Improved Linear Programming Decoding using Frustrated Cycles

Shrinivas Kudekar
Qualcomm Research
Bridgewater, NJ, USA.
Email: skudekar@qti.qualcomm.com

Jason K. Johnson
Numerica Corporation
Loveland, CO, USA.
Email: jasonj@lanl.gov

Misha Chertkov
Center for Nonlinear Studies & Theoretical Division T-4
Los Alamos National Laboratory, Los Alamos, NM, USA
Email: chertkov@lanl.gov

***Abstract*—We consider data transmission over a binary-input additive white Gaussian noise channel using low-density parity-check codes. One of the most popular techniques for decoding low-density parity-check codes is the linear programming decoder. In general, the linear programming decoder is suboptimal. In this paper we present a systematic approach to enhance the linear programming decoder. More precisely, in the cases where the linear program outputs a fractional solution, we give a simple algorithm to identify *frustrated cycles* which cause the output of the linear program to be fractional. Then adding these cycles, *adaptively* to the basic linear program, we show *improved word error rate performance*.**

I. Introduction

We consider data transmission over a binary-input additive white Gaussian noise channel (BIAWGNC) using low-density parity-check (LDPC) codes. The two most fundamental decoders in this context are the belief propagation (BP) decoder [1] and the linear programming (LP) decoder [2]. In this paper we are interested in the performance of the LP decoder. There is an extensive literature on the analysis and design of the LP decoder for LDPC codes [2]–[7]. As is well known, LP decoders have the advantage that they provide the *ML certificate*. This means that, if the LP decoder outputs an integer solution, then it must be the maximum likelihood (ML) codeword. Thus in this case the LP behaves as an optimal decoder. One can also say that in this case there is no *duality gap*.

However, it is also known that in general the LP decoder is suboptimal [2]. I.e., there exists channel noise realizations such that the LP decoder outputs a fractional solution, known as *pseudocodewords* [4], but still there exists a unique codeword which *minimizes* the objective function. This implies that the LP decoder is not successful in finding the ML codeword. As a result, there is a gap between the performance of the LP decoder and the ML decoder. Hence it is an interesting question to understand what causes the LP decoder to fail and further if there exists methods to *improve* the LP decoder. It is well known that adding redundant parity-check nodes to the Tanner graph of the LDPC code improves the LP decoder [2], [8]. However it is not desirable to add all such constraints as it will slow down the LP decoder considerably.

In this work we propose an approach to adaptively add constraints to the LP decoder which, simultaneously, reduce the duality gap and are tractable (i.e., the number of such additional constraints are small and also each constraint involves only a small number of variables). Such approaches, which try to get rid of the fractional solution (or make the LP polytope tighter), have been used to improve the LP decoding of LDPC codes [2], [8]–[13]. The new LP decoder which we propose, identifies *frustrated cycles* (see Section III-A) when the basic LP produces a fractional solution. We show that these frustrated cycles are the cause of inconsistency in the solution. Then we adaptively add them, as constraints, to the basic LP decoder. This enables us to recover the transmitted codeword in many cases. We show empirically that the new LP decoder has an improved word error rate performance. Furthermore, the new LP decoder also has tractable complexity.

II. Channel Model, Maximum Likelihood Decoder and Linear Programming decoder

A. Setup and Nomenclature

We consider data transmission over a BIAWGNC with noise distribution given by $\mathcal{N}(0, \sigma^2)$. We use LDPC code with blocklength n and denote $\underline{x} = \{x_1, x_2, \dots, x_n\}$ as the transmitted codeword. The input codebit takes value in $\{0, 1\}$. The received message is denoted by $\underline{y} \in \mathbb{R}^n$. We will use the loglikelihood ratio (LLR) to represent the channel observations. More precisely, we have $l_i = \log \frac{p_{y|x}(y_i \mid 0)}{p_{y|x}(y_i \mid 1)}$, where $p_{y|x}(y|x)$ is the channel transition pdf. Let $\underline{l}$ represent the vector of LLRs.

The LDPC code is represented by the usual Tanner graph representation [1]. Throughout the paper we will use (d_l, d_r)-regular LDPC code ensembles to demonstrate our approach. The design rate of the LDPC code is given by $1 - d_l/d_r$. In the experiments we perform later, we consider the random $(3, 4)$-regular LDPC code ensemble and the fixed 155-Tanner code [14] which has degree-3 variable nodes and degree-5 check nodes. We use V to denote the set of n variable nodes or codebits and C to denote the set of m parity check nodes. A generic variable node and a check node is denoted by the letter i and c respectively. Let $\mathcal{C}$ represent the code (or the set of codewords).

B. ML Decoder

The ML decoder can be written as the following combinatorial optimization problem [2], $\min_{\underline{x} \in \mathcal{C}} \sum_{i=1}^{n} l_i x_i$. This is also the Integer Program (IP) representing ML decoding.

C. Basic Linear Programming Decoder

For every check node $c \in C$, let $x_c = \{x_i \,|\, i \in c\}$. We also use $c \setminus i$ to denote the set of all variable nodes contained in check node c except for the variable node i. The above IP can be relaxed to

$$\begin{aligned}
&\min_{\underline{b}} \sum_{i=1}^{n} \sum_{x_i \in \{0,1\}} l_i x_i b_i(x_i) \\
&\text{s.t.} \quad \forall i \in V: \quad \sum_{x_i \in \{0,1\}} b_i(x_i) = 1, \\
&\forall c \in C, \forall i \in c, \, x_i \in \{0,1\}: \quad b_i(x_i) = \sum_{x_{c\setminus i}} b_c(x_i, x_{c\setminus i}) \\
&\forall c \in C, \forall x_c \text{ s.t.} \sum_{i \in c} x_i = 1, \; b_c(x_c) = 0, \text{ (local codeword)} \\
&0 \le b_i(x_i) \le 1, \; \forall i \in V, \quad 0 \le b_c(x_c) \le 1, \; \forall c \in C,
\end{aligned}$$

which constitutes the standard LP decoder [2]. Here $b_i(x_i)$ represents the "belief" of the variable node i and $b_c(x_c)$ represents the "belief" associated to the check node c. In the sequel, we will also say that $b_i(x_i)$ is the belief associated to the singleton clique i and $b_c(x_c)$ is the belief associated to a higher order *clique*[1]. Also, $\underline{b}$ represents the vector of all the variable node and check node beliefs. Note that the objective function represents the "cost" of decoding a bit to 0. This cost is reduced if the corresponding LLR is negative. The second condition imposed by the LP above is the *consistency* condition. In the third condition, the sum is over GF(2).

III. Main Results: Improved LP decoding

As mentioned earlier, our approach is to adaptively add constraints to the LP which decrease the duality gap. Furthermore, the number of additional constraints should be small and each constraint should involve only a small number of variables.

There are many existing approaches to improve the LP decoder [2], [8]–[13]. In [2] an improved LP decoder based on "lift-and-project" method was introduced. In [10], the LP is enhanced by eliminating the facet containing the fractional solution. In [8], [9], extra constraints are added by combining parity checks which correspond to violated constraints to improve the LP performance. In [11] a mixed-integer LP was introduced by fixing the most "uncertain" bit of the pseudocodeword. In [12] an adaptive LP decoder was introduced based on *loop calculus*. Critical loops were identified and then broken by fixing bits on the loop. In [13] a non-linear programming decoder was designed for decoding LDPC codes.

A. LP Decoders using Frustrated Subgraphs

Although our approach is in the same spirit as aforementioned works, the main ideas are different and originate in [15] and [16]. Similar ideas have been independently used in [17], [18]. We begin with the notion of a *frustrated graph.*

[1]In a *clique*, every node is connected to every other node. The LPs given in this paper always have beliefs associated to cliques.

Definition 1 (Frustrated Graph): Consider a constraint satisfaction problem (CSP) defined on n binary (boolean) variables, $\underline{x}$, and m constraint nodes (each of which constraints a small set of variables). For each constraint c there are only certain configurations of $x_c \in \{0,1\}^{|c|}$ which satisfy it. Then, we say that the graph is *frustrated* if and only if there is no assignment of $\underline{x}$ which satisfies all m constraint nodes simultaneously. ■

Let us now define a CSP for our set-up.

Definition 2 (CSP obtained by the LP Solution): Assume that the output of LP, $\underline{b}$, is a fractional solution, i.e., we have a duality gap. For every clique c (with size at least two), the set of x_c which *satisfy* the clique are those for which $b_c(x_c) > 0$. In other words, the set of x_c satisfying the clique c, correspond to the support set of $b_c(x_c)$. Consequently, the CSP is given by the n (binary) variables, $\{x_i\}_{i=1}^n$ and the set of cliques c (constraining the variables as described previously). ■

We now show that if the output of the LP has a frustrated subgraph, then it must have a duality gap, i.e., the solution must be fractional.

Lemma 3: If there exists a frustrated subgraph, then there is a duality gap.

Proof: Indeed, suppose on the contrary there was no duality gap, i.e, the output of the LP is integral. Thus for every clique c (singleton or higher order), $b_c(x_c) = 1$ for some $x_c \in \{0,1\}^{|c|}$ and $b_c(x_c) = 0$ for the rest. Consider any subset of the cliques, $\mathcal{C} = \{c_1, c_2, \ldots, c_r\}$. Let $x_{c_i}^*$ be such that $b_{c_i}(x_{c_i}^*) = 1.0$. We claim that $\cup_{i=1}^r x_{c_i}^*$ satisfies the CSP represented by $\mathcal{C}$. Indeed, this follows from the consistency imposed by the LP. Thus no subgraph is frustrated. ■

Thus our strategy is as follows: first identify a frustrated subgraph from the output of the basic LP; if we add (as detailed below) this frustrated subgraph as a constraint in our LP, then we ensure that this subgraph cannot be frustrated. In our experiments we see that, in many cases, adding the frustrated subgraphs eliminates the duality gap. To ensure that the subgraph we add as a constraint to the LP becomes consistent (or is not frustrated), we need to add all its maximal cliques and their intersections to the LP. More precisely, we add the maximal cliques of the junction tree[2] of that subgraph as extra beliefs to the LP.

The main challenge that remains is to find a frustrated subgraph in tractable time. In general, it is hard to find an arbitrary subgraph which is frustrated. We also remark that in [15] it was found empirically that the random field ising model could typically be solved (duality gap eliminated) by adding frustrated cycles arising in the LP solution. It is also known from Barahona's work (see references within [17]) that adding cycles is sufficient to solve the zero-field planar ising model. Hence as a first step, we focus on finding *frustrated cycles* of the graph. Frustrated cycles and a procedure to find them are

[2]See [15] for a discussion on Junction trees. It can be shown that running LP on the junction tree of a graph is optimal (equal to the original combinatorial optimization problem). If frustrated subgraph is a cycle then we just add all the triangles which chordalizes the cycle.

described in the next section. The procedure is tractable and uses the implication graph method (to solve 2SAT problem) of [15], [19]. For details see Appendix B in [15].

B. Implication Graph and Frustrated Cycles

For every clique c, consider all the two-projections of its belief. I.e., for every $b_c(x_c)$, consider all the $b_{ij}(x_i, x_j)\ \forall\ i, j \in c$. These are obtained by summing out the other variables. We construct the implication graph as follows. In the implication graph each node i is present as i_+ (for $x_i = 0$) and i_- (for $x_i = 1$). Thus, the implication graph has a total of $2n$ nodes. There is a directed edge present between i and j which represents the logical implication obtained from $b_{ij}(x_i, x_j)$. Let us explain this in more details. To generate the logical implication, consider the set T of configurations of (x_i, x_j) which render $b_{ij}(x_i, x_j) > 0$ and can introduce inconsistency. Thus, T is any of the following $(01, 10), (01, 10, 11), (01, 10, 00), (00, 11), (00, 11, 10)$ and $(00, 11, 01)$. Indeed, moments thought shows that other configurations, e.g., $(00, 01, 10, 11)$, are not restrictive and hence do not form any logical implication. Also, nodes which have integer beliefs are present as isolated nodes in the graph and do not have any edges entering or leaving it. Draw the directed edges using this T. E.g., suppose that LP outputs beliefs such that $b_{ij}(0, 1) > 0, b_{ij}(1, 0) > 0, b_{ij}(1, 1) > 0, b_{i,j}(0, 0) = 0$ then $T = (01, 10, 11)$. This implies a directed edge from $i_+ \to j_-$ and $j_+ \to i_-$, because if $x_i = 0$ then we must have $x_j = 1$ and if $x_j = 0$ then $x_i = 1$. Figure 1 shows the basic building blocks for constructing the implication graph.

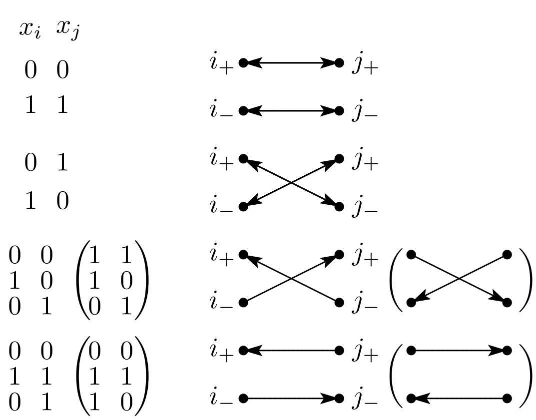

Fig. 1. All implications between x_i, x_j used to create the implication graph.

Finally, a frustrated cycle is defined to be a *directed cycle or a directed path* which visits both i_+ and i_-, once, for any i. One can find all such cycles and paths in a time which is linear in the number of nodes of the implication graph.

Figure 2 shows the possible frustrated cycles which are obtained from the implication graph. The figure on the left shows true frustration. I.e., from the logical implications, obtained by the LP solution, we have that $x_i = 0$ implies $x_i = 1$ and $x_i = 1$ implies $x_i = 0$. This means that the set of local beliefs (which lie on the cycle connecting i_+ to i_-) are not consistent. Hence it naturally suggests that there is frustration in the LP solution. The other kind of frustration, suggested by the remaining figures, is called as *quasi-frustration*. The figure in the middle demonstrates that $x_i = 1$ implies that $x_i = 0$ but not the other way around. This quasi-frustration implies that there cannot be a global joint distribution (on all the variable nodes) such that it is consistent with the local beliefs. Indeed, if it were true, then we know that it must assign $b_i(x_i = 0) > 0$ and $b_i(x_i = 1) > 0$. This is because the variable node i is present in the implication graph and hence must have a fractional solution for $b_i(x_i)$. However, from the implication graph $x_i = 1$ implies $x_i = 0$, hence any configuration (on all nodes), which has a non-zero probability, cannot have $x_i = 1$, i.e., $b_i(x_1 = 1) = 0$, a contradiction. We

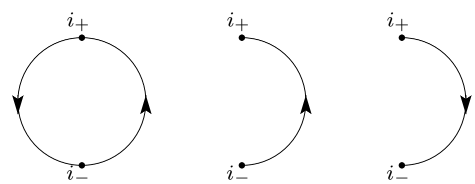

Fig. 2. Figure shows the possible frustrated cycles present in the implication graph. The first cycle is truly frustrated, since we must have $x_i = 0$ implies $x_i = 1$ and vice-versa. The remaining two cycles are quasi-frustrated, since either $x_i = 0$ implies $x_i = 1$ or vice-versa, but not both at the same time. These cycles are added to the LP and the enhanced decoder is termed LP-Frustrated Cycles (LP-FC).

remark here that once we have found a frustrated cycle on the implication graph, one can easily obtain the cycle on the original graph, by just projecting the nodes on the implication graph back to the nodes on the original graph. The method in which we add the frustrated cycle to the LP is illustrated in the example below.

Example 4 (Triangulation of Frustrated Cycles): Figure 3 shows a cycle $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ which we add to the LP as a constraint. Adding the entire belief, $b(x_1, x_2, \ldots, x_8)$, as a constraint, would be expensive and result in 2^8 extra variables and constraints amongst them. Instead we add the maximal cliques of its junction tree. To do this, we first chordalize or triangulate the cycle, as shown in the figure 3, into the 6 triangles given by $(x_1, x_2, x_3), (x_1, x_3, x_4), (x_1, x_4, x_5), (x_1, x_5, x_6)$, (x_1, x_6, x_7), (x_1, x_7, x_8). These triangles are the maximal cliques and we add them as constraints to the LP. E.g., we add $b_{x_1 x_2 x_3}(x_1, x_2, x_3)$ for all $x_1, x_2, x_3 \in \{0, 1\}$. For every belief that we add to the LP, we add constraints to ensure consistency with previously added beliefs. E.g., when we add $b_{x_1,x_2,x_3}(x_1, x_2, x_3)$ and $b_{x_1,x_3,x_4}(x_1, x_3, x_4)$ we introduce the constraint $\sum_{x_2} b_{x_1,x_2,x_3}(x_1, x_3) = \sum_{x_4} b_{x_1,x_3,x_4}(x_1, x_3, x_4)$ for all values of x_1, x_3. In other words, every clique that we add to the LP, must be consistent across its intersections.

C. Experiments using Frustrated Cycles

We consider BIAWGNC where the standard deviation of the noise is denoted by σ. We consider two types of

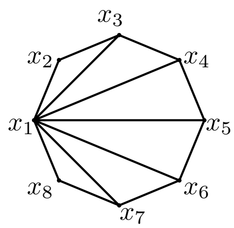

Fig. 3. The triangles chordalize the cycle and form the maximal cliques.

LDPC encoding: (i) regular $(3,4)$ LDPC code ensemble with design rate equal to $1/4$ and the (ii) 155-Tanner code [14]. The 155-Tanner code has a design rate of $2/5$. We let the standard deviation of the noise, σ, take values in the set $\{0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.00, 1.05, 1.10, 1.15, 1.20\}$. We run 2000 trials for each value of σ. We run experiments for both the $(3,4)$-regular ensemble and the 155-Tanner code. For the $(3,4)$-regular ensemble, in each trial a code is generated uniformly at random and used for transmission.

Since we are transmitting over a symmetric channel we assume wlog that we are transmitting the all-zero codeword [1]. Under this assumption, the distribution of the LLRs are given by $\mathcal{N}(\frac{2}{\sigma^2}, \frac{4}{\sigma^2})$. The generated LLRs are fed to both basic LP and LP-FC decoder. The LP-FC algorithm is described below. For any decoder, if the output equals the all-zero codeword, we declare success, else there is an error. We plot the word error rate (WER) versus the SNR (in dB).

LP-FC Decoder:

1) Run the basic LP[a]. Go to step 4.
2) If the output is fractional, find the shortest frustrated cycle (FC) and add all its triangles.
3) Rerun the LP.
4) If output is integral, stop, else go to 2.

[a] We use the GNU linear programming kit to solve the LP.

1) Experiments with $(3,4)$-regular LDPC code ensemble: Figure 4 shows the performance curve when we use the $(3,4)$-regular ensemble with blocklength 160. The dark curve represents the performance (averaged over 2000 trials where in each trial a code and noise realization is picked uniformly at random) when we use the basic LP decoder. The gray curve denotes the performance under LP-FC. We remark here that for each simulation trial, the LP and LP-FC were run on the same code and noise realization. We observe that there were many trials where the basic LP decoder failed. However, adding a small number of cycles to the LP helped in retrieving the transmitted all-zero codeword. From the figures we observe that LP-FC performs better than the basic LP.

Table I demonstrates various quantities for different values of the SNR for the case when we use the $(3,4)$-regular LDPC code ensemble with blocklength 160. The second column shows the average number of LPs called in the LP-FC algorithm, i.e, the number of times step 3 is called in the LP-

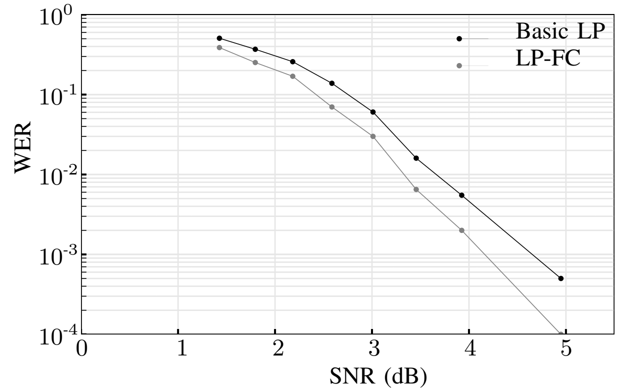

Fig. 4. The figure shows the performance improvement of LP-FC over the basic LP. In this experiment $(3,4)$-regular LDPC code ensemble of blocklength 160 was used. The dark curve depicts the word error rate (WER) performance of the basic LP and the gray curve shows the performance of the LP-FC.

SNR (in dB)	Num. of LPs (avg.)	Non-zeros (avg.) LP ($\times 10^4$)	Non-zeros (avg.) LP-FC ($\times 10^4$)	Non-zeros (max) LP-FC ($\times 10^4$)	Dim. for LP (rows,cols)	Dim. (avg.) for LP-FC (rows,cols)
3.93	3	2.6760	2.7028	2.9252	(6490,4920)	(6569,4970)
3.46	6	2.6597	2.7190	3.1780	(6458,4896)	(6635,5009)
3.01	7	2.6694	2.7471	3.8020	(6477,4910)	(6708,5059)
2.59	6	2.6613	2.7312	3.6116	(6461,4899)	(6669,5032)
2.18	7	2.6637	2.7514	4.1012	(6466,4902)	(6727,5070)
1.79	7	2.6659	2.7403	3.7796	(6470,4905)	(6692,5047)
1.43	8	2.6572	2.7483	3.8984	(6453,4893)	(6725,5068)

TABLE I
COMPLEXITY COMPARISON OF LP AND LP-FC DECODERS.

FC algorithm. The remaining columns illustrate the complexity of the extra LPs which are solved in the LP-FC algorithm. The third and the sixth column show the number of non-zeros in the constraint matrix and the dimensions of the constraint matrix when the basic LP is run. The fourth and the last column show the average number of non-zero entries in the constraint matrix and the average dimensions of the of the constraint matrix, when the LP-FC algorithm is run, respectively. Also shown in the fifth column is the maximum number of non-zero entries in any constraint matrix which occurs in the LP-FC algorithm. Thus, the table demonstrates that the size of the LP, after adding the frustrated cycles, does not increase by much. Hence the LP-FC decoder is kept tractable.

We also observe that every cycle we add is a *simple* cycle, without any self-intersections.

2) Experiments with 155-Tanner code [14]: We also perform experiments with the 155-Tanner code which has 155 variable nodes and 93 check nodes. The experimental set-up is same as before.

Figure 5 shows the performance curve (averaged over 2000 noise realizations for each value of σ) when we use the 155-Tanner code. Again, we observe that LP-FC performs better than the basic LP.

We also perform experiments at very high SNR for the 155-Tanner code. This is known as the error-floor regime. The

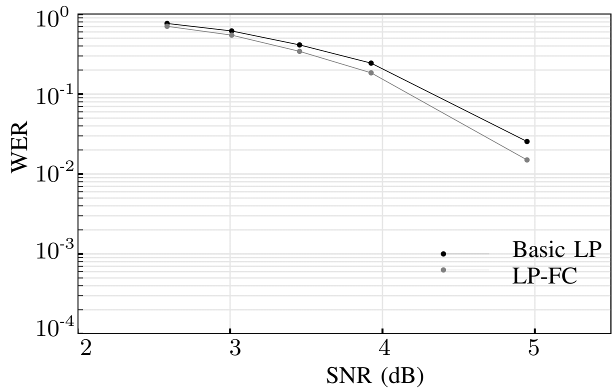

Fig. 5. The figure shows the performance improvement of LP-FC over the basic LP when the 155-Tanner code was used.

error-floor occurs because of low-weight pseudocodewords which are fractional, i.e., not codewords. In [20] a pseudocodeword search algorithm was used to generate pseudocodewords which are not codewords. We pick 200 worst pseudocodewords which have effective weight [4] less than the minimum Hamming distance of 20. Also, all these pseudocodewords will dominate the WER when SNR becomes very large.

The experiment we perform is as follows. We take the corresponding noise realizations which gave rise to these 200 pseudocodewords. We run the basic LP on them and confirm that it fails on all these noise realizations and indeed we recover the fractional pseudocodewords. On the same noise realizations, we also run the LP-FC. Remarkably, the LP-FC is able to recover the correct (all-zero) codeword for all the 200 worst-case noise realizations. Furthermore, the step 3 in the LP-FC algorithm was just called once. The constraint matrix for the basic LP has 51,646 non-zeros entries and a dimension of (8618, 7006). On the other hand, the enhanced LP has on an average, 52,676 non-zeros entries and an average dimension of (8925, 7163). Again, the LP-FC is kept tractable.

IV. Discussion

In this work we show that the presence of frustration in the output of the basic LP solution is the cause of inconsistency. We add these frustrated cycles as constraints to the LP, thus enhancing it. We observe empirically that the LP-FC decoder eliminates the duality gap, in a large number of cases and has a better performance when compared to the basic LP. One future research direction is to investigate if one can add a *frustrated subgraph*, which is not a cycle, to enhance the LP, when the addition of cycles is not enough to eliminate the duality gap. The reason we choose to add frustrated cycles, is that the algorithm for finding such cycles is simple. It is not clear if there exists simple algorithms to find minimal frustrated subgraphs.

Recently, improved LP detectors based on frustrated cycles was also used in [21] for 2DISI channel. It would be interesting to investigate other combinatorial problems in graphical coding, e.g., minimum pseudocodeword weight problem, minimum Hamming distance, etc. Another future direction would be to develop message-passing versions for the LP-FC.

V. Acknowledgments

Our work at LANL was carried out under the auspices of the National Nuclear Security Administration of the U.S. Department of Energy at Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396. SK acknowledges support of NMC via the NSF collaborative grant CCF-0829945 on "Harnessing Statistical Physics for Computing and Communications."

References

[1] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[2] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inform. Theory*, vol. 51, no. 3, Mar. 2005.

[3] J. Feldman, "Decoding error-correcting codes via linear programming," 2003, phD dissertation.

[4] R. Kötter and P. O. Vontobel, "Graph covers and iterative decoding of finite-length codes," in *Proc. of the Int. Conf. on Turbo Codes and Related Topics*, Brest, France, Sept. 1–5, 2003, pp. 75–82.

[5] C. Daskalakis, A. G. Dimakis, R. M. Karp, and M. J. Wainwright, "Probabilistic analysis of linear programming decoding," *IEEE Trans. Inform. Theory*, vol. 54, no. 8, pp. 3565–3578, Aug. 2008.

[6] X. Zhang and P. Siegel, "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes," *IEEE Trans. Inform. Theory*, vol. 58, no. 10, Oct. 2012.

[7] M. Helmling, S. Ruzika, and A. Tanatmis, "Mathematical programming decoding of binary linear codes: theory and algorithms," *IEEE Trans. Inform. Theory*, vol. 58, no. 7, July 2012.

[8] D. Burshtein and I. Goldenberg, "Improved linear programming decoding and bounds on the minimum distance of LDPC codes," in *Proc. of the IEEE Inform. Theory Workshop*, Dublin, Ireland, Aug. 2010.

[9] M.-H. Taghavi and P. Siegel, "Adaptive methods for linear programming decoding," *IEEE Trans. Inform. Theory*, vol. 54, no. 12, pp. 5396–5410, 2006.

[10] A. Dimakis, A. Gohari, and M. Wainwright, "Guessing facets: Polytope structure and improved lp decoder," *IEEE Trans. Inform. Theory*, vol. 55, no. 8, pp. 3479–3487, 2009.

[11] S. Draper, J. Yedidia, and Y. Wang, "Ml decoding via mixed-integer adaptive linear programming," in *Proc. of the IEEE Int. Symposium on Inform. Theory*, June 2007, pp. 1656–1660.

[12] M. Chertkov, "Reducing the error floor," in *Proc. of the IEEE Inform. Theory Workshop*, Sept. 2007, pp. 230–235.

[13] K. Yang, J. Feldman, and X. Wang, "Nonlinear programming approaches to decoding low-density parity-check codes," *IEEE J. Sel. Area. Commun.*, vol. 24, no. 8, pp. 1603–1613, Aug. 2006.

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

[15] J. Johnson, "Convex relaxation methods for graphical models: Lagrangian and maximum entropy approaches," 2008, PhD Thesis.

[16] J. Johnson, D. Malioutov, and A. Willsky, "Lagrangian relaxation for map estimation in graphical models," in *Proc. of the Allerton Conf. on Commun., Control, and Computing*, Sept. 2007.

[17] D. Sontag and T. Jaakkola, "New outer bounds on the marginal polytope," in *Neural Information Processing Systems (NIPS)*, Dec. 2007.

[18] N. Komodakis, N.Paragios, and G. Tziritas, "MRF energy minimization and beyond via dual decomposition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, (in press).

[19] B. Aspvall, M. Plass, and R. Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," in *Information Processing Letters*, 1979, 8(3).

[20] M. Chertkov and M. Stepanov, "An Efficient Pseudocodeword Search Algorithm for Linear Programming Decoding of LDPC Codes," *IEEE Trans. Inform. Theory*, vol. 54, no. 4, Apr. 2008.

[21] S. Kudekar, J. K. Johnson, and M. Chertkov, "Linear programming based receivers for detection of two-dimensional intersymbol interference channels," in *Proc. of the IEEE Int. Symposium on Inform. Theory*, July 2011.