RZ 3907 (# ZUR1604-007) 11/10/2016
Electrical Engineering 5 pages

Research Report

Improving the Error-Floor Performance of Binary Half-Product Codes

Thomas Mittelholzer, Thomas Parnell, Nikolaos Papandreou and Haralampos Pozidis

IBM Research – Zurich
8803 Rüschlikon
Switzerland

The final version of this paper has been published by IEEE: T. Mittelholzer, T. Parnell, N. Papandreou and H. Pozidis, "Improving the Error-Floor Performance of Binary Half-Product Codes," in *Proc*. International Symposium on Information Theory and Its Applications (ISITA), 295-299

Improving the Error-Floor Performance of Binary Half-Product Codes

Thomas Mittelholzer, Thomas Parnell, Nikolaos Papandreou and Haralampos Pozidis
IBM Research – Zurich
8803 Rüschlikon, Switzerland
Email: {tmi,tpa,npo,hap}@zurich.ibm.com

***Abstract*—Recently, a new type of product-like codes, known as half-product codes, has been studied for OTN applications. Binary half-product codes show excellent bit-error-rate performance under iterative hard-decision decoding in the waterfall region. To further improve the performance in the error floor region, a novel post-processor algorithm is proposed that relies on conditional bit flipping. For a class of important practical half-product codes, the error floor after post-processing is characterized.**

I. Introduction

With the shift towards longer codes in applications such as optical transport networks (OTN), product codes have been re-considered and new product-like codes proposed [1]–[8]. A key feature of these codes is that they have relatively simple hard-decision based decoders and that their bit-error-rate (BER) performance under iterative decoding can be analyzed analytically for low BERs [5].

In this paper, we will study binary half-product codes (HPC), which are derived from symmetry-invariant subcodes of $n\times n$ product codes. For comparable parameters, HPCs often outperform product codes both in the waterfall and the error-floor region.

To further lower the error floor, erasure-based post-processors (PP) were proposed in [6] and [7]. In this paper, we consider a different type of PP that applies to binary product-like codes. The novel PP relies on conditional bit flipping; it outperforms the erasure-based PP in many cases. We also provide an analytic approximation of the error-floor performance of the PP, which is crucial for applications.

In Section II, we review half-product codes and their associated graphical models as defined in [3], [5]. In Section III, analytical performance approximations for the waterfall and error-floor regions are given, following the approach in [5] and [8]. In Section IV, the novel bit-flipping-based PP is described; by using the graphical model of HPCs, the minimum stopping sets of the PP are characterized and enumerated. We validate the analytical performance approximations by simulations in Section V, and draw our conclusions in Section VI.

II. Half-Product Codes and Complete Graphs

Half-product codes are subcodes of two-dimensional product codes that are invariant under transposition [3], [5]. More formally, for a given (square) product code $\mathcal{C}_P$, based on a component code C, the corresponding *half-product code* $\mathcal{C}_H$ is defined by the set of upper triangular matrices in the set of anti-symmetric matrices (see [5]):

$$\tilde{\mathcal{C}}_H = \{X - X^T : X \in \mathcal{C}_P\}. \tag{1}$$

Here, $X = [x_{ij}] \in \mathcal{C}_P$ is a $n\times n$ square array whose rows and columns are codewords of C, and X^T denotes the transpose of X.

Given a (n, k, d) component code, the HPC has length $N = n(n-1)/2$ and dimension $K = k(k-1)/2$ [5]. The minimum distance formula $d(d-1)/2$ as originally given in [5] is too conservative; namely, HPCs have larger minimum distances as specified by the bound $D_H \geq 3d^2/4$, which has recently been proved in [9].

The encoding of HPCs is similar to the encoding of product codes: One forms a (anti)-symmetrical $k \times k$ array with zero diagonal and encodes the corresponding product code.

The graphical model for the HPC is the complete graph with n vertices, where n is the length of the component code [5]. The n vertices correspond to check nodes, which impose the constraints of the component code. The $n(n-1)/2$ edges of the complete graph correspond to the $n(n-1)/2$ codeword components in the upper triangular matrices.

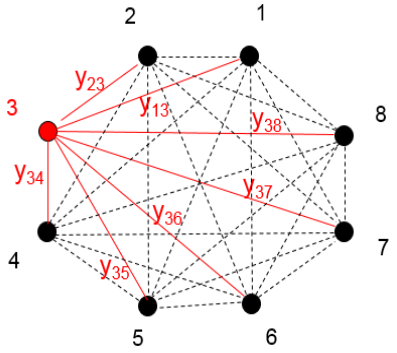

Fig. 1: Graphical model for HPC in Example 1.

Example 1: The codewords of the binary ($N = 28$, $K = 6$, $D_H = 12$) HPC based on the ($n = 8$, $k = 4$, $d = 4$) extended

Hamming code correspond to symmetric 8×8 matrices

$$Y = \begin{bmatrix} 0 & y_{12} & y_{13} & y_{14} & y_{15} & y_{16} & y_{17} & y_{18} \\ y_{12} & 0 & y_{23} & y_{24} & y_{25} & y_{26} & y_{27} & y_{28} \\ y_{13} & y_{23} & 0 & y_{34} & y_{35} & y_{36} & y_{37} & y_{38} \\ y_{14} & y_{24} & y_{34} & 0 & y_{45} & y_{46} & y_{47} & y_{48} \\ y_{15} & y_{25} & y_{35} & y_{45} & 0 & y_{56} & y_{57} & y_{58} \\ y_{16} & y_{26} & y_{36} & y_{46} & y_{56} & 0 & y_{67} & y_{68} \\ y_{17} & y_{27} & y_{37} & y_{47} & y_{57} & y_{67} & 0 & y_{78} \\ y_{18} & y_{28} & y_{38} & y_{48} & y_{58} & y_{68} & y_{78} & 0 \end{bmatrix}$$

where all rows and columns are codewords of the length-8 extended Hamming code. The upper triangular part of Y determines the HPC codeword. When one restricts oneself to the upper triangular part, one can identify the rows of Y as folded structures with a 90-degree folding at the zero diagonal. For example, the length-8 component codeword in the 3rd row is represented by the folded structure high-lighted in `type-font`.

The graphical model for this HPC is the complete graph on 8 vertices as illustrated in Fig. 1. Vertices represent folded codewords of the component code, and edges represent the codeword bits. For instance, the edges leaving vertex 3 correspond to the 7 off-diagonal symbols $\mathtt{y}_{13}$, $\mathtt{y}_{23}$, $\mathtt{y}_{34}$, $\mathtt{y}_{35}$, $\mathtt{y}_{36}$, $\mathtt{y}_{37}$, $\mathtt{y}_{38}$ of the folded structure, which represents row 3. Two folded structures i and j have exactly one codeword component in common, viz., y_{ij}, and hence the vertices i and j are connected by one edge with edge label y_{ij}.

III. Performance of Half-Product Codes

Product codes and half-product codes can be iteratively decoded using the associated graphical structures, namely, bipartite graphs and complete graphs, respectively. Hard-decision iterative decoding of product codes is performed based on the graph by applying bounded-distance decoding to all row codewords and then applying bounded-distance decoding to all column codewords. Each time a codeword is successfully decoded, the edges leaving the appropriate node are corrected. The process iterates until decoding is complete, i.e., until either all syndromes are zero or the decoder makes no further progress. In the first case, the decoder output is a codeword, and decoding is successful or produces a miscorrection. In the second case, the decoder fails to decode. In a similar way, iterative decoding of HPCs is based on bounded-distance decoding of the folded codewords of the underlying component code.

In the next two subsections, we review the performance limits of iterative decoding, which is based on iterative decoding thresholds and approximate BER performance analysis in the waterfall and the error-floor region.

A. Iterative Decoding Threshold

For component codes of increasing lengths but with a fixed error-correction capability t, iterative decoding has been analyzed using various techniques [1]. If miscorrections are neglected, these approaches have matching results. In [5], the analysis has been extended to HPCs using the threshold behavior of the appearance of k-cores in a random graph [10]. We briefly review this result.

Starting from the complete graph of a HPC with n vertices and $n(n-1)/2$ edges, one obtains an *error graph* by transmitting a codeword through a binary symmetric channel (BSC) with crossover probability p. The channel will flip each edge label (codeword component) with probability p. The subgraph of the flipped edges (components) is known as error graph $\Gamma_e = (V_e, E_e)$. On average, the error graph has n vertices and $pn(n-1)/2$ edges. For the complete graph with $n(n-1)/2$ edges, each node of the error graph has $p(n-1)$ edges on average and the edge distribution at each node is binomial. For large n, it is well approximated by a Poisson distribution with parameter $\lambda = p(n-1)$.

A k-core is a subgraph of the (error) graph with an edge degree of at least k for all its vertices. Consider an HPC based on a t-error correcting component code. The decoder will fail if and only if the error graph contains a $(t+1)$-core [1]. For a random graph with v nodes and e edges, asymptotically there exists a k-core for $k > 2$ with high probability when $e > vc_k/2$, where the threshold c_k is determined by a truncated Poisson distribution [10]; in particular, $c_3 = 3.35$, $c_4 = 5.14$, $c_5 = 6.80$, $c_6 = 8.37$. This is the basis for the definition of the *iterative decoding threshold* as

$$p_c = vc_{t+1}/2e. \tag{2}$$

For large code lengths, iterative decoding succeeds with high probability if and only if the BSC has crossover probability $p < p_c$.

B. Approximate Analytical Performance Analysis

A length-N codeword that was sent over the BSC with crossover probability p has an error distribution $f_{\text{obs},p}(s)$ of the observed errors, which is binomial with mean Np and variance $Np(1-p)$. Here s denotes the actual observed error rate within a codeword. For large N, this is well approximated by the normal distribution with the same mean and variance. Following the argument in Section 4.1.1 of [4], we write the frame error rate as

$$\text{FER}(p) = \int_0^1 f_{\text{obs},p}(s)\text{Pr[Frame error }|s]ds. \tag{3}$$

The threshold property of iterative decoding of long (sub)-product codes implies that Pr[Frame error $|s$] is well approximated by a step function that jumps from 0 to 1 at the iterative decoding threshold p_c, which leads to

$$\text{FER}(p) \approx \int_{p_c}^1 f_{\text{obs},p}(s)ds = \frac{1}{2}\text{erfc}\left(\frac{(p_c-p)\sqrt{N}}{\sqrt{2p(1-p)}}\right). \tag{4}$$

Here erfc denotes the complementary error function. When decoding fails, we assume that the number of bit errors is $N\max\{p_c, p\}$, and thus the output BER is approximated by

$$\text{BER}(p) \approx \frac{1}{2}\max\{p_c, p\}\text{erfc}\left(\frac{(p_c-p)\sqrt{N}}{\sqrt{2p(1-p)}}\right). \tag{5}$$

The formula for BER(p) applies to the waterfall region of the BER curve. To obtain the performance in the error-floor region, we study error patterns that make the decoder fail, which were termed *stalling patterns* in [4] and *stopping sets* in [9]. In terms of graphical models, a stopping set is an error graph Γ_S on which the iterative decoder fails to make progress.

For a product code of length $N = n^2$, which is based on a t-error-correcting component code (for rows and columns), the stopping sets of minimum weight are easy to characterize: the minimum weight patterns have weight $w = (t+1)^2$, and consist of $t+1$ rows with $t+1$ errors at the same locations, i.e., they are square structures with $(t+1)\times(t+1)$ errors. The corresponding error graph Γ_S is a fully connected bipartite graph in which each partition has exactly $t+1$ vertices. The number of these stopping patterns is given by

$$\mu = \left(\begin{array}{c} n \\ t+1 \end{array} \right) \left(\begin{array}{c} n \\ t+1 \end{array} \right).$$

The error-floor performance is approximated as (see [2], [5])

$$\text{BER}_{\text{floor}} \approx \mu p^w w/N. \tag{6}$$

In [3], the stopping sets for HPCs with $t = 3$ have been analyzed. The minimum-weight stopping sets have weight $w_H = (t+2)(t+1)/2 = 10$ and their multiplicity is

$$\mu_H = \left(\begin{array}{c} n \\ t+2 \end{array} \right).$$

The multiplicity μ equals the number of complete graphs on $t+2$ vertices within the complete graph on n vertices. Clearly, this holds for any t. With these parameters, one can readily generalize the error-floor approximation (6) for the HPC case.

IV. Lowering the Error Floor

A. Post-Processor for HPCs

One can reduce the error floor by using a post-processor (PP) in conjunction with the iterative decoder that detects and corrects stopping sets of small weight. For binary HPCs, the smallest stopping sets $\Gamma_S = (V_S, E_S)$ are complete graphs on $t+2$ vertices V_S. When the iterative decoder gets stuck on a minimum stopping set, Γ_S can be determined from those folded codewords of the component code that have nonzero syndromes as they correspond to the vertices V_S. As the edges E_S of the complete graph are determined by the vertices V_S, one can correct all $(t+2)(t+1)/2$ bit errors by flipping all edge labels of Γ_S.

The PP can be extended to correct more than just the minimal error patterns of a binary HPC. Suppose the iterative decoder fails to decode and halts on an unknown stopping (stalling) set $\Gamma_S = (V_S, E_S)$. As above, the vertex set V_S can be determined from the set of folded codewords with nonzero syndromes. However, if the stopping set is not minimal, only limited information is available on the edge set E_S. We know that the edge degree in each vertex is at least $t+1$ as the iterative decoder would have corrected all folded codewords (vertices) with t or fewer errors. Thus, the cardinality $|E_S|$ of the edge set is in the range

$$\left\lceil \frac{(t+1)|V_S|}{2} \right\rceil \le |E_S| \le \frac{|V_S|(|V_S|-1)}{2}, \tag{7}$$

where $\lceil x \rceil$ denotes the smallest integer not smaller than x.

To specify the operation of the PP, we introduce the notion of the *ambient error graph* $\Gamma_A = (V_A, E_A)$ of the stopping set Γ_S, which is defined to be the complete graph with vertex set $V_A = V_S$. The vertex set V_S is known from the set of folded component codewords with nonzero syndromes and Γ_A is the largest possible error graph on V_S.

Single-step post-processor:
Consider an HPC that is based on a t-error-correcting component code and suppose the iterative decoder gets stuck on a stopping set $\Gamma_S = (V_S, E_S)$. Then,

- if $|V_S| \le 2t+2$,
  1) the PP reverses all bits corresponding to the edges of the ambient error graph Γ_A, and
  2) the PP applies 1 additional round of iterative decoding;
- otherwise, the PP leaves the output of the iterative decoder unchanged and declares a failure.

Proposition 1: Every stopping set of the HPC with at most $2t+2$ vertices is correctly decoded by the PP.

To prove this proposition, we consider the complement $\overline{\Gamma}_S$ of the error graph, which is a graph on the same vertices as Γ_S such that two distinct vertices of $\overline{\Gamma}_S$ are adjacent if and only if they are not adjacent in Γ_S. The bit reversal of the PP transforms the stopping set Γ_S into its complement $\overline{\Gamma}_S$.

By (7), a stopping set on $v = |V_s|$ vertices has at least $\lceil (t+1)v/2 \rceil$ edges and its complement $\overline{\Gamma}_S$ has at most $(v-1)v/2 - \lceil (t+1)v/2 \rceil$ edges. This number is upper bounded by $tv/2$ because of the PP condition $v \le 2t+2$. Thus, the complementary stopping set $\overline{\Gamma}_S$ has fewer edges than the lower bound (7) and, therefore, it is correctable by iterative decoding. In fact, each vertex of $\overline{\Gamma}_S$ has edge degree at most t because each vertex in Γ_S has at least degree $t+1$, and thus a single iteration step is sufficient to decode.

Multi-step post-processor:
Using a similar argument based on the stopping set and its complement, one can show that for even t, the PP can be extended to correct all stopping sets with $|V_S| \le 2t+3$. The bit-reversal step 1 remains unchanged; however, in step 2 more than one round of iterative decoding may be necessary.

Example 1 (cont.ed) Suppose a codeword of the HPC that is based on the extended $(8,4,4)$ Hamming code is corrupted by noise, which results in 4 errors at locations y_{34}, y_{38}, y_{47}, y_{78}. These 4 errors form a stopping set with $V_S = \{3,4,7,8\}$ and $E_S = \{(3,4),(3,8),(4,7),(7,8)\}$, which is illustrated in Fig. 2. In Fig. 2 and Fig. 3a the ambient error graph Γ_A is shown as the complete graph on the vertices V_S, which are connected by bold dashed and solid lines. When the iterative

TABLE I: Cardinality of automorphism groups of the 16 non-isomorphic 4-regular graphs on 9 vertices.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\lvert\mathrm{Aut}(\Gamma_i)\rvert$	16	32	4	8	12	18	2	2	4	2	8	8	16	72	12	18

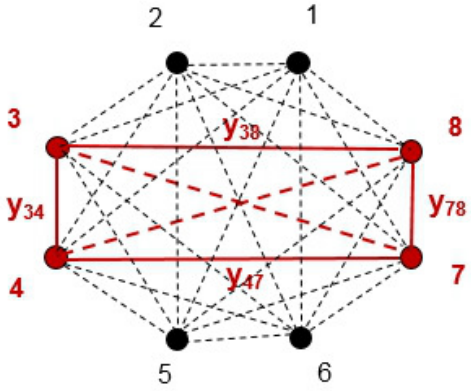

Fig. 2: Stopping set with 4 errors of HPC in Example 1.

(a) Stopping set Γ_S (b) Complementary stopping set $\overline{\Gamma}_S$ (c) No errors

Fig. 3: Illustration of post-processor operations

decoder gets stuck on the (unknown) stopping set, the PP can determine the vertex set V_S from the 4 nonzero syndromes of the folded codewords. As $|V_S| \leq 2t+2$ holds, the PP applies bit reversal to the ambient error graph Γ_A. This transforms the stopping set into its complement $\overline{\Gamma}_S$, which is shown in Fig. 3b. One additional round of iterative decoding can then remove the errors, and this results in the error graph in Fig. 3c without errors.

The concept of ambient error graph and complement of a stopping set can be applied to other types of binary product-like codes that are iteratively decoded by a bounded-distance decoder of the underlying component code(s). In particular, one can formulate similar PP decoding rules for product and quarter-product codes based on conditional bit flipping.

B. Error Floor of the Post-Processor

We will determine the error floor of the PP by specifying the weight and multiplicity of the smallest stopping sets that cannot be corrected by the PP.

Theorem 1: Let $\Gamma_S = (V_S, E_S)$ be a minimal stopping set of the PP (and its extension for t even). Then, the number of vertices equals

$$|V_S| = \begin{cases} 2t+3 & \text{for } t \text{ odd} \\ 2t+4 & \text{for } t \text{ even,} \end{cases} \tag{8}$$

and the number of edges, i.e., the weight, is given as

$$|E_S| = \begin{cases} (2t+3)(t+1)/2 & \text{for } t \text{ odd} \\ (2t+4)(t+1)/2 & \text{for } t \text{ even} \end{cases}. \tag{9}$$

For a minimal stopping set, clearly, the cardinality $|V_S|$ is by 1 larger than the upper PP bounds $|V_S| \leq 2t+2$ and $|V_S| \leq 2t+3$ for odd and even t, respectively.

The cardinality $|E_S|$ is the number of edges in a regular graph on $|V_S|$ vertices and edge degree $t+1$.

To find the multiplicity of PP stopping sets of minimum weight $w = |E_S|$, we count the number of different $(t+1)$-regular subgraphs on $v = |V_S|$ vertices within the complete graph on n vertices, where n is the length of the underlying component code of the HPC.

If we know the number of non-isomorphic $(t+1)$-regular graphs on v vertices, together with their automorphism groups, this graph counting problem is easily solved. However, for large t, it is considered to be hard because of the conjectured difficulty of the graph isomorphism problem. In the paper, we will treat the case $t = 3$, which is of most practical interest [3]. For small t, other cases can be treated similarly.

First, we consider the reduced problem of counting the number of different $(t+1)$-regular graphs with v vertices. For $t = 3$, there are 16 classes of non-isomorphic $(t+1)$-regular graphs Γ_i on $2t+3 = 9$ vertices, which can be found by a clever exhaustive search [11]. The automorphism groups $\mathrm{Aut}(\Gamma_i)$ and their orders are given in the associated web-page of [11]; for convenience, the orders $|\mathrm{Aut}(\Gamma_i)|$ are listed in Table I.

For a fixed vertex labelling of 9 vertices, each graph Γ_i corresponds to a distinct 'initial' stopping set. For each graph Γ_i, there are $9!/|\mathrm{Aut}(\Gamma_i)|$ different stopping sets; namely, for each permutation σ of the 9 vertices, there are $|\mathrm{Aut}(\Gamma_i)|$ equivalent permutations $\sigma\gamma$, $\gamma \in \mathrm{Aut}(\Gamma_i)$ that correspond to the same stopping set as σ. Thus, only $9!/|\mathrm{Aut}(\Gamma_i)|$ out of the 9! permutations give rise to distinct stopping sets. By adding across all 16 graphs, one gets $\mu_0 = \sum_i 9!/|\mathrm{Aut}(\Gamma_i)| = 1,024,380$.

To obtain the multiplicity of all weight-18 stopping sets, it is sufficient to note that the $v = 9$ stopping set vertices can be chosen freely out of the n vertices of the graphical model of the HPC. Combining with the multiplicity μ_0, the theorem below follows.

Theorem 2: For HPCs based on length-n 3-error correcting component codes, the multiplicity of the (minimal) weight-18 stopping sets of the PP is

$$\mu = 1,024,380 \begin{pmatrix} n \\ 9 \end{pmatrix}. \tag{10}$$

In comparison to the erasure-based PP in [7], the novel bit-flipping-based PP achieves lower error floors whenever the minimum distance $d = 2t+1$ of the binary component code is odd or when t is even. This follows from comparing the weight $|E_S|$ in Theorem 1 with the reported minimum weight

$S_{\min}^{\text{HPC}} = (d+1)(t+1)/2$ in [7]. Furthermore, in Theorem 2, we specify the multiplicity μ of the weight-18 PP stopping sets for $t = 3$, which allows one to get an approximation of the PP error floor using (6).

V. Performance of Selected Codes

To validate the analytical error-floor performance of the HPC and the PP, a short 'toy' HPC code was designed that is based on the 3-error-correcting binary $(31, 16, 7)$ BCH component code. Its performance curves are shown in Fig. 4. There is good agreement between the simulations of the pseudo-decoder (without miscorrections) and the analytical error-floor estimates (6). As the BCH code is very short, miscorrections during iterative decoding result in a noticeable gap between the performance of true and pseudo decoding. In

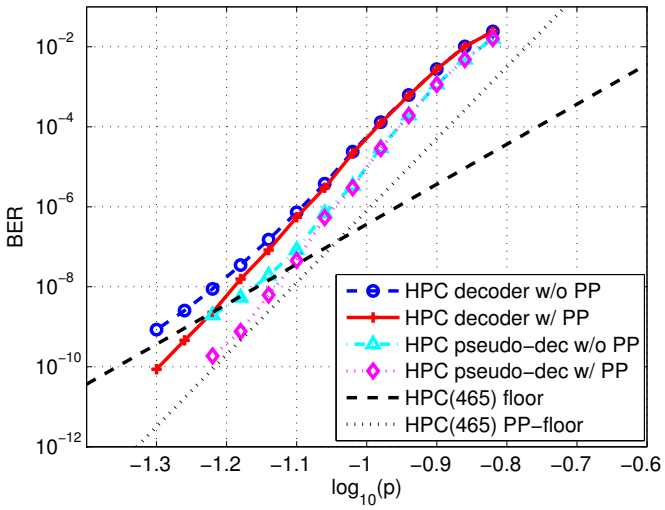

Fig. 4: Validation of approximate analytical error floor curves.

[3], an HPC was proposed for coding over optical transport networks (OTN) that is based on the binary $(1021, 990, 8)$ BCH component code. This HPC has a rate of 0.9402, which is slightly higher than that of the Reed–Solomon code in Appendix I.9 of the OTN standard G.975.1. In Fig. 5, we illustrate the lowering of the error floor of the HPC to a target BER of 10^{-19}. Note that the analytical BER performance as given by (5) is slightly too optimistic compared with the performance of the true decoder; however, it captures well the performance of the pseudo-decoder (without miscorrections). The HPC performs close to the capacity limit of the BSC. In both cases, the maximum number of decoding iterations was limited to 20.

VI. Conclusions

We have introduced a novel post-processor for lowering the error floor of iterative bounded-distance decoding of binary HPCs. The PP rule is based on bit-flipping using the graphical model of the HPC and its stopping sets. This PP outperforms the erasure-based PP in [7] in many cases, e.g., when the component code has odd minimum distance $d = 2t + 1$.

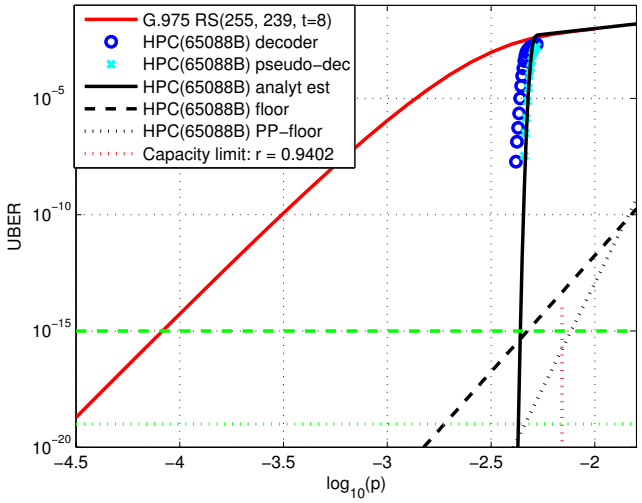

Fig. 5: Performance comparison of the reference G.975 code and a similar-rate HPC on a BSC with crossover probability p.

For HPCs with 3-error-correcting component codes, we have enumerated the minimal stopping sets of the PP and obtained an analytical approximation of the PP error-floor performance. The same technique applies to other small values of t.

The new concept of ambient error graph and complement of a stopping set can be applied to other types of product-like codes that are iteratively decoded by a bounded-distance decoder of the underlying component code(s).

References

[1] J. Justesen and T. Høholdt, "Analysis of Iterated Hard Decision Decoding of Product with Reed-Solomon Component Codes," *IEEE Proc. ITW 2007*, Lake Tahoe, CA, USA, Sept. 2007, pp. 174–177.

[2] J. Justesen, "Iterated Decoding of Modified Product Codes in Optical Networks," *IEEE Proc. Inform. Th. and Appl. Workshop ITA2009*, San Diego, CA, USA, Feb. 2009, pp. 160–163.

[3] J. Justesen, K.J. Larsen, and L.A. Pedersen,"Error Correcting Coding for OTN" *IEEE Commun. Mag.*, Vol. 48, No. 9, pp. 70–75, Sept. 2010.

[4] Benjamin P. Smith, Error-Correcting Codes for Fibre-Optic Communication Systems, Ph.D. Thesis, Univ. Toronto, 2011.

[5] J. Justesen, "Performance of Product Codes and Related Structures with Iterated Decoding," *IEEE Trans. Commun.*, Vol. 59, No. 2, pp. 407–415, Feb. 2011.

[6] Y.-Y. Jian, H.D. Pfister, K. Narayanan, R. Rao, and R. Mazahreh, "Iterative Hard-Decision Decoding of Braided BCH Codes for High-Speed Optical Communication," *IEEE Proc. GLOBECOM 2013*, pp. 2376–2381.

[7] S. Emmadi, K.R. Narayanan, and H.D. Pfister, "Half-Product Codes for Flash Memory," extended abstract and unpublished conference presentation at *Nonvolatile Memories Workshop (NVMW 2015)*, Mar, 2015, San Diego, CA, USA.

[8] T. Mittelholzer, T. Parnell, N. Papandreou, and H. Pozidis, "Symmetry-based Subproduct Codes," *Proc. IEEE Intl Symp. Inform. Th. (ISIT 2015)*, Hong Kong, June 2015, pp. 251 – 255.

[9] H.D. Pfister, S.K. Emmadi, and K. Narayanan, "Symmetric Product Codes," *IEEE Proc. ITA 2015*, San Diego, CA, USA, Feb. 2015, pp. 282 – 290.

[10] S. Janson, M. Luczak, "A Simple Solution to the k-Core Problem," *Random Structures Algorithms*, Vol. 30, pp. 50–62, 2007.

[11] M. Meringer, "Fast Generation of Regular Graphs and Construction of Cages," *J. Graph Theory 30*, pp. 137–146, 1999 (see also www.mathe2.uni-bayreuth.de/markus/reggraphs).