no. 3, pp. 6–12, Sept 1954.

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

[3] S. Kudekar, S. Kumar, M. Mondelli, H. D. Pfister, E. Şaşoğlu, and R. Urbanke, “Reed-Muller codes achieve capacity on erasure channels,” *IEEE Trans. Inf. Theory*, vol. 63, no. 7, pp. 4298–4316, 2017.

[4] M. Ye and E. Abbe, “Recursive projection-aggregation decoding of Reed-Muller codes,” in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Paris, France, 2019.

[5] ——, “Recursive projection-aggregation decoding of Reed-Muller codes,” *IEEE Trans. Inf. Theory*, Mar. 2020.

[6] ——, “Recursive projection-aggregation decoding of Reed-Muller codes,” *arxiv:1902.0147v2*, 2019.

[7] E. Santi, C. Häger, and H. D. Pfister, “Decoding Reed-Muller codes using minimum-weight parity checks,” in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Vail, CO, 2018.

[8] S. A. Hashemi, N. Doan, M. Mondelli, and W. J. Gross, “Decoding Reed-Muller and polar codes by successive factor graph permutations,” in *Proc. IEEE Int. Symp. Turbo Codes and Iterative Information Processing (ISTC)*, Hong Kong, Hong Kong, Dec. 2018.

[9] M. Bossert and F. Hergert, “Hard- and soft-decision decoding beyond the half minimum distance—An algorithm for linear codes,” *IEEE Trans. Inf. Theory*, vol. 32, no. 5, pp. 709–714, Sept. 1986.

[10] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[11] V. M. Sidel’nikov and A. S. Pershakov, “Decoding of Reed-Muller codes with a large number of errors,” *Problemy peradichi informatsii*, vol. 28, no. 3, pp. 80–94, 1992.

[12] P. Loidreau and B. Sakkour, “Modified version of Sidel’nikov-Pershakov decoding algorithm for binary second order Reed-Muller codes,” in *Proc. Int. Workshop Algebraic and Combinatorial Coding Theory (ACCT)*, Kranevo, Bulgaria, 2004.

[13] B. Sakkour, “Decoding of second order Reed–Muller codes with a large number of errors,” in *Proc. IEEE Information Theory Workshop (ITW)*, Rotorua, New Zealand, 2005.

[14] I. Dumer, “Recursive decoding and its performance for low-rate Reed–Muller codes,” *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 811–823, May 2004.

[15] ——, “Soft-decision decoding of Reed-Muller codes: a simplified algorithm,” *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, March 2006.

[16] I. Dumer and K. Shabunov, “Soft-decision decoding of Reed-Muller codes: Recursive lists,” *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, March 2006.

[17] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Elsevier, 1977.

[18] A. Ashikhmin, G. Kramer, and S. ten Brink, “Extrinsic information transfer functions: model and erasure channel properties,” *IEEE Trans. Inf. Theory*, vol. 50, no. 11, pp. 2657–2673, Nov. 2004.

[19] F. Kschischang, B. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[20] M. Lian, C. Häger, and H. D. Pfister, “Decoding Reed-Muller codes using redundant code constraints,” *to appear on arXiv*, 2020.

[21] A. Ashikhmin and S. Litsyn, “Simple MAP decoding of first-order Reed–Muller and Hamming codes,” *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1812–1818, 2004.

[22] R. Palanki, M. P. Fossorier, and J. S. Yedidia, “Iterative decoding of multiple-step majority logic decodable codes,” *IEEE Trans. Commun.*, vol. 55, no. 6, pp. 1099–1102, June 2007.

[23] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be’ery, “Deep learning methods for improved decoding of linear codes,” *IEEE J. Sel. Topics Signal Proc.*, vol. 12, no. 1, pp. 119–131, Feb. 2018.

[24] M. Lian, F. Carpi, C. Häger, and H. D. Pfister, “Learned belief-propagation decoding with simple scaling and SNR adaptation,” in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Paris, France, 2019.

[25] M. P. C. Fossorier and S. Lin, “Soft-decision decoding of linear block codes based on ordered statistics,” *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1379–1396, Sept. 1995.

[26] D. Chase, “A class of algorithms for decoding block codes with channel measurement information,” *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.

[27] I. Dumer and K. Shabunov, “Recursive list decoding for Reed-Muller codes and their subcodes,” in *Information, Coding and Mathematics*. Springer US, 2002, pp. 279–298.