Efficient Maximum-Likelihood Decoding of Linear Block Codes on Binary Memoryless Channels

Michael Helmling[†], Eirik Rosnes[§], Stefan Ruzika[†], and Stefan Scholl[‡]
[†]Mathematical Institute, University of Koblenz-Landau, 56070 Koblenz, Germany
Email: {helmling, ruzika}@uni-koblenz.de
[§]Department of Informatics, University of Bergen, N-5020 Bergen, Norway, and the Simula Research Lab.
Email: eirik@ii.uib.no
[‡]Microelectronic Systems Design Research Group, University of Kaiserslautern, 67653 Kaiserslautern, Germany
Email: scholl@eit.uni-kl.de

***Abstract*—In this work, we consider efficient maximum-likelihood decoding of linear block codes for small-to-moderate block lengths. The presented approach is a branch-and-bound algorithm using the cutting-plane approach of Zhang and Siegel (*IEEE Trans. Inf. Theory*, 2012) for obtaining lower bounds. We have compared our proposed algorithm to the state-of-the-art commercial integer program solver CPLEX, and for all considered codes our approach is faster for both low and high signal-to-noise ratios. For instance, for the benchmark $(155, 64)$ Tanner code our algorithm is more than 11 times as fast as CPLEX for an SNR of $1.0\,\mathrm{dB}$ on the additive white Gaussian noise channel. By a small modification, our algorithm can be used to calculate the minimum distance, which we have again verified to be much faster than using the CPLEX solver.**

I. Introduction

Determining the optimal decoding behavior of error-correcting codes is of significant importance, e. g., to benchmark different coding schemes. When no *a priori* information on the transmitted codeword is known, maximum-likelihood decoding (MLD) is an optimal decoding strategy. It is known that this problem is NP-hard in general [1] such that its complexity grows exponentially in the block length of the code, unless $\mathrm{P} = \mathrm{NP}$. Currently, the best known approach for general block codes is to use a state-of-the-art (commercial) integer program (IP) solver (see [2]) like CPLEX [3].

In this work, we present a branch-and-bound approach for efficient MLD of linear block codes. The problem of MLD is closely related to that of calculating the minimum distance of a code, which has attracted some attention recently. For instance, in [4, 5], Rosnes *et al.* proposed an efficient branch-and-bound algorithm to determine all low-weight stopping sets/codewords in a low-density parity-check (LDPC) code. Although the two problems are similar, the bounding step in the algorithm from [4, 5] cannot efficiently be adapted to the scenario of MLD. Conversely, however, the algorithm presented here can also calculate the minimum distance, and our numerical experiments show that this is very efficient compared to CPLEX.

This work was partially funded by the DFG (grant RU-1524/2-1) and by DAAD / RCN (grant 54565400 within the German-Norwegian Collaborative Research Support Scheme).

Linear programming (LP) decoding of binary linear codes, as first introduced by Feldman *et al.* in [6], approximates MLD by relaxing the decoding problem into an easier to solve LP problem. The LP problem contains a set of linear inequalities that are derived from the parity-check constraints of a (redundant) parity-check matrix representing the code. As shown in [7], these constraints can iteratively and adaptively be added to the decoding problem, which significantly reduces the overall complexity of LP decoding. Elaborating on this idea, Zhang and Siegel [8] proposed an efficient search algorithm for new *violated* (redundant) parity-check constraints (or “cuts”) that tighten the decoding polytope. Depending on the structure of the underlying code, for some codes, this “cutting-plane” LP decoding algorithm performs close to MLD for high signal-to-noise ratios (SNRs) on the additive white Gaussian noise (AWGN) channel, although for most codes, e. g., the $(155, 64)$ Tanner code [9], there is still a gap in decoding performance to MLD [8]. For lower values of the SNR, there could be a significant performance degradation with respect to MLD.

The algorithm proposed in this work closes that gap by using the cutting-plane algorithm for lower bounds and the well-known sum-product (SP) decoder [10] with order-i re-encoding [11] for upper bounds within a sophisticated branch-and-bound framework such that the output always and provably is the maximum-likelihood (ML) codeword. Our numerical study in Section VI shows that it is much faster than CPLEX for all codes under consideration, and moreover is able to decode some of the codes on which CPLEX fails completely.

II. Notation and Background

This section establishes some basic definitions and results needed for the rest of the paper.

Let $\mathcal{C}$ denote a binary linear code of length n represented by an $m \times n$ parity-check matrix $\mathbf{H}$. The code is used on a binary-input memoryless output-symmetric channel with input $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathcal{C}$ and channel output denoted by the length-n vector $\mathbf{r} = (r_0, \ldots, r_{n-1})$. The ML decoder can be described by the following optimization problem [12]:

$$\hat{\mathbf{c}}_{\mathrm{ML}} = \arg\min_{\mathbf{c}\in\mathcal{C}} \psi_{\boldsymbol{\lambda}}(\mathbf{c}) = \arg\min_{\mathbf{c}\in\mathrm{conv}(\mathcal{C})} \psi_{\boldsymbol{\lambda}}(\mathbf{c}) \tag{1}$$

where $\psi_{\boldsymbol{\lambda}}(\mathbf{c}) = \boldsymbol{\lambda}\cdot\mathbf{c}^T$ and $(\cdot)^T$ denotes the transpose of its argument, $\boldsymbol{\lambda} = (\lambda_0, \dots, \lambda_{n-1})$ is a vector of log-likelihood ratios (LLRs) defined by

$$\lambda_i = \log\left(\frac{\Pr(r_i|c_i=0)}{\Pr(r_i|c_i=1)}\right)$$

for all i, $0 \le i \le n-1$, and $\mathrm{conv}(\mathcal{C})$ is the *convex hull* of $\mathcal{C}$ in $\mathbb{R}^n$, where $\mathbb{R}$ denotes the real numbers. The MLD problem in (1) can be formulated as an IP which in general is an NP-hard problem. As an approximation to MLD, Feldman *et al.* [6] relaxed the codeword polytope $\mathrm{conv}(\mathcal{C})$ in the following way.

Define

$$\mathcal{C}_j = \{\mathbf{c}\in\{0,1\}^n \,|\, \mathbf{h}_j\cdot\mathbf{c}^T = 0\}$$

where $\mathbf{h}_j = (h_{j,0}, \dots, h_{j,n-1})$ is the jth row of the parity-check matrix $\mathbf{H}$ and $0 \le j \le m-1$. Furthermore, let $\mathrm{conv}(\mathcal{C}_j)$ denote the convex hull of $\mathcal{C}_j$ in $\mathbb{R}^n$. The *fundamental polytope* $\mathcal{P}(\mathbf{H})$ of the parity-check matrix $\mathbf{H}$ is defined as [13]

$$\mathcal{P}(\mathbf{H}) = \bigcap_{j=0}^{m-1} \mathrm{conv}(\mathcal{C}_j). \tag{2}$$

The MLD problem in (1) can now be relaxed to

$$\hat{\mathbf{p}}_{\mathrm{LP}} = \arg\min_{\mathbf{p}\in\mathcal{P}(\mathbf{H})} \psi_{\boldsymbol{\lambda}}(\mathbf{p})$$

where the solution, by definition, is a *pseudocodeword* with fractional entries in general. Note that the LP decoder has the *ML certificate* property, which means that in case $\hat{\mathbf{p}}_{\mathrm{LP}}$ is integral, it is an optimal solution to (1).

For each row $\mathbf{h}_j$, $0 \le j \le m-1$, in the matrix $\mathbf{H}$ the linear inequalities behind the fundamental polytope in (2) are

$$\sum_{i\in\mathcal{V}} p_i - \sum_{i\in\mathcal{N}(j)\setminus\mathcal{V}} p_i \le |\mathcal{V}|-1, \text{ for all odd-sized } \mathcal{V}\subseteq\mathcal{N}(j) \tag{3}$$

where $\mathcal{N}(j) = \{i : h_{j,i} = 1,\ 0 \le i \le n-1\}$.

For a given row $\mathbf{h}_j$ of a parity-check matrix $\mathbf{H}$ and a vector $\mathbf{p}\in[0,1]^n$: If there exists an odd set $\mathcal{V}\subseteq\mathcal{N}(j)$ such that the corresponding inequality from (3) does not hold, then we say that the jth parity-check constraint induces a *cut* at $\mathbf{p}$.

Central to our branch-and-bound algorithm is the concept of *constraint sets*. A constraint set F is a set $\{(\rho_i, c_{\rho_i}) : c_{\rho_i}\in\{0,1\}\ \forall\rho_i\in\Gamma\}$, where $\Gamma\subseteq\{0,\dots,n-1\}$. If (ρ_i, c_{ρ_i}) is a constraint, then position ρ_i is said to be c_{ρ_i}-constrained, which means that position ρ_i is committed to the value c_{ρ_i} in a codeword, while positions not in F are uncommitted.

Let $\mathcal{C}^{(F)}$ denote the subset of codewords from $\mathcal{C}$ consistent with the constraint set F. Then, we can define

$$\psi_{\min,\boldsymbol{\lambda}}^{(F)} = \min_{\mathbf{c}\in\mathcal{C}^{(F)}} \psi_{\boldsymbol{\lambda}}(\mathbf{c})$$

as the minimum value of the objective function for codewords consistent with F. In the following, $\bar{\psi}_{\min,\boldsymbol{\lambda}}^{(F)}$ will denote any lower bound on $\psi_{\min,\boldsymbol{\lambda}}^{(F)}$. Also, a constraint set F is said to be *valid* if $\mathcal{C}^{(F)}$ is nonempty.

Our proposed branch-and-bound MLD algorithm relies heavily on tight lower bounds on $\psi_{\min,\boldsymbol{\lambda}}^{(F)}$, which are provided by an LP-based decoding algorithm by Zhang and Siegel [8]. We briefly describe this algorithm, denoted as the ZS decoding algorithm, below in Section II-A.

A. Zhang and Siegel's LP-Based Decoding Algorithm

The ZS decoding algorithm is based on adaptive LP decoding as described in [7] and incorporates an efficient cut-search algorithm, as described in [8]. First the LP problem is initialized with the box constraints. †Solve the LP problem to get an optimal solution $\mathbf{p}^*$. If $\mathbf{p}^*$ is integral, then terminate the algorithm and return the ML codeword $\mathbf{p}^*$. Otherwise, the cut-search algorithm (Algorithm 1 in [8]) is applied to each row of the parity-check matrix of the code. If at least one is found, add all found cuts into the LP problem and repeat the procedure from †. Otherwise, search for cuts from redundant parity-checks. To this end, reduce $\mathbf{H}$ by Gaussian elimination to *reduced row echelon* form, where the columns of $\mathbf{H}$ are processed in the order of the "fractionality" (i. e., closeness to $\frac{1}{2}$) of the corresponding coordinate of $\mathbf{p}^*$. Now, the cut-search algorithm is applied to each row of the obtained modified matrix $\tilde{\mathbf{H}}$. If no cut is found, then terminate. Otherwise, add all found cuts into the LP problem as constraints and repeat the procedure from †. The algorithm above has been detailed in Algorithm 2 in [8], and we refer the interested reader to [8] for further details.

III. Basic Branch-and-Bound Algorithm

Our proposed algorithm is a branch-and-bound algorithm on constraint sets and uses the ZS decoding algorithm, as briefly described above, as a basic component in the bounding step. Thus, there is a one-to-one correspondence between the nodes in the search tree and constraint sets. In the following, when we speak about the *left* and *right* child constraint set, denoted by $F^{\downarrow 0}$ and $F^{\downarrow 1}$, respectively, we mean the constraint set of the left and right child nodes in the search tree.

Now, to each constraint set F, we associate three real numbers $\bar{\psi}_{\min,\boldsymbol{\lambda}}^{(F)}$, $\bar{\psi}_{\min,\boldsymbol{\lambda}}^{(F)\downarrow 0}$, and $\bar{\psi}_{\min,\boldsymbol{\lambda}}^{(F)\downarrow 1}$ which are *current* lower bounds on $\psi_{\min,\boldsymbol{\lambda}}^{(F)}$, $\psi_{\min,\boldsymbol{\lambda}}^{(F^{\downarrow 0})}$, and $\psi_{\min,\boldsymbol{\lambda}}^{(F^{\downarrow 1})}$, respectively. When a constraint set is created, these values are initiated to $-\infty$.

The algorithm maintains a list L of active constraint sets which is initiated with the unconstrained set $\emptyset$. In each iteration, a constraint set F is selected from the list according to the node selection rule (see below). A valid codeword, i. e., a feasible solution of the IP, is generated by decoding the LLR vector (where the constraints imposed by F are enforced by altering the according LLR values to $\pm\infty$) using the SP algorithm [10] with order-i re-encoding [11], for some integer i, as a post-processing step. The upper bound on the objective function decreases if the decoder output has a lower objective function value than the previous best candidate. Afterwards a lower bound on $\psi_{\min,\boldsymbol{\lambda}}^{(F)}$ is computed by running the ZS algorithm, where the variables contained in F are fixed to their corresponding values. If an integral solution is returned, i. e., a pseudocodeword with no fractional coordinates, it is considered another candidate codeword in the same way as above. Otherwise, and if the computed lower bound is less than the current upper bound τ, the algorithm branches on a fractional position, selected by the branching rule (see below), of the pseudocodeword by adding two constraint sets, namely F augmented by the chosen branching position fixed to 0 and 1, respectively, to the list of active nodes. Then, the next set is chosen from L until one of the termination criteria in Step 2 of Algorithm 1 (which gives a formal description of the overall algorithm) is fulfilled. Note that the computations to produce a lower bound on $\psi^{(F)}_{\min,\boldsymbol{\lambda}}$ for a given constraint set F are collected into Algorithm 2, denoted by LUBD.

A. Bounding Step

The complexity of Algorithm 1 depends heavily on the tightness of the lower bounds computed in Step 5 (from Algorithm 2), i. e., on how close $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}})$ is to the value $\psi^{(F)}_{\min,\boldsymbol{\lambda}}$. To find the best pseudocodeword $\hat{\mathbf{p}}$, we have used the procedure detailed in Algorithm 2.

Note that $\min\{\psi^{(F\downarrow 0)}_{\min,\boldsymbol{\lambda}}, \psi^{(F\downarrow 1)}_{\min,\boldsymbol{\lambda}}\} = \psi^{(F)}_{\min,\boldsymbol{\lambda}}$ for any node (constraint set) F. This allows us to update the current lower bound $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}}$ of F (and, recursively, also the ancestors of F), potentially *increasing* its value, once both of its children have been processed (see Steps 14–18 of Algorithm 1). Tightening the bounds in the search tree is important for decreasing the complexity of the algorithm because nodes whose lower bound exceeds the objective value of the currently best candidate solution (i. e., the current upper bound) can be skipped, thereby reducing the search space.

B. Branching Step

We have used the following simple branching rule to select the position p in Step 13 of Algorithm 1: Take an unconstrained position where the corresponding entry in the decoded pseudocodeword $\hat{\mathbf{p}}$ is closest to $\frac{1}{2}$. This simple procedure seems to work very well in practice.

C. The Processing Order of the List L

The node selection rule, i. e., the method by which a constraint set F is selected from L in Step 3 of Algorithm 1, has great influence on the overall complexity. The most common schemes are *depth-first search*, according to LIFO (last in – first out) processing of L, and *breadth-first search*, where L is processed in FIFO (first in – first out) fashion. Another popular method, called *best-bound search*, selects the next constraint set by $F' = \arg\min_{F\in L} \bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}}$, with the goal of tightening the overall lower bound as fast as possible.

In our experiments, the following mixed strategy has proven to be most efficient. Apply depth-first processing in general, but every M iterations, for a fixed integer M, and only if $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}} < \tau - \delta$ for a fixed $\delta > 0$, where F is the constraint set from the previous iteration, select the next node by the best-bound rule above.

IV. Improvements

In this section, we present some improvements to the basic algorithm from Section III.

Algorithm 1 Maximum-Likelihood Decoding (MLD)

Input: The received LLR vector $\boldsymbol{\lambda}$ and the order i of re-encoding.
Output: An ML decoded codeword $\hat{\mathbf{c}}_{\mathrm{ML}}$.

```
1:  Initialize $\tau \leftarrow \infty$ and $L \leftarrow \{\emptyset\}$
2:  while $L \neq \emptyset$ and $\bar{\psi}^{(\emptyset)}_{\min,\boldsymbol{\lambda}} < \tau$ do
3:    Choose and remove a constraint set $F$ from $L$.
4:    if $F$ is valid and $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}} < \tau$ then
5:      let $(\hat{\mathbf{c}}, \hat{\mathbf{p}}) \leftarrow \mathrm{LUBD}(\boldsymbol{\lambda}, F)$
6:      if $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{c}}) < \tau$ then
7:        let $\hat{\mathbf{c}}_{\mathrm{ML}} \leftarrow \hat{\mathbf{c}}$ and $\tau \leftarrow \psi_{\boldsymbol{\lambda}}(\hat{\mathbf{c}})$
8:      $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}} \leftarrow \psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}})$
9:      if $\hat{\mathbf{p}}$ is integral then
10:       if $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}}) < \tau$ then
11:         let $\hat{\mathbf{c}}_{\mathrm{ML}} \leftarrow \hat{\mathbf{p}}$ and $\tau \leftarrow \psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}})$
12:     else if $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}}) < \tau$ then
13:       choose an unconstrained position $p$ based on $\hat{\mathbf{p}}$,
          construct two new constraint sets $F' = F \cup \{(p,0)\}$
          and $F'' = F \cup \{(p,1)\}$, and append them to $L$.
14:   if $F \neq \emptyset$ then
15:     determine (the unique) $\tilde{F}$ such that $F = \tilde{F}^{\downarrow i}$, where
        $i = 0$ or 1, and update parent bounds as follows:
16:     $\bar{\psi}^{(\tilde{F})\downarrow i}_{\min,\boldsymbol{\lambda}} \leftarrow \max\{\bar{\psi}^{(\tilde{F})\downarrow i}_{\min,\boldsymbol{\lambda}}, \bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}}\}$
17:     $\bar{\psi}^{(\tilde{F})}_{\min,\boldsymbol{\lambda}} \leftarrow \max\{\bar{\psi}^{(\tilde{F})}_{\min,\boldsymbol{\lambda}}, \min\{\bar{\psi}^{(\tilde{F})\downarrow 0}_{\min,\boldsymbol{\lambda}}, \bar{\psi}^{(\tilde{F})\downarrow 1}_{\min,\boldsymbol{\lambda}}\}\}$
18:     If $\bar{\psi}^{(\tilde{F})}_{\min,\boldsymbol{\lambda}}$ increased in the previous step, recurse to
        Step 14 with $F$ replaced by $\tilde{F}$.
19: Return $\hat{\mathbf{c}}_{\mathrm{ML}}$.
```

Algorithm 2 Lower and Upper Bound Algorithm (LUBD)

Input: The received LLR vector $\boldsymbol{\lambda}$ and a constraint set F.
Output: The pair $(\hat{\mathbf{c}}, \hat{\mathbf{p}})$.

```
1: Perform SP decoding with order-$i$ re-encoding on an LLR
   vector constrained according to $F$ as follows:
     • $+\infty$ for positions corresponding to 0-constraints.
     • $-\infty$ for positions corresponding to 1-constraints.
     • The original channel LLRs for positions not in $F$.
   The resulting decoded codeword is denoted by $\hat{\mathbf{c}}$.
2: Perform ZS decoding on the received LLR vector $\boldsymbol{\lambda}$ with
   equality constraints according to $F$. Denote the decoded
   pseudocodeword by $\hat{\mathbf{p}}$.
3: Return the pair $(\hat{\mathbf{c}}, \hat{\mathbf{p}})$.
```

A. Tuning the ZS Algorithm for Adaptive LP Decoding

A linear inequality constraint of the general form $\mathbf{a}\cdot\mathbf{x}^T \leq b$, where $\mathbf{a}$ and b are constants, is called *active* at the point $\mathbf{x}^*$ if it holds with equality for $\mathbf{x} = \mathbf{x}^*$. Otherwise, it is called *inactive*. For an LP problem with a set of linear inequality constraints, the optimal solution $\mathbf{x}_{\mathrm{LP}}$ is a *vertex* of the polytope formed by the hyperplanes of all constraints that are active at $\mathbf{x}_{\mathrm{LP}}$. Thus, constraints inactive at $\mathbf{x}_{\mathrm{LP}}$ can be removed without changing the optimal solution.

The ZS decoding algorithm uses adaptive LP decoding, which implies that a lot of linear programs (of increasing size in the number of constraints) are solved successively. Consequently, a simple way to reduce the overall complexity is to remove inactive constraints from time to time.

Our implementation uses the *dual simplex method* for solving LP problems, which is very effective in the case of iteratively added constraints by employing a warm-start technique that reuses the basis information of the previously optimal solution (see, e. g., [14] for details). The removal of constraints, however, is expensive because afterwards a new simplex basis has to be computed. Thus, we remove the inactive constrains only when the number of constraints in the current LP problem exceeds T, for some integer T. Note that this differs from the algorithm called MALP-B in [8], where the inactive constraints are removed in each iteration.

Another way to decrease the running-time of the ZS algorithm in some cases is to terminate the ZS decoder prematurely as soon as the objective value exceeds the current upper bound τ in Algorithm 1. In that event the objective function value cannot possibly be improved below the current node, and it can be skipped immediately.

B. *Tradeoff Between Tightness and Speed of the ZS Algorithm*

The cut-search procedure used in the ZS decoding algorithm yields tight lower bounds on the MLD solution at the cost of a high number of cuts and thus increased processing time spent in the LP solver and for Gaussian elimination (see Section II-A). In two different ways, a tradeoff between speed and tightness can be realized. First, limit the maximum number of times R that the search for redundant parity-check cuts is applied, and secondly, only add a cut if the *cutoff*, i. e., the distance between the current (infeasible) solution and the cutting hyperplane, exceeds a fixed quantity $\gamma > 0$.

Our numerical experiments have shown that both approaches help to significantly reduce the running-time complexity of our algorithm. Additionally, for the first approach, it has proven helpful to use a higher value R^{bb} in those iterations where a best-bound node has been selected (cf. Section III-C).

C. *Special Case: MLD Performance Simulation*

For benchmarking purposes we are only interested in the actual MLD curve, in which case the MLD algorithm can be simplified. First, since the underlying code is always linear, the error probability of MLD is independent of the actual transmitted codeword, thus we can always, without loss of generality, transmit the all-zero codeword. Furthermore, when the all-zero codeword is transmitted and a codeword $\mathbf{c}$ with objective value $\psi_{\boldsymbol{\lambda}}(\mathbf{c}) < 0 = \psi_{\boldsymbol{\lambda}}(\mathbf{0})$ has been identified, the search can be terminated, since any ML decoder would also fail on this received LLR vector $\boldsymbol{\lambda}$.

V. Minimum Distance Computation

The MLD problem is closely related to the computation of the minimum distance $d_{\min}$ of a code as follows. If the all-zero codeword is explicitly forbidden, then an MLD algorithm with the input $\boldsymbol{\lambda} = \mathbf{1}$ will output a codeword of minimum weight:

$$d_{\min}(\mathcal{C}) = \min_{\mathbf{c}\in\mathcal{C}\setminus\{\mathbf{0}\}} \psi_{\mathbf{1}}(\mathbf{c}) = \min_{\mathbf{c}\in\operatorname{conv}(\mathcal{C}\setminus\{\mathbf{0}\})} \psi_{\mathbf{1}}(\mathbf{c}). \quad (4)$$

Our proposed decoding algorithm can be modified to exclude the all-zero codeword by the following changes:

1) Extend the condition in Step 9 of Algorithm 1 to "$\hat{\mathbf{p}}$ is integral and $\hat{\mathbf{p}} \neq \mathbf{0}$", which avoids decoding to the all-zero codeword.
2) In the order-i re-encoding performed in Algorithm 2, exclude $\mathbf{0}$ from the set of candidate codewords.

Moreover, note that all feasible solutions (i. e., codewords) of (4) have an integral objective value. This allows us to change the right hand side in Steps 2, 4, and 12 to $\tau - 1 + \varepsilon$, for a small $\varepsilon > 0$, since $\left\lceil \bar{\psi}^{(F)}_{\min,\mathbf{1}} \right\rceil = \tau$ implies that $\psi^{(F)}_{\min,\mathbf{1}} \geq \tau$.

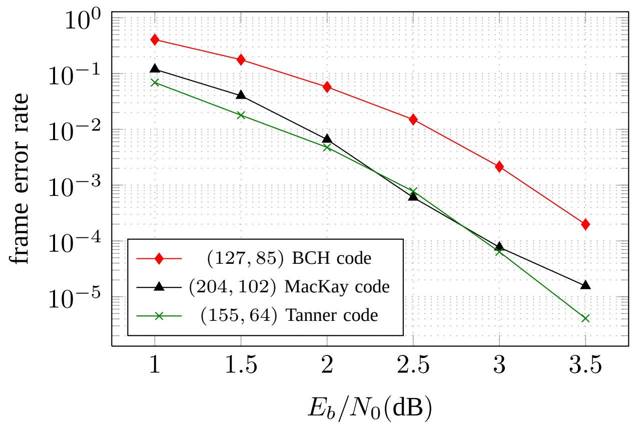

Fig. 1. MLD performance of the codes considered in this paper.

VI. Numerical Results

In this section, we present some numerical results for our proposed MLD algorithm, with all the improvements outlined above in Section IV, for several codes on the AWGN channel. We have used order-2 re-encoding ($i = 2$ in Algorithm 2), and the open-source GLPK library [16] to solve the LP problems. The following set of parameters was heuristically found to perform well for all codes: $M = 30$, $\delta = 2$, $T = 100$, $R = 5$, $R^{\text{bb}} = 100$, and $\gamma = 0.2$.

As a benchmark, we use the CPLEX IP solver [3], with the IP formulation named IPD1 in [2] which was found to be most efficient in that paper. In case of all-zero decoding, we configured CPLEX to terminate as soon as a codeword with objective value below zero was found, mimicking the adaptions of our algorithm described in Section IV-C.

We compare the algorithms with respect to both (single-core) average CPU time T_{avg} and average number N_{avg} of branch-and-bound nodes processed per frame. For our algorithm, N_{avg} equals the number of times the main loop (Step 2 of Algorithm 1) is processed, while for CPLEX we report the attribute "number of processed nodes". Note that the latter drops below one for high SNR, which is probably due to CPLEX' presolve strategy that establishes optimality in some cases without ever starting the branch-and-bound procedure.

TABLE I
NUMERICAL COMPARISON OF OUR PROPOSED ALGORITHM AND CPLEX FOR SEVERAL CODES FOR DIFFERENT VALUES OF THE SNR ON THE AWGN CHANNEL. T_{avg} IS THE AVERAGE DECODING TIME PER FRAME IN SECONDS, AND N_{avg} IS THE AVERAGE NUMBER OF NODES (PER FRAME) PROCESSED BY THE BRANCH-AND-BOUND ALGORITHMS. THE NUMBERS IN THE PARENTHESES ARE FOR CPLEX. IN ALL CASES BUT THE FIRST WE USED ALL-ZERO DECODING AS DESCRIBED IN SECTION IV-C.

	SNR in dB	1.0	1.5	2.0	2.5	3.0	3.5
$(155, 64)$ Tanner code [9]	T_{avg}	0.81 (9.49)	0.24 (2.95)	0.05 (0.63)	0.014 (0.17)	0.005 (0.095)	0.004 (0.086)
	N_{avg}	51 (4795)	15 (1816)	3.5 (370)	1.4 (61)	1.04 (5.4)	1.004 (0.3)
$(155, 64)$ Tanner code [9] (all-zero)	T_{avg}	0.24 (3.23)	0.11 (1.16)	0.025 (0.28)	0.006 (0.06)	0.001 (0.02)	0.0005 (0.01)
	N_{avg}	14 (2799)	6.6 (963)	2.1 (210)	1.18 (38)	1.05 (4.5)	1.002 (0.3)
$(204, 102)$ MacKay code [15] (all-zero)	T_{avg}	2.2 (14.6)	0.73 (4.7)	0.15 (0.83)	0.02 (0.12)	0.003 (0.03)	0.0005 (0.018)
	N_{avg}	90 (12364)	30.5 (3421)	6.5 (573)	1.66 (61)	1.04 (4.9)	1.003 (0.19)
$(127, 85)$ BCH code (all-zero)	T_{avg}	86 (–)	67 (–)	33 (–)	9 (–)	2.2 (–)	0.29 (3.5)
	N_{avg}	7617 (–)	5855 (–)	2549 (–)	655 (–)	159 (–)	19 (4132)

TABLE II
NUMERICAL RESULTS FOR MINIMUM DISTANCE COMPUTATION.

	$d_{\min}$	T^{MLD}	T^{CPLEX}	N^{MLD}	N^{CPLEX}
$(155, 64)$ Tanner code	20	137 s	3682 s	42785	21842224
$(204, 102)$ MacKay code	8	1.6 s	11.49 s	371	44830
$(408, 204)$ MacKay code	14	152 s	6893 s	9345	936570

All calculations were performed on a desktop PC with an Intel Core i5-3470 CPU (3.2 GHz) and 8 GB of RAM.

A. Maximum-Likelihood Decoding

A comparison of CPLEX and our MLD algorithm, for the different codes outlined below, is given in Table I, both in terms of T_{avg} and N_{avg}. The numbers in the parentheses are for CPLEX; a dash indicates that CPLEX was not able to decode a sufficient number of frames without running out of memory. The corresponding MLD performance curves are plotted in Fig. 1. For the curves, we have counted 100 erroneous frames for each simulation point.

The $(155, 64)$ Tanner code from [9] is often used as a benchmark code, and was also considered in [8]. For all SNRs, the ZS decoding algorithm showed a performance loss compared to the MLD curve [8]. As can be seen from Table I, our algorithm is more than 11 times as fast as CPLEX for an SNR of 1.0 dB. For higher values of the SNR our proposed algorithm is even faster compared to CPLEX. In the case of all-zero decoding, both algorithms are faster by a factor of 2 to 3, while the relative performance gain by our algorithm remains roughly the same.

The second example is a $(3, 6)$-regular $(204, 102)$ LDPC code taken from the online database of sparse graph codes from MacKay's website [15] (called 204.33.484 there). As can be seen from Table I, also for this code, our algorithm is significantly faster than CPLEX for all simulated SNRs.

In order to evaluate the performance of our algorithm for dense codes, Table I includes results for the $(127, 85)$ BCH code. CPLEX was not able to decode a significant number of frames for this code, and to our knowledge the MLD curve, as presented in Fig. 1, was previously unknown.

B. Minimum Distance Computation

In the case of computing the minimum distance, we used different values for some of the parameters, namely $M = 120$, $R = 1$, $R^{\mathrm{bb}} = 1$, and $\gamma = 0.3$. Results are shown in Table II, which additionally contains the $(408, 204)$ MacKay code (named 408.33.844 at the website [15]) that was used also in [8]. Note that in case of the $(155, 64)$ Tanner code, we can exploit the symmetry and fix $c_0 = 1$ before starting the algorithm. We compare our algorithm to CPLEX with the same formulation as for MLD and the additional constraint $\sum_{i=0}^{n-1} c_i \geq 1$ to exclude the all-zero codeword.[1]

REFERENCES

[1] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Trans. Inf. Theory*, vol. 24, no. 3, pp. 384–386, May 1978.

[2] A. Tanatmis, S. Ruzika, M. Punekar, and F. Kienle, "Numerical comparison of IP formulations as ML decoders," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Cape Town, South Africa, May 2010.

[3] "IBM ILOG CPLEX Optimization Studio," Commercial Software Package, 2013, version 12.6.

[4] E. Rosnes, Ø. Ytrehus, M. A. Ambroze, and M. Tomlinson, "Addendum to 'An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices'," *IEEE Trans. Inf. Theory*, vol. 58, no. 1, pp. 164–171, Jan. 2012.

[5] E. Rosnes and Ø. Ytrehus, "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices," *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4167–4178, Sep. 2009.

[6] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.

[7] M. H. Taghavi and P. H. Siegel, "Adaptive methods for linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5396–5410, Dec. 2008.

[8] X. Zhang and P. H. Siegel, "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes," *IEEE Trans. Inf. Theory*, vol. 58, no. 10, pp. 6581–6594, Oct. 2012.

[9] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. Int. Symp. Commun. Theory and Appl. (ISCTA)*, Ambleside, England, Jul. 2001.

[10] F. R. Kschischang, B. J. 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.

[11] 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, Sep. 1995.

[12] M. Helmling, S. Ruzika, and A. Tanatmis, "Mathematical programming decoding of binary linear codes: Theory and algorithms," *IEEE Trans. Inf. Theory*, vol. 58, no. 7, pp. 4753–4769, Jul. 2012.

[13] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," arXiv:cs/0512078 [cs.IT], Dec. 2005. [Online]. Available: http://arxiv.org/abs/cs.IT/0512078/

[14] U. Faigle, W. Kern, and G. Still, *Algorithmic Principles of Mathematical Programming*. Kluwer Academic Publishers, 2010, vol. 24.

[15] D. J. C. MacKay, encyclopedia of sparse graph codes. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[16] "GNU Linear Programming Kit (GLPK)," Software Library, version 4.52. [Online]. Available: http://www.gnu.org/software/glpk

[1] As a remark, for the $(408, 204)$ MacKay code we have used the previous CPLEX 12.5 instead of 12.6; apparently there is a bug in the latter, causing it to output a $d_{\min}$ of 20 instead of the correct value 14.