List-based Optimization of Proximal Decoding for LDPC Codes

Andreas Tsouchlos, Holger Jäkel, and Laurent Schmalen, *Fellow, IEEE*

***Abstract*—In this paper, the proximal decoding algorithm is considered within the context of *additive white Gaussian noise* (AWGN) channels. An analysis of the convergence behavior of the algorithm shows that proximal decoding inherently enters an oscillating behavior of the estimate after a certain number of iterations. Due to this oscillation, frame errors arising during decoding can often be attributed to only a few remaining wrongly decoded bit positions. In this letter, an improvement of the proximal decoding algorithm is proposed by establishing an additional step, in which these erroneous positions are attempted to be corrected. We suggest an empirical rule with which the components most likely needing correction can be determined. Using this insight and performing a subsequent "ML-in-the-list" decoding, a gain of up to 1 dB is achieved compared to conventional proximal decoding, depending on the decoder parameters and the code.**

***Index Terms*—Optimization-based decoding, Proximal decoding, ML-in-the-list.**

I. Introduction

Channel coding using binary linear codes is a way of enhancing the reliability of data by detecting and correcting any errors that may occur during its transmission or storage. One class of binary linear codes, *low-density parity-check* (LDPC) codes, has become especially popular due to its ability to reach arbitrarily small error probabilities at code rates up to the capacity of the channel [5], while retaining a structure that allows for very efficient decoding. While the established decoders for LDPC codes, such as belief propagation (BP) and the min-sum algorithm, offer good decoding performance, they are generally not optimal and exhibit an error floor for high *signal-to-noise ratios* (SNRs) [8], rendering them inadequate for applications with extreme reliability requirements.

Optimization based decoding algorithms are an entirely different way of approaching the decoding problem: they map the decoding problem onto an optimization problem in order to leverage the vast knowledge from the field of optimization theory. A number of different such algorithms have been introduced in the literature. The field of *linear programming* (LP) decoding [3], for example, represents one class of such algorithms, based on a relaxation of the *maximum likelihood* (ML) decoding problem as a linear program. Many different optimization algorithms can be used to solve the resulting problem [1], [9], [10]. Recently, proximal decoding for LDPC codes was presented by Wadayama *et al.* [11]. Proximal decoding relies on a non-convex optimization formulation of the *maximum a posteriori* (MAP) decoding problem.

The aim of this work is to improve the performance of proximal decoding by first presenting an analysis of the algorithm's behavior and then suggesting an approach to mitigate some of its flaws. This analysis is performed for *additive white Gaussian noise* (AWGN) channels. We first observe that the algorithm initially moves the estimate in the right direction; however, in the final steps of the decoding process, convergence to the correct codeword is often not achieved. Subsequently, we attribute this behavior to the nature of the decoding algorithm itself, comprising two separate gradient descent steps working adversarially.

We, thus, propose a method to mitigate this effect by appending an additional step to the iterative decoding process. In this additional step, the components of the estimate with the highest probability of being erroneous are identified. New codewords are then generated, over which an "ML-in-the-list" [4] decoding is performed. The main point of the paper at hand is to improve list generation such that it is especially tailored to the nature of proximal decoding. Using the improved algorithm, a gain of up to $1\,\mathrm{dB}$ can be achieved compared to conventional proximal decoding, depending on the decoder parameters and the code.

The authors are with the Communications Engineering Lab (CEL), Karlsruhe Institute of Technology (KIT), corresponding author: holger.jaekel@kit.edu This work has received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001899).

II. Preliminaries

A. Notation

When considering binary linear codes, data words are mapped onto codewords, the lengths of which are denoted by $k \in \mathbb{N}$ and $n \in \mathbb{N}$, respectively, with $k \leq n$. The set of codewords $\mathcal{C} \subset \mathbb{F}_2^n$ of a binary linear code can be characterized using the parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{m \times n}$, where m represents the number of parity-checks:

$$\mathcal{C} := \left\{ \boldsymbol{c} \in \mathbb{F}_2^n : \boldsymbol{H}\boldsymbol{c}^{\mathrm{T}} = \boldsymbol{0} \right\}$$

The check nodes indexed by $j \in \mathcal{J} := \{1, \ldots, m\}$ correspond to the parity checks, i.e., to the rows of $\boldsymbol{H}$. The variable nodes indexed by $i \in \mathcal{I} := \{1, \ldots, n\}$ correspond to the components of a codeword, i.e., to the columns of $\boldsymbol{H}$. The neighborhood of a parity check j, i.e., the set of component indices relevant for the according parity check, is denoted by $\mathcal{N}_{\mathrm{c}}(j) := \{i \in \mathcal{I} : \boldsymbol{H}_{j,i} = 1\}, \ \ j \in \mathcal{J}$.

In order to transmit a codeword $\boldsymbol{c} \in \mathbb{F}_2^n$, it is mapped onto a *binary phase shift keying* (BPSK) symbol via $\boldsymbol{x} = 1 - 2\boldsymbol{c}$, with $\boldsymbol{x} \in \{\pm 1\}^n$, which is then transmitted over an AWGN channel. The received vector $\boldsymbol{y} \in \mathbb{R}^n$ is decoded to obtain an estimate $\hat{\boldsymbol{c}} \in \mathbb{F}_2^n$ of the transmitted codeword. A distinction is made between $\boldsymbol{x} \in \{\pm 1\}^n$ and $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$, the former denoting the transmitted BPSK symbols and the latter being used as a variable during the optimization process. The likelihood of receiving $\boldsymbol{y}$ upon transmitting $\boldsymbol{x}$ is expressed by the *probability density function* (PDF) $f_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y} \mid \boldsymbol{x})$.

B. Proximal Decoding

With proximal decoding, the proximal gradient method [7] is used to solve a non-convex optimization formulation of the MAP decoding problem. With the equal prior probability assumption for all codewords, MAP and ML decoding are equivalent and, specifically for AWGN channels, correspond to a nearest-neighbor decision. For this reason, decoding can be carried out using a figure of merit that describes the distance from a given vector to a codeword. One such expression, formulated under the assumption of BPSK, is the *code-constraint polynomial* defined in [11]

$$h(\tilde{\boldsymbol{x}}) = \underbrace{\sum_{i=1}^{n} \left(\tilde{x}_i^2 - 1\right)^2}_{\text{Bipolar constraint}} + \underbrace{\sum_{j=1}^{m} \left[\left(\prod_{i \in \mathcal{N}_c(j)} \tilde{x}_i\right) - 1\right]^2}_{\text{Parity constraint}}.$$

Its intent is to penalize vectors far from a codeword and, thus, it serves as an objective function describing the quality of possible estimates. Please note that all valid codewords are local minima of $h(\tilde{\boldsymbol{x}})$. The code-constraint polynomial comprises two terms: the first part is representing the bipolar constraint due to using BPSK, whereas the second part is representing the parity constraint, incorporating all information regarding the code. Please note that the first part of the code-constraint polynomial may be easily adapted to higher order constellations, whereas the second part of the code-constraint polynomial requires bit values in $\mathbb{F}_2$. This can be achieved by employing a bit-metric decoder.

The channel can be characterized using the negative log-likelihood $L(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}) = -\ln(f_{\boldsymbol{Y}|\tilde{\boldsymbol{X}}}(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}))$. Then, the information about the channel and the code are consolidated in the objective function [11]

$$g(\tilde{\boldsymbol{x}}) = L(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}) + \gamma h(\tilde{\boldsymbol{x}}), \qquad \gamma > 0.$$

The objective function $g(\tilde{\boldsymbol{x}})$ is minimized using the proximal gradient method, which amounts to iteratively performing two gradient-descent steps [11] with the given objective function in AWGN channels. To this end, two helper variables $\boldsymbol{r}$ and $\boldsymbol{s}$ are introduced, describing the result of each of the two steps:

$$\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega \nabla L(\boldsymbol{y} \mid \boldsymbol{s}), \qquad \omega > 0, \tag{1}$$

$$\boldsymbol{s} \leftarrow \boldsymbol{r} - \gamma \nabla h(\boldsymbol{r}), \qquad \gamma > 0. \tag{2}$$

Derivation of $\nabla L(\boldsymbol{y} \mid \boldsymbol{s}) = \boldsymbol{s} - \boldsymbol{y}$ for AWGN and an equation for determining $\nabla h(\boldsymbol{r})$ are given in [11], where it is also proposed to initialize $\boldsymbol{s} = \boldsymbol{0}$. It should be noted that $\boldsymbol{r}$ and $\boldsymbol{s}$ represent $\tilde{\boldsymbol{x}}$ during different stages of the decoding process.

As the gradient of the code-constraint polynomial can attain very large values in some cases, an additional step is introduced in [11] to ensure numerical stability: every estimate $\boldsymbol{s}$ is projected onto the hypercube $[-\eta, \eta]^n$ by a projection $\Pi_\eta : \mathbb{R}^n \rightarrow [-\eta, \eta]^n$ defined as component-wise clipping, i.e., $\Pi_\eta(x_i) = \arg\min_{-\eta \le \xi \le \eta} |x_i - \xi|$ as in [11], where η is a positive constant larger than one, e.g., $\eta = 1.5$. The resulting decoding process is given in Algorithm 1.

Algorithm 1 Proximal decoding in AWGN [11]

$\boldsymbol{s} \leftarrow \boldsymbol{0}$
for K iterations **do**
    $\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega(\boldsymbol{s} - \boldsymbol{y})$
    $\boldsymbol{s} \leftarrow \Pi_\eta\left(\boldsymbol{r} - \gamma \nabla h(\boldsymbol{r})\right)$
    $\hat{\boldsymbol{c}} \leftarrow \mathbb{1}_{\{\boldsymbol{s} \le 0\}}$
    if $\boldsymbol{H}\hat{\boldsymbol{c}} = \boldsymbol{0}$ **do**
        return $\hat{\boldsymbol{c}}$
return $\hat{\boldsymbol{c}}$

III. Improved Algorithm

A. Analysis of the Convergence Behavior

In Fig. 1, the *frame error rate* (FER), *bit error rate* (BER), and *decoding failure rate* (DFR) of proximal decoding are shown for the LDPC code [204.33.484] [6] with $n = 204$ and $k = 102$. Hereby, a *decoding failure* is defined as returning a *non valid codeword*, i.e., as non-convergence of the algorithm. The parameters chosen in this simulation are $\gamma = 0.05$, $\omega = 0.05$, $\eta = 1.5$, and $K = 200$ (K describing the maximum number of iterations). They adhere to [11] and were determined to offer the best performance in a preliminary examination, where the effect of changing multiple parameters was simulated over a wide range of values. It is apparent that the DFR completely dominates the FER for sufficiently high SNR. This means that most frame errors are not due to the algorithm converging to the wrong codeword, but due to the algorithm not converging at all.

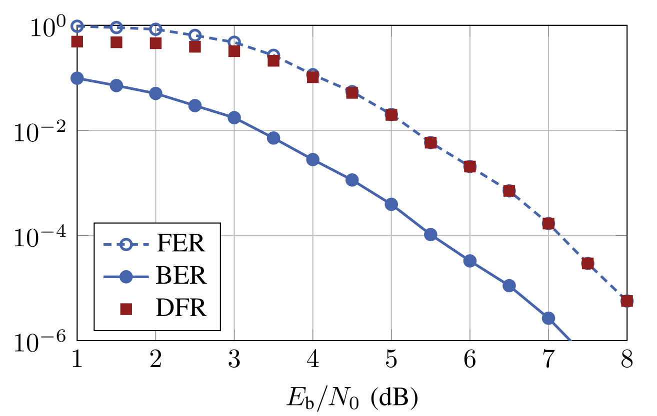

Fig. 1. FER, DFR, and BER for $(3, 6)$-regular LDPC code with $n = 204, k = 102$ [6, 204.33.484]. Parameters: $\gamma = 0.05, \omega = 0.05, \eta = 1.5, K = 200$.

As proximal decoding is an optimization-based decoding method, one possible explanation for this effect might be that during the decoding process, convergence to the final codeword is often not achieved, although the estimate is moving into the right direction. This would suggest that most frame errors occur due to only a few incorrectly decoded bits.

An approach for lowering the FER might then be to add an "ML-in-the-list" [4] step to the decoding process shown in Algorithm 1. This step consists in determining the $N \in \mathbb{N}$ positions $\mathcal{I}' \subset \mathcal{I}$ of bits that are most probably erroneous, generating a list of 2^N codeword candidates out of the current estimate $\hat{\boldsymbol{c}}$ with bits in $\mathcal{I}'$ adopting all possible values,i.e.,

$$\mathcal{L}' = \left\{ \hat{\boldsymbol{c}}' \in \mathbb{F}_2^n : \hat{c}'_i = \hat{c}_i, i \notin \mathcal{I}' \text{ and } \hat{c}'_i \in \mathbb{F}_2, i \in \mathcal{I}' \right\}, \quad (3)$$

and performing ML decoding on this list. Whereas list decoding is usually based on the analysis of received values, e.g., ML-in-the-list decoding or Chase decoding [2], the following consideration aims at generating this list by exploiting characteristic properties of proximal decoding.

The aforementioned process crucially relies on identifying the positions of bits that are most likely erroneous. Therefore, the convergence properties of proximal decoding are investigated. Fig. 2 shows the two gradients performed for a repetition code with $n = 2$. It is apparent that a net movement will result as long as the two gradients have a common component. As soon as this common component is exhausted, they will work in opposing directions resulting in an oscillation of the estimate. This behavior supports the conjecture that the reason for the high DFR is a failure to converge to the correct codeword in the final steps of the optimization process.

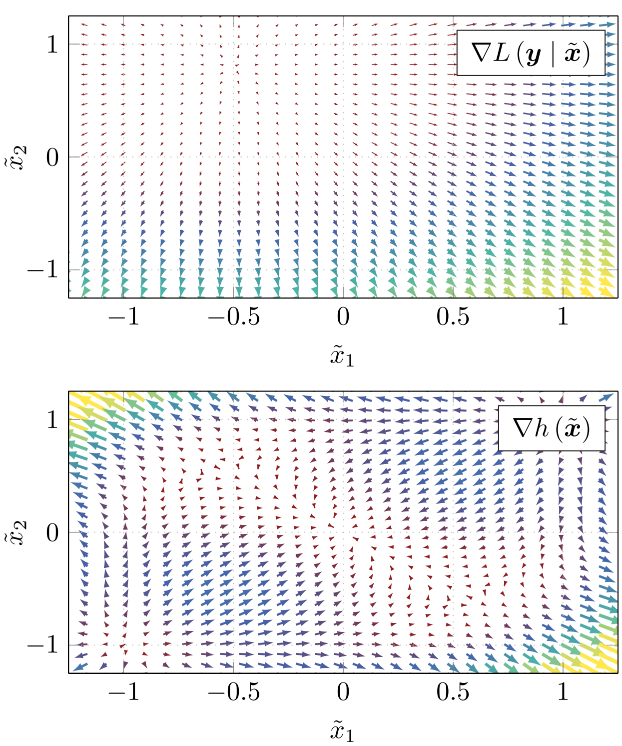

Fig. 2. Gradients $\nabla L(\boldsymbol{y} \mid \tilde{\boldsymbol{x}})$ and $\nabla h(\tilde{\boldsymbol{x}})$ for a repetition code with $n = 2$. Shown for $\boldsymbol{y} = \begin{pmatrix} -0.5 & 0.8 \end{pmatrix}$.

In Fig. 3, we show the component $\tilde{x}_1$ and corresponding gradients during decoding for the [204.33.484] LDPC code. We observe that both gradients start oscillating after a certain number of iterations. Furthermore, it can be observed the both gradients have approximately equal average magnitudes, but possess opposing signs, leading to an oscillation of $\tilde{x}_1$.

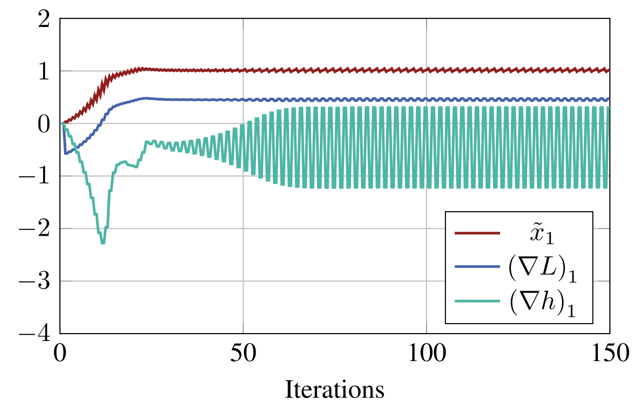

Fig. 3. Visualization of component $\tilde{x}_1$ for a decoding operation for a (3,6) regular LDPC code with $n = 204, k = 102$ [6, 204.33.484]. Parameters: $\gamma = 0.05, \omega = 0.05, \eta = 1.5, E_b/N_0 = 6\,\text{dB}$.

B. Improvement Using "ML-in-the-List" Step

Based on the observations depicted in Fig. 2 and Fig. 3, it seems a meaningful approach to tag the $N \in \mathbb{N}$ most likely erroneous bits based on the oscillation of the gradient of the code-constraint polynomial. To this end, let $\Delta_i^{(h)} := \left| (\nabla h)_i [K] - (\nabla h)_i [K-1] \right|$ be the oscillation height at the last iteration with $(\nabla h)_i [K]$ denoting the gradient at position i and iteration K. Now, let $\boldsymbol{i}' = (i'_1, \ldots, i'_n)$ be a permutation of $\{1, \ldots, n\}$ such that $\Delta_{i'}^{(h)}$ is arranged according to increasing oscillation height and select its N smallest indices, i.e.,

$$\boldsymbol{i}' = (i'_1, \ldots, i'_n) \in S_n : \left| \Delta_{i'_1}^{(h)} \right| \leq \cdots \leq \left| \Delta_{i'_n}^{(h)} \right| \quad (4)$$

$$\mathcal{I}' = \{i'_1, \ldots, i'_N\} \text{ with } \boldsymbol{i}' \text{ as defined in (4)} \quad (5)$$

with S_n denoting the symmetric group of $\{1, \ldots, n\}$. To reason this approach, Fig. 4 shows Monte Carlo simulations of the probability that the decoded bit $\hat{c}_{i'}$ at position i' of the estimated codeword is wrong. It can be observed that lower magnitudes of oscillation height correlate with a higher probability that the corresponding bit was not decoded correctly. Thus, the oscillation height might be used as a feasible indicator for identifying the N bits that are most likely erroneous.

The proposed improved algorithm is given in Algorithm 2. First, the proximal decoding Algorithm 1 is applied. If a valid codeword has been reached, i.e., if the algorithm has converged, we return this solution. Otherwise, $N \in \mathbb{N}$ components are selected as described in eq. (5). Originating from $\hat{\boldsymbol{c}} \in \mathbb{F}_2^n$, the result of proximal decoding, the list $\mathcal{L}'$ of codeword candidates with bits in $\mathcal{I}'$ modified is generated and an "ML-in-the-list" step is performed. If the list $\mathcal{L}'$ does not contain a valid codeword and, thus, $\mathcal{L}'_{\text{valid}} = \emptyset$, the additional step boils down to the maximization of 2^N correlations $\langle 1 - 2\boldsymbol{c}'_l, \boldsymbol{y} \rangle$, $\boldsymbol{c}'_l \in \mathcal{L}'$, in which ties happen with probability zero and are solved arbitrarily. Note that 2^N parity checks have to be evaluated for elements in $\mathcal{L}'$ in order to determine $\mathcal{L}'_{\text{valid}}$ either way. Restricting the correlations to the (non-empty) list $\mathcal{L}'_{\text{valid}}$ may reduce the computational burden and ensure that a valid codeword is returned.

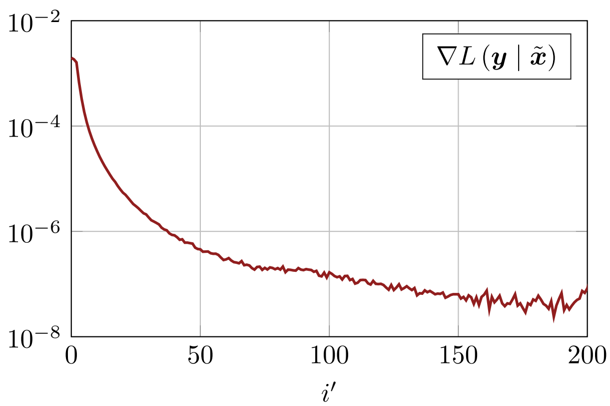

Fig. 4. Probability that $P(\hat{c}_{i'} \neq c_{i'})$ for a (3,6) regular LDPC code with $n = 204, k = 102$ [6, 204.33.484]. Indices i' are ordered as in eq. (4). Parameters: $\gamma = 0.05, \omega = 0.05, \eta = 1.5, E_\mathrm{b}/N_0 = 6\,\mathrm{dB}$, 10^9 codewords.

Algorithm 2 Proposed improved proximal decoding in AWGN

$\hat{\boldsymbol{c}} \leftarrow$ proximal decoding$(\boldsymbol{y})$
if $\boldsymbol{H}\hat{\boldsymbol{c}} = \boldsymbol{0}$ **do**
  return $\hat{\boldsymbol{c}}$
$\mathcal{I}' \leftarrow \{i'_1, \ldots, i'_N\}$ (indices of N probably wrong bits)
$\mathcal{L}' \leftarrow \left\{\hat{\boldsymbol{c}}' \in \mathbb{F}_2^n : \hat{c}'_i = \hat{c}_i, i \notin \mathcal{I}' \text{ and } \hat{c}'_i \in \mathbb{F}_2, i \in \mathcal{I}'\right\}$
$\mathcal{L}'_\text{valid} \leftarrow \{\hat{\boldsymbol{c}}' \in \mathcal{L}' : \boldsymbol{H}\hat{\boldsymbol{c}}' = \boldsymbol{0}\}$ (select valid codewords)
if $\mathcal{L}'_\text{valid} \neq \emptyset$ **do**
  return $\arg\max\{\langle 1 - 2\hat{\boldsymbol{c}}'_l, \boldsymbol{y}\rangle : \hat{\boldsymbol{c}}'_l \in \mathcal{L}'_\text{valid}\}$
else
  return $\arg\max\{\langle 1 - 2\hat{\boldsymbol{c}}'_l, \boldsymbol{y}\rangle : \hat{\boldsymbol{c}}'_l \in \mathcal{L}'\}$

IV. Simulation Results & Discussion

Fig. 5 shows the FER and BER resulting from applying proximal decoding as presented in [11] and the proposed improved algorithm, when both are applied to the $(3,6)$-regular LDPC code [204.33.484] [6] with $n = 204$ and $k = 102$. The parameters chosen for the simulation are $\gamma = 0.05, \omega = 0.05, \eta = 1.5, K = 200$ as for proximal decoding, since those parameters also turned out close-to-optimum for the improved algorithm in our simulations. The number of possibly wrong components was selected as $N = 8$. To reason this choice, Table I shows the SNRs required for $N \in \{4, 6, 8, 10, 12\}$ to achieve an FER of 10^{-2} and 10^{-3}, respectively.

TABLE I
SNR (in dB) to achieve target FERs 10^{-2} and 10^{-3}

N	4	6	8	10	12
FER $= 10^{-2}$	4.94	4.76	4.67	4.60	4.54
FER $= 10^{-3}$	5.84	5.61	5.49	5.39	5.32

A noticeable improvement can be observed both in the FER and the BER. The gain varies significantly with the SNR, which is to be expected since higher SNR values result in a decreased number of bit errors, making the correction of those errors in the "ML-in-the-list" step more likely. For an FER of 10^{-6}, the gain is approximately $1\,\mathrm{dB}$. As shown in Fig. 5, it can be seen that BP decoding with 200 iterations outperforms the improved scheme by approximately $1.7\,\mathrm{dB}$. Nevertheless, note that Algorithm 2 requires only linear operations and could be favorable in applications as, e.g., massive MIMO, in which application of BP is prohibitive [11]. Similar behavior to Fig. 5 was observed with a number of different codes, e.g., [6, PEGReg252x504, 204.55.187, 96.3.965]. Furthermore, we did not observe an immediate relationship between the code length and the gain during our examinations.

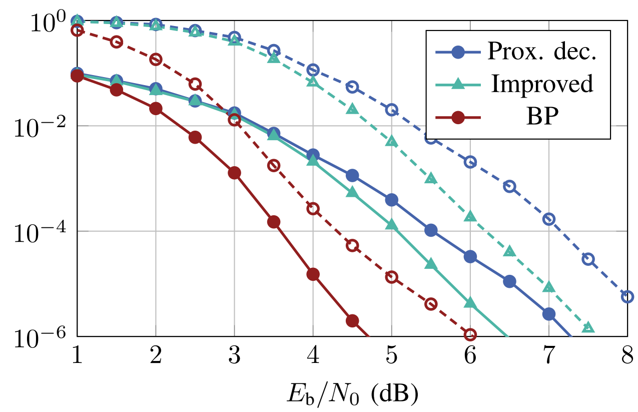

Fig. 5. FER (- - -) and BER (—) of proximal decoding [11] and the improved algorithm for a $(3,6)$-regular LDPC code with $n = 204, k = 102$ [6, 204.33.484]. Parameters: $\gamma = 0.05, \omega = 0.05, \eta = 1.5, K = 200, N = 8$.

V. Conclusion

In this paper, an improvement on proximal decoding as presented by Wadayama *et al.* [11] is proposed for AWGN channels. It relies on the fact that most errors observed in proximal decoding stem from only a few components of the estimate being wrong. These few erroneous components can mostly be corrected by appending an additional step to the original algorithm that is only executed if the algorithm has not converged. A gain of up to $1\,\mathrm{dB}$ can be observed, depending on the code, the code parameters, and the SNR.

References

[1] S. Barman, X. Liu, S. C. Draper, and B. Recht, "Decomposition methods for large scale lp decoding," *Trans. Inf. Theory*, vol. 59, no. 12, pp. 7870–7886, 2013.
[2] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inf. Theory*, 1972.
[3] J. Feldman, M. Wainwright, and D. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, 2005.
[4] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. t. Brink, "Automorphism ensemble decoding of reed–muller codes," *IEEE Trans. Commun.*, vol. 69, no. 10, pp. 6424–6438, 2021.
[5] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, 1999.
[6] D. J. MacKay. "Encyclopedia of sparse graph codes." (Apr. 2023), [Online]. Available: www.inference.org.uk/mackay/codes/data.html.
[7] N. Parikh, S. Boyd, *et al.*, *Proximal algorithms*. Now Publishers, Inc., 2014, vol. 1, pp. 127–239.
[8] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.
[9] M. H. Taghavi and P. H. Siegel, "Adaptive linear programming decoding," in *IEEE Proc. ISIT*, 2006, pp. 1374–1378.
[10] P. O. Vontobel, "Interior-point algorithms for linear-programming decoding," in *Proc. ITA*, 2008, pp. 433–437.
[11] T. Wadayama and S. Takabe, "Proximal decoding for LDPC codes," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2022.