Multi-stage Decoding of LDPC Codes

Yige Wang, Jonathan Yedidia, Stark Draper

TR2009-029 July 2009

Abstract

In this paper we present a three-stage decoding strategy that combines quantized and unquantized belief propagation (BP) decoders with a mixed-integer linear programming (MILP) decoder. Each decoding stage is activated only when the preceeding stage fails to converge to a valid codeword. The faster BP decoding stages are able to correct most errors, yielding a short average decoding time. Only in the rare cases when the iterative stages fail is the slower but more powerful MILP decoder used. The MILP decoder iteratively adds binary constraints until either the maximum likelihood codeword is found or some maximum number of binary constraints has been added. Simulation results demonstrate a large improvement in the word error rate (WER) of the proposed multi-stage decoder in comparison to belief propagation. The improvement is particularly noticeable in the low crossover probability (error floor) regime. Through introduction of an accelerated "active-set" version of the quantized BP decoder we significantly speed up the pace of simulation to simulate low density parity check (LDPC) codes of length up to around 2000 down to a WER of around 10(10) on the binary symmetric channel. We demonstrate that for certain codes our approach can efficiently approach the optimal ML decoding performance for low crossover probabilities.

IEEE International Symposium on Information Theory

ISIT 2009, Seoul, Korea, June 28 - July 3, 2009

Multi-stage Decoding of LDPC Codes

Yige Wang[†] Jonathan S. Yedidia[†], Stark C. Draper[*]
[†] Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA, {yigewang, yedidia}@merl.com
[*] Dept. of ECE, University of Wisconsin, Madison, WI 53706, USA, sdraper@ece.wisc.edu

***Abstract*—In this paper we present a three-stage decoding strategy that combines quantized and un-quantized belief propagation (BP) decoders with a mixed-integer linear programming (MILP) decoder. Each decoding stage is activated only when the preceding stage fails to converge to a valid codeword. The faster BP decoding stages are able to correct most errors, yielding a short average decoding time. Only in the rare cases when the iterative stages fail is the slower but more powerful MILP decoder used. The MILP decoder iteratively adds binary constraints until either the maximum likelihood codeword is found or some maximum number of binary constraints has been added.**

Simulation results demonstrate a large improvement in the word error rate (WER) of the proposed multi-stage decoder in comparison to belief propagation. The improvement is particularly noticeable in the low crossover probability (error floor) regime. Through introduction of an accelerated "active-set" version of the quantized BP decoder we significantly speed up the pace of simulation to simulate low density parity check (LDPC) codes of length up to around 2000 down to a WER of around 10^{-10} on the binary symmetric channel. We demonstrate that for certain codes our approach can efficiently approach the optimal ML decoding performance for low crossover probabilities.

I. Introduction

Low-density parity-check (LDPC) codes [1] were first proposed by Gallager in 1960s. They have received significant attention since the 1990s due to near-Shannon limit error performance. LDPC codes are usually decoded using belief propagation (BP). Feldman et al. [2], [3] introduce an alternate decoding algorithm suitable for binary codes. By relaxing the binary constraints of maximum likelihood (ML) decoding a linear program (LP) is obtained. LP decoding has some attractive features that BP does not. An LP decoder deterministically converges and when it outputs a binary solution, it is guaranteed to be the ML solution. When the output is non-binary, a well-defined pseudo-codeword has been found.

When the LP's solution is non-binary, one is motivated to tighten the original LP relaxation. The goal is to produce a modified LP that eliminates the (formerly) optimum pseudo-codeword without eliminating any binary vertexes, hopefully yielding the ML solution. A number of proposals add additional linear constraints, e.g., redundant parity-checks (RPCs) [3], [4], [5], [6], or "lift and project" [2]. An alternate approach is to add a small number of integer (actually binary) constraints [7], [8], giving a mixed integer linear program (MILP). In [8] we use this approach to find the ML decoding performance of a (155,64) LDPC code introduced in [9].

Unfortunately, LP decoding is at first sight more complex than BP decoding. One powerful approach to reducing the computational load, that we use here and in [8], is the *adaptive* linear programming (ALP) decoder introduced by Taghavi and Siegel [4], [10]. Even with the speed-up of ALP decoding, however, LP decoding remains far slower than BP decoding.

To extend the benefits of MILP decoding to realistic block lengths and the low crossover probability (error floor) regime, one needs a very fast decoder, much faster than our previous MILP decoder. Therefore, in this paper we design a combined quantized BP, standard BP, and MILP decoder which is on average nearly as fast as a pure quantized BP decoder and performs at least as well as a pure MILP decoder. In [11] we presented initial ideas in this vein, using a first-stage BP decoder to tackle most errors, and using a second-stage MILP decoder only when the first stage fails to converge. In this paper, we take this philosophy significantly further by presenting a combined decoder that pipelines a quantized BP decoder [12], [13] (called "Algorithm E" in [12]), with standard BP and MILP decoders. The MILP decoder is not activated unless the BP decoder fails to decode, and the BP decoder is not activated unless Algorithm E fails. A different combination of LP and BP decoders wherein an initial LP decoder seeds a BP decoder is discussed in [14].

We further introduce a fast version of Algorithm E appropriate for simulations on the binary symmetric channel (BSC). In this "active-set" decoder, outgoing messages from variable and check nodes are only updated when incoming messages have been changed. Nodes whose messages require updating constitute the set of active nodes. We assume, as is legitimate for simulations of linear codes over the BSC, that the all-zeroes codeword is transmitted. When the crossover probability is small only a tiny fraction of messages from variable and check nodes will be in the incorrect non-zero state. Thus, decoding requires only a small fraction of the computation compared to an ordinary implementation of Algorithm E, and this enables us to simulate word error rates (WERs) down to nearly 10^{-10} for LDPC codes with length up to around 2000.

The rest of the paper is organized as follows. In Section II we present our multi-stage decoding architecture. In Section III, we review the constituent sub-decoders and introduce our active-set version of Algorithm E. And in Section IV, we give numerical results for several LDPC codes.

II. Multi-stage Decoding Architecture

The multi-stage decoder introduced in this paper is a combination of a (fast) Algorithm E decoder, a BP decoder, and a MILP decoder as shown in Fig. 1. When the Algorithm E decoder fails to decode, the BP decoder is activated. When BP decoding fails, the MILP decoder is activated. Our general

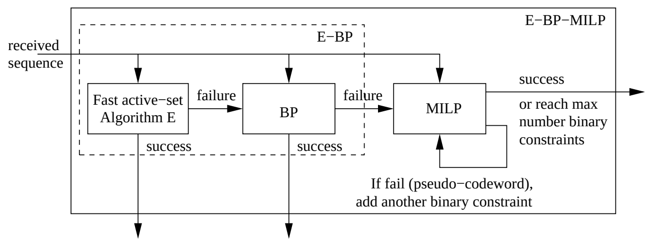
E−BP−MILP
E−BP
received sequence
Fast active−set Algorithm E
failure
BP
failure
MILP
success
or reach max number binary constraints
success
success
If fail (pseudo−codeword), add another binary constraint

Fig. 1. Structure of an E-BP-MILP decoder.

goal is for the multi-stage decoder to perform as well as a powerful but slow MILP decoder with an average throughput approaching that of the fast algorithm E decoder.

We will want to discuss the incremental decoding performance of a subset of the decoding stages. Thus, we refer to the combination of the Algorithm E and BP decoders as an E-BP decoder and to the full decoder, i.e., the combination of E-BP with MILP, as an E-BP-MILP decoder. Further, we denote an E-BP-MILP decoder using a *maximum* of t binary constraints as a E-BP-MILP(t) decoder. If $t = 0$, we refer to the full decoder as an E-BP-LP decoder, because in that case the MILP decoder is actually equivalent to an LP decoder.

We note that in our discussion "decoding failure" indicates that the specified algorithm fails to output a valid codeword. For example, the BP algorithm may fail to converge, or the LP decoder may yield a pseudo-codeword. If the algorithm outputs a valid, but incorrect codeword, this is a decoder "success", and does not trigger the use of the next stage. Of course, decoding "successes" that do not agree with the transmitted codeword contribute to the word error rate (WER).

A simple analysis can be used to approximate the average throughput of any multi-stage decoder. If we assume that a given decoder takes a processing time of T per block, has a word error rate of WER, and that nearly all errors are decoding failures, then a multi-stage E-BP-MILP decoder will have an approximate average processing time per block of

$$T_{\mathrm{E}}\left(1+\mathrm{WER}_{\mathrm{E}}\frac{T_{\mathrm{BP}}}{T_{\mathrm{E}}}+\mathrm{WER}_{\mathrm{E-BP}}\frac{T_{\mathrm{MILP}}}{T_{\mathrm{E}}}\right). \qquad (1)$$

Thus, so long as $\mathrm{WER}_{\mathrm{E}} \ll T_{\mathrm{E}}/T_{\mathrm{BP}}$ and $\mathrm{WER}_{\mathrm{E-BP}} \ll T_{\mathrm{E}}/T_{\mathrm{MILP}}$, the average throughput will be approximately the same as that of Algorithm E, even while the performance is at least as good as the MILP decoder.

III. Decoding Algorithms

In the following subsections we discuss the details of each algorithm in turn. In Section III-A we present background on Algorithm E. In Section III-B we describe our accelerated Algorithm E. In Section III-C we briefly discuss LP and MILP decoding (a detailed discussion can be found in [8]). Since the sum-product BP algorithm we use is completely standard, we do not devote space to a discussion of it.

We use the following notation. Consider a binary length-N linear code $\mathcal{C}$. A codeword $\boldsymbol{c} \in \mathcal{C}$ is transmitted over a BSC and the destination observes $\boldsymbol{y}$, where we assume binary phase shift keying wherein each 0 symbol maps to 1 and each 1 symbol maps to -1 so $y_n \in \{-1, 1\}$. Let $\boldsymbol{H} = [H_{mn}]$ be the M by N parity check matrix of an LDPC code. We denote the set of variable nodes that participate in check j by $\mathcal{N}(j) = \{k : H_{jk} = 1\}$ and the set of checks in which variable k participates as $\mathcal{Q}(k) = \{j : H_{jk} = 1\}$. We also denote using $\mathcal{N}(j)\backslash k$ the set $\mathcal{N}(j)$ with codeword symbol k excluded, and $\mathcal{Q}(k)\backslash j$ the set $\mathcal{Q}(k)$ with check j excluded.

A. Algorithm E

Algorithm E was proposed and analyzed in [12], [13]. It quantizes BP messages into -1, 0, or $+1$ values. Messages and beliefs associated with the ith iteration are denoted as

- $u_{mn}^{(i)}$: message from check node m to variable node n
- $v_{mn}^{(i)}$: message from variable node n to check node m
- $v_n^{(i)}$: belief of variable node n

For the BSC Algorithm E is carried out as [12, pp. 606-607]:

Initialization: Set $i = 1$ and the maximum number of iteration to I_{max}. For each m, n, set $v_{mn}^{(0)} = y_n$.

Step 1: For $1 \le m \le M$ and each $n \in \mathcal{N}(m)$, process

$$u_{mn}^{(i)} = \prod_{n' \in \mathcal{N}(m)\backslash n} v_{mn'}^{(i-1)}.$$

Step 2: For $1 \le n \le N$ and each $m \in \mathcal{Q}(n)$, process

$$v_{mn}^{(i)} = \mathrm{sgn}\left(w^{(i)} \cdot y_n + \sum_{m' \in \mathcal{Q}(n)\backslash m} u_{m'n}^{(i)}\right),$$

where $\mathrm{sgn}(x) = -1$ if $x < 0$, $\mathrm{sgn}(x) = 0$ if $x = 0$, and $\mathrm{sgn}(x) = 1$ if $x > 0$, and where $w^{(i)}$ is a weight chosen to optimize performance. For example, in [12], the authors show that $w^{(1)} = 2$ and $w^{(i)} = 1$ for $i \ge 2$ optimize the decoding threshold for a regular $(3, 6)$ LDPC code.

$$v_n^{(i)} = \mathrm{sgn}\left(w^{(i)} \cdot y_n + \sum_{m' \in \mathcal{Q}(n)} u_{m'n}^{(i)}\right).$$

Step 3: Create $\hat{\boldsymbol{c}}^{(i)} = [\hat{c}_n^{(i)}]$ such that $\hat{c}_n^{(i)} = 1$ if $v_n^{(i)} < 0$, $\hat{c}_n^{(i)} = 0$ if $v_n^{(i)} > 0$ and flip a coin to decide $\hat{c}_n^{(i)}$ if $v_n^{(i)} = 0$. If $\boldsymbol{H}\hat{\boldsymbol{c}}^{(i)} = \boldsymbol{0}$ or $i = I_{max}$, stop the decoding iteration and output $\hat{\boldsymbol{c}}^{(i)}$ as the decoded codeword. Otherwise, set $i := i + 1$ and return to Step 1.

B. Active-set Algorithm E

In a standard implementation of Algorithm E, all messages from check nodes and variable nodes are updated at each iteration. Updating is not needed by the nodes for which the incoming messages have not changed since the last iteration. In fact, at low crossover probabilities, most messages never need to be updated during decoding because the channel only flips a few symbols. Algorithmic complexity can thus be reduced significantly by updating messages only when necessary.

In simulations wherein one can assume that the same codeword (normally the all-zeroes codeword) is transmitted for each block, there is an important additional speed-up possible at initialization. Whenever a block is decoded successfully, the initialization step for the next block takes only $O(Np)$ time to record the positions of the newly flipped bits, where p is the crossover probability of the channel. It would otherwise necessarily take $O(N)$ time if we allowed arbitrary codewords to be transmitted as all received symbols would need to be initialized to $+1$ or -1 with equal probability. In our case, we assume that the all-zeroes codeword is transmitted.

To describe active-set Algorithm E, we define $\mathcal{A}_v$ and $\mathcal{A}_c$ as the *active* sets of variable nodes and check nodes, respectively. The active sets contain the nodes for which the outgoing messages need to be updated. The algorithm is described as:

Initialization: Set $i = 1$, $\mathcal{A}_c = \emptyset$ and the maximum number of iteration to I_{max}. For each m, n, set $v_{mn}^{(0)} = 1$. For each n, set $v_n^{(0)} = y_n$. If $y_n = -1$, set $v_{mn}^{(0)} = -1$ for all $m \in \mathcal{Q}(n)$ and add these m into $\mathcal{A}_c$.

Step 1: Set $\mathcal{A}_v = \emptyset$. For each $m \in \mathcal{A}_c$ and $n \in \mathcal{N}(m)$, process

$$u_{mn}^{(i)} = \prod_{n' \in \mathcal{N}(m) \backslash n} v_{mn'}^{(i-1)}.$$

Add all these n into $\mathcal{A}_v$.

Step 2: Set $\mathcal{A}_c = \emptyset$. For each $n \in \mathcal{A}_v$ and $m \in \mathcal{Q}(n)$, process

$$v_{mn}^{(i)} = \text{sgn}\left(w^{(i)} \cdot y_n + \sum_{m' \in \mathcal{Q}(n) \backslash m} u_{m'n}^{(i)} \right).$$

If $v_{mn}^{(i)} \neq v_{mn}^{(i-1)}$, add m into $\mathcal{A}_c$.

$$v_n^{(i)} = \text{sgn}\left(w^{(i)} \cdot y_n + \sum_{m' \in \mathcal{Q}(n)} u_{m'n}^{(i)} \right).$$

Step 3: Create $\hat{\boldsymbol{c}}^{(i)} = [\hat{c}_n^{(i)}]$ such that $\hat{c}_n^{(i)} = 1$ if $v_n^{(i)} < 0$, $\hat{c}_n^{(i)} = 0$ if $v_n^{(i)} > 0$ and flip a coin to decide $\hat{c}_n^{(i)}$ if $v_n^{(i)} = 0$. If $\boldsymbol{H}\hat{\boldsymbol{c}}^{(i)} = \mathbf{0}$ or $i = I_{max}$ is reached, stop the decoding iteration and output $\hat{\boldsymbol{c}}^{(i)}$ as the decoded codeword. Otherwise, set $i := i + 1$ and return to Step 1.

Table I shows the average number of updates required by a standard implementation of Algorithm E and the fast active-set implementation of Algorithm E for a length-1908 rate-8/9 LDPC code obtained from [15]. In Table I, "variable updates" represents the average number of variable nodes processed in Step 2 for each block and "check updates" means the average number of check nodes processed in Step 1 for each block. These statistics are obtained by averaging over 10^5 transmissions of the all-zeroes codeword.

Table I shows that the fast active-set Algorithm E becomes increasingly effective as the crossover probability decreases. This is especially useful for investigation of the error floor of a code.

C. LP and adaptive LP decoding

The derivation of LP decoding starts with the ML decoding problem:

$$\text{minimize} \qquad \gamma^T \hat{\boldsymbol{c}} \qquad \text{subject to} \qquad \hat{\boldsymbol{c}} \in \mathcal{C} \tag{2}$$

Algorithm	Stnd E	Fast E	Stnd E	Fast E
Crossover probability	0.003		0.001	
Resulting WER	0.15		0.00055	
Variable updates	17694	11924	2000	362
Check updates	1966	1518	222	15

TABLE I
COMPARISON OF THE NUMBER OF UPDATES NECESSARY FOR STANDARD AND FAST IMPLEMENTATIONS OF ALGORITHM E FOR A LENGTH-1908 RATE-8/9 LDPC CODE OBTAINED FROM [15].

where γ is the known vector of negative log-likelihood ratios defined whose nth entry is defined as

$$\gamma_n = \log\left(\frac{\Pr[y_n | c_n = 0]}{\Pr[y_n | c_n = 1]}\right).$$

When the channel is BSC, $\gamma_n = \log[p/(1-p)]$ if the received BPSK symbol $y_n = -1$, and $\gamma_n = \log[(1-p)/p]$ if the received BPSK symbol $y_n = 1$.

The variables and constraints in (2) are binary. In [3] a relaxed version of the problem is proposed. Each symbol $\hat{c}_n$ is relaxed to a corresponding variable $\hat{b}_n$ which can take values between 0 and 1. Each parity check is replaced by a number of local linear constraints that the codewords must satisfy. The intersection of these constraints defines a polytope over which the LP solver operates. The binary vertexes of the polytope correspond to codewords in $\mathcal{C}$. When the LP optimum is at such a vertex, (2) is satisfied and the ML solution is found. Non-integer solutions are termed pseudo-codewords. For a more explicit description of the linear constraints used to implement LP decoding see [3] and for more details on the adaptive LP decoder see [4] and [8].

D. Mixed integer LP (MILP) decoding

When the LP decoding yields a non-binary pseudo-codeword, the MILP decoder we use sequentially introduces integer constraints to tighten the LP relaxation. In particular, we identify the symbol whose value is closest to 0.5. For this index, $n^* = \text{argmin}_n |\hat{b}_n - 0.5|$, we add the binary constraint $\hat{b}_{n^*} \in \{0, 1\}$ and re-run the LP decoder. Since many LP solvers can accommodate integer constraints (we use GLPK [16] managed by a Python script) these constraints are easy to add. The integration of this strategy with ALP decoding is discussed in full in [8].

IV. SIMULATION RESULTS

In this section we present performance results for the decoding algorithms discussed in this paper for several LDPC codes over the binary symmetric channel (BSC). In Algorithm E and BP we set the maximum number of iterations to $I_{max} = 50$ and use weights $w^{(1)} = 2$ and $w^{(i)} = 1$ for $i \geq 2$.

A. Decoding Performance

First of all, our simulations show that a combined decoder always performs at least as well as its constituent sub-decoders. In Fig. 2 we plot the WER of the length-1057 rate-0.77 LDPC code obtained from [15]. Notice that standard BP decoding and E-BP have nearly the same performance. This means that for this code using the fast active-set Algorithm

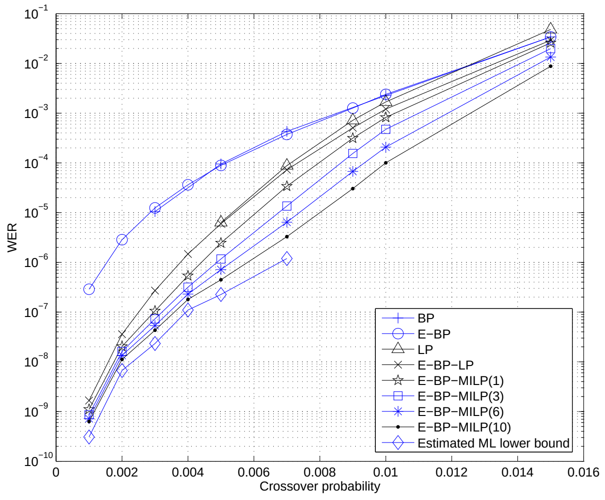

Fig. 2. WER performance of a length-1057 rate-0.77 LDPC code obtained from [15] using BP, LP, E-BP, and E-BP-MILP decoding algorithms.

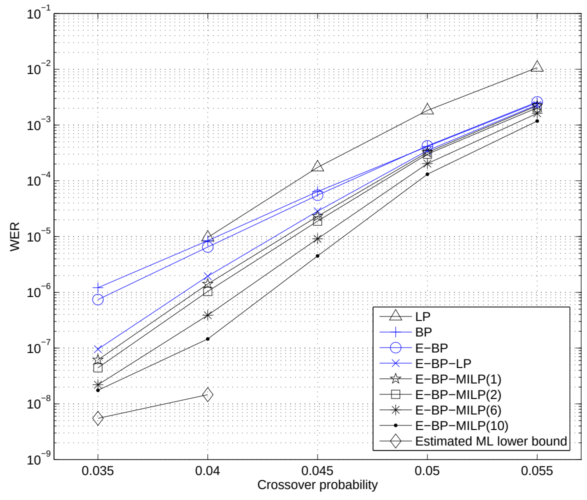

Fig. 3. WER performance of a length-1056 rate-1/2 QC-LDPC code using LP, BP, E-BP and E-BP-MILP decoding algorithms.

E as a first stage decoder prior to BP does not significantly change the overall decoding performance, while it speeds up simulations considerably. Similarly, E-BP-LP performs slightly better than standard LP decoding. As we shall see below, for other codes the combined E-BP-LP decoder has a more significant performance improvement compared to standard LP decoding, while also being faster.

In Fig. 2 we also plot the performance of E-BP-MILP(t) decoder as a function of the maximum number of MILP binary constraints t and compare to a lower bound on the ML performance for comparison. We find that for $t = 10$, the performance of the E-BP-MILP(10) decoder is close to the ML lower bound, and at low crossover probabilities, the performance of the E-BP-MILP(t) is quite close to the estimate even for smaller values of t. Our lower bound on ML decoding performance is estimated by counting the number of non-all-zeroes codewords decoded by E-BP-MILP that have higher likelihood than the transmitted all-zeroes codeword, and dividing by the total number of processed blocks.

Figure 3 depicts the WER performance of a rate-1/2 girth-10 regular (3,6) quasi-cyclic (QC) LDPC code of block-length 1056 constructed using the hill-climbing girth-maximizing algorithm described in [17]. For this code, we found 33 ML errors at crossover probability 0.035. This is sufficient to estimate a lower bound on ML decoding performance and provides an upper bound of 24 on the minimum distance of the code. Again the WER of E-BP-MILP(10) approaches the optimal (ML) performance for low crossover probabilities.

In Fig. 3 we also observe that E-BP slightly out-performs standard BP, while E-BP-LP rather significantly out-performs pure LP decoding. It appears that, for this code, E-BP-LP decoding performs better because the set of blocks that cause the BP and the LP failures are, to some extent, disjoint sets.

Figure 4 plots the WER of a length-1908 rate-8/9 LDPC code from [15]. For this code BP and E-BP again have similar

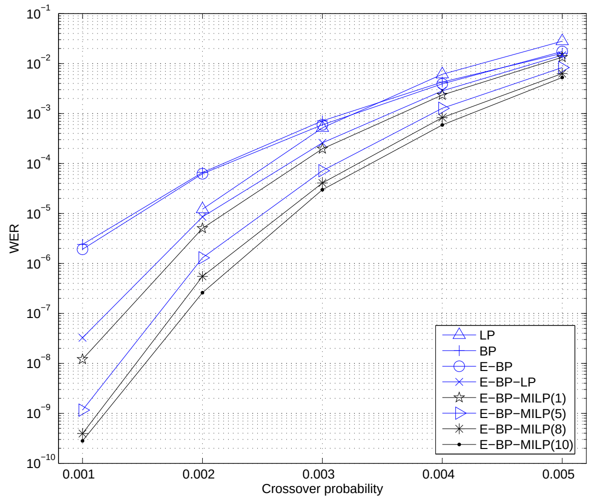

Fig. 4. WER performance of a length-1908 rate-8/9 LDPC code obtained from [15] using BP, LP, E-BP, and E-BP-MILP decoding algorithms.

performance. LP is slightly worse than BP at high, but better at low, crossover probabilities. No ML errors were observed during simulation. In addition, E-BP-MILP(t) decoding performance improves consistently as t increases. At a crossover probability of 0.001, the WER of E-BP-MILP(10) is four orders of magnitude lower than BP, and our simulations are fast enough to estimate the WER down to nearly 10^{-10}.

Figure 5 shows the WER performance of a length-1000 rate-1/2 random LDPC code. This code was intentionally designed to have a prominent error floor (girth-6 cycles that give rise to trapping sets were introduced), but its minimum distance appears to have remained large as no undetected errors were found. For this code the LP and combined decoders improve performance very significantly at low crossover probabilities where the BP decoder suffers from an error floor.

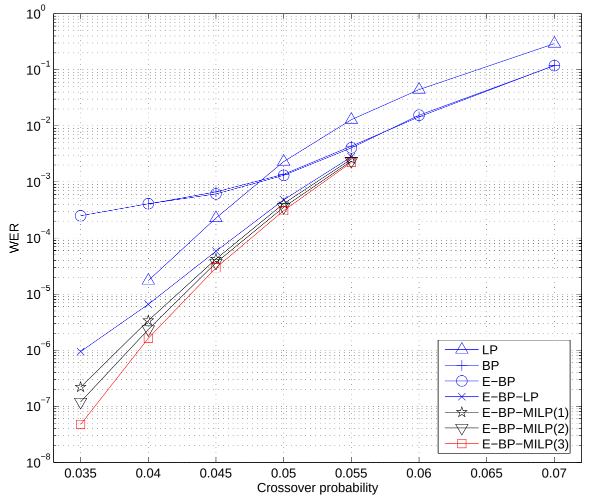

Fig. 5. WER performance of a length-1000 rate-1/2 random LDPC code using BP, E-BP, and E-BP-MILP decoding algorithms.

N_{bin}	1	2	3	4	5	6	7	8	9	10
	3.86	9.47	19.2	35.5	57.6	93.8	164.6	230.6	363.4	617.4

TABLE II
MILP PROCESSING TIME PER BLOCK IN SECONDS. N_{bin} IS THE NUMBER OF BINARY CONSTRAINTS REQUIRED TO DECODE SUCCESSFULLY.

B. Processing Time

The above simulations show that typically the E-BP-MILP decoder significantly out-performs a standard BP decoder. In this sub-section, we discuss the processing times for our decoders. The most important point is that, as is suggested by the analysis given in equation (1), the average processing time of the E-BP-MILP decoder is indeed often only slightly longer than that of its fastest component (the active-set Algorithm E decoder), although the worst case for a single block is controlled by the processing time of its slowest component (the MILP decoder).

In our implementations, the E-BP and standard BP decoders are programmed in C and the MILP subroutine is programmed in Python, calling the C-language GLPK [16] linear programming library. All the processing time statistics are obtained using the same machine (a mini-mac).

To illustrate, we take data from the simulation of the length-1908 LDPC code. As mentioned, the *worst-case* processing time per block is dominated by the processing time of the MILP decoder. This worst-case time grows rapidly with the number of binary constraints required to decode as shown in Table II. There we tabulate for the length-1908 code at crossover probability of 0.002, the average MILP decoder processing time for blocks that required a given number N_{bin} of binary constraints before MILP decoding is successful.

On the other hand, the average processing time of E-BP-MILP(10) is nearly the same as active-set Algorithm E. In Table III we tabulate the processing times for the variety of decoders discussed in this paper. As a sample calculation consider the BSC with crossover probability 0.001. The average processing time per block using E-BP is 0.000337 seconds. Since the WER of E-BP at crossover probability 0.001 is 2×10^{-6}, and assuming each of these errors is detectable, on average two blocks out of each million blocks will be fed to the MILP decoder. Through simulation we find that the average processing time per block of the MILP(10) decoder is 0.65 seconds. Thus, the average processing time per block using E-BP-MILP is $(2 \times 0.65 + 10^6 \times 0.000337)/10^6 = 0.000338$ seconds. Hence, at low enough crossover probabilities the average throughput of the E-BP-MILP(10) will approach that of Algorithm E.

Crossover	fast-E	BP	LP	MILP(10)	E-BP	E-BP-MILP(10)
0.001	0.000332	0.0095	0.16	0.65	0.000337	0.000338
0.002	0.0011	0.118	0.80	8.68	0.0014	0.0019

TABLE III
AVERAGE PROCESSING TIME PER BLOCK IN SECONDS FOR FAST ALGORITHM E, BP, LP, PURE MILP(10), E-BP, AND E-BP-MILP(10).

REFERENCES

[1] R. G. Gallager, *Information Theory and Reliable Communication*. John Wiley and Sons, 1968.

[2] J. Feldman, "Decoding error-correcting codes via linear programming," Ph.D. dissertation, Mass. Instit. of Tech., 2003.

[3] J. Feldman, M. J. Wainwright, and D. Karger, "Using linear programming to decoding binary linear codes," *IEEE Trans. Inform. Theory*, vol. 51, pp. 954–972, Mar. 2005.

[4] M.-H. N. Taghavi and P. H. Siegel, "Adaptive linear programming decoding," in *Proc. Int. Symp. Inform. Theory*, Seattle, USA, July 2006, pp. 1374–1378.

[5] A. Tanatmis, S. Ruzika, H. Hamacher, M. Punekar, F. Kienle, and N. Wehn, "A separation algorithm for improved LP-decoding of linear block codes," in *Proc. Int. Symp. Turbo Codes and Related Topics*, Lausanne, Switzerland, Sept. 2008.

[6] M. Miwa, T. Wadayama, and I. Takumi, "A cutting plane method based on redundant rows for improving fractional distance," in *Proc. Int. Symp. Turbo Codes and Related Topics*, Lausanne, Switzerland, Sept. 2008.

[7] K. Yang, J. Feldman, and X. Wang, "Nonlinear programming approaches to decoding low-density parity-check codes," *IEEE J. Select. Areas Commun.*, vol. 24, pp. 1603–1613, Aug. 2006.

[8] S. C. Draper, J. S. Yedidia, and Y. Wang, "ML decoding via mixed-integer adaptive linear programming decoding," in *Proc. Int. Symp. Inform. Theory*, Nice, France, July 2007.

[9] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. ICSTA*, Ambleside, UK, 2001.

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

[11] S. C. Draper and J. S. Yedidia, "Complexity scaling of mixed-integer linear programming decoding," in *UCSD Workshop Inform. Theory Apps.*, San Diego, Jan. 2008.

[12] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[13] M. Mitzenmacher, "A note on low density parity check codes for erasures and errors," SRC Technical Note, Tech. Rep. 1998-017, 1998.

[14] M. Lunglmayr, J. Berkmann, and M. Huemer, "Combined linear programming/belief propagation decoder," *Electron. Lett.*, vol. 44, no. 12, pp. 751–752, June 2008.

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

[16] "GNU Linear Programming Kit," http://www.gnu.org/software/glpk.

[17] Y. Wang, J. S. Yedidia, and S. C. Draper, "Construction of high-girth QC-LDPC codes," in *Proc. Int. Symp. Turbo Codes and Related Topics*, Lausanne, Switzerland, Sept. 2008.