Uniformly reweighted APP Decoder for memory efficient decoding of LDPC Codes

Velimir Ilić[1], Elsa Dupraz[2], David Declercq[3], Bane Vasić[4]

***Abstract*— In this paper we propose a uniformly reweighted a posteriori probability (APP) decoder. It is derived as an algorithm of approximate Bayesian inference on the LDPC code graph, and a correction parameter is introduced and numerically optimized to overcome the suboptimaly of the decoder and improve performance compared to the original APP decoder. In addition, we propose a memory efficient implementation of the algorithm that requires memory that is linear only in the codeword length.**

I. Introduction

Belief propagation (BP) is an iterative message-passing algorithm for decoding low density parity check (LDPC) codes, widely used in many systems. Despite its good error correction performances and capability of approaching the Shannon limit, BP suffers from large memory requirements for message processing and storage, proportional to the number of edges in the Tanner graph of the code. Such large memory requirements, coupled with additional hardware resources needed for the message updating, make the BP less attractive in applications with stringing constraints.

A posteriori probability (APP) decoder [1] is a suboptimal alternative to BP, in which the variable node processing is simplified by allowing variables to send messages in an intrinsic manner, and a message from a variable node corresponds to a posteriori value used to estimate that variable, This property admits a memory efficient implementation.

The APP decoder can be seen as an iterative Bayesian procedure which computes the a posteriori bit probabilities conditioned on the previously estimated probabilities and the set of check node constraints. Its suboptimality comes from the approximation by which only the closest check constraint neighborhood of a variable node is taken into account during the iterative Bayesian procedure.

In this paper we propose a parameterized version of the APP decoder called uniformly reweighted APP decoder (URAPP), which can be seen as a suboptimal variant of the tree reweighted belief propagation [2], [3]. The algorithm is derived in the Bayesian framework and the source of the suboptimalily is recognized and treated using a reweighted parameter which, in general, depends on the iteration and node considered. As the simplest case, we consider the uniformly reweighted APP decoder, where a constant value is used for the correction parameter and optimized through simulations. The experiments on binary symmetric channel show that the optimal parameter value does not depend on the cross-over probability value and that URAPP has significantly better performance than the original version of the APP introduced in [1].

Although the APP decoders can be implementes in a memory efficient way, this advantage has not been recognized in the original paper [1]. On the other hand, the existing memory-aware hardware architectures of APP decoders [4], [5] suffer from the performance degradation caused by further approximations of variable node processing function, introduced to reduce complexity. As the second contribution of the paper, we propose a memory-efficient realization of the reweighted APP decoder, which requires memory proportional to the number of nodes in the Tanner graph of the LDPC code, rather than to the number of edges, as proposed in [1], which is a significant saving.

The paper is organized as follows. In Section II we introduce the notation and give deffinitions. The URAPP decoder is derived in the Section III, while its memory efficient implementation is considered in the Section IV. The simulation results are presented in the Section V.

II. Preliminaries

An regular LDPC code is a linear block code defined by a sparse parity-check matrix H. We denote by (M, N) the size of H. A codeword is a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_N) \in \{0,1\}^N$ that satisfies $H\boldsymbol{x}^T = 0$, where $\boldsymbol{x}^T$ denotes the transposed (column) vector. The Tanner graph [6] of an LDPC code is a bipartite graph whose adjacency matrix is the parity-check matrix of the code H. It contains two types of nodes: a set of variable-nodes $\mathcal{N} = \{v_1, v_2, \ldots, v_N\}$, corresponding to the N columns of H, and a set of check-nodes $\mathcal{M} = \{c_1, c_2, \ldots, c_M\}$, corresponding to the M rows of H. A variable-node v_n and a check-node c_m are connected by an edge if and only if the corresponding entry of H is non-zero.

The set of indices of check-nodes connected to the variable-node v_n is denoted with $\mathcal{M}(n)$ and the set of indices of variable-nodes connected to the check-node c_m is denoted with $\mathcal{N}(m)$. The direct neighborhood of a node v_n is defined with $\bar{\mathcal{M}}(n) = \bigcup_{m\in\mathcal{M}(n)} \bigcup_{i\in\mathcal{N}(m)\setminus n} v_i$.

We consider the binary symmetric channel (BSC), where a binary codeword $\boldsymbol{x}$ is transmitted, and we denote by $\boldsymbol{y} =$

*This work is funded in by the NSF under grants CCF-1314147 and CCF-0963726 and in part by the Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project).

[1]Velimir Ilić is with Mathematical Institute SANU, Belgrade, Serbia velimir.ilic@gmail.com

[2]Elza Dupraz is with ETIS laboratory, Cergy-Pontoise, France dupraz@ensea.fr

[3]David Declercq is with ETIS laboratory, Cergy-Pontoise, France declercq@ensea.fr

[4]Bane Vasić is with Department of ECE, University of Arizona, Tucson vasic@ece.arizona.edu

$(y_1, y_2, \ldots, y_N)$ the received sequence. The BSC is defined by the probabilistic model

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{n=1}^{N} \Pr(y_n|x_n) \prod_{m=1}^{M} \mathbf{1}(\sum_{n\in\mathcal{N}(m)} x_n)$$

where $\Pr(x|y)$ is a channel likelihood, $\mathbf{1}$ is the indicator function and $\sum_{n\in\mathcal{N}(m)} x_n$ are modulo 2 sums determined by the parity check matrix. The probability $P_e \equiv \Pr(x \neq y)$ is called cross-over probability.

III. Reweighted APP Decoder

The goal of the decoding is to compute the a posteriori probability

$$\Pr(x_n|\boldsymbol{y}, \mathcal{C}) \tag{1}$$

where $\mathcal{C} = \left\{ \sum_{k\in\mathcal{N}(m)} x_k = 0 \right\}_{m\in\mathcal{M}}$ is the set of constraints defined by the parity check matrix . According to the channel model the a posteriori probability can be expressed by the usage of the Bayesian formula

$$\Pr(x_n|\boldsymbol{y}, \mathcal{C}) \propto \Pr(y_n|x_n) \times \Pr\left(\mathcal{C}|x_n, \boldsymbol{y}_{\setminus n}\right), \tag{2}$$

where we denote $\boldsymbol{y}_{\setminus n} = (y_1, \ldots, y_{n-1}, y_{n+1}, \ldots, y_N)$.

Belief propagation (BP) has been introduced to approximately compute the a posteriori probability (1) by iterative computation of the check likelihoods $\Pr\left(\mathcal{C}|x_n, \boldsymbol{y}_{\setminus n}\right)$. Using the computed check likelihoods, BP in each iteration successively produces estimated a posteriori probabilities $\{\hat{p}_n\}_{n=1\ldots N}$, for the codeword symbols x_n, which are used for decision making on bit values.

The computation procedure is exact if the Tanner graph corresponding to the code is a tree, while it achieves very good accuracy for sparse codes. Its computational complexity is low, but it requires a memory storage growing with the number of edges in the graph, which is often seen as a limitation for practical applications. In this section we consider its suboptimal variant called the a posteriori probability (APP) decoder, and propose an uniformly reweighted APP decoder, which do not have this drawback.

Let a set of the constraints which corresponds to the neighborhood of the node v_n be denoted with $\mathcal{C}_n = \left\{ \sum_{k\in\mathcal{N}(m)} x_k = 0 \right\}_{m\in\mathcal{M}(n)}$. Suppose that we have already given the set of estimated probabilities $\{\hat{q}_k\}_{k\in\bar{\mathcal{M}}(n)}$ for the bits in the neighborhood of a node v_n called a priori probabilities . The APP decoder [1] makes simplification of a posteriori probabilities (1) and computes the probabilities $\Pr\left(x_n|\boldsymbol{y}, \mathcal{C}_n, \{\hat{q}_k\}_{k\in\bar{\mathcal{M}}(n)}\right)$ instead of $\Pr(x_n|\boldsymbol{y}, \mathcal{C})$. According to the Bayes formula we have

$$\Pr(x_n|\boldsymbol{y}, \mathcal{C}) \approx \Pr\left(x_n|\boldsymbol{y}, \mathcal{C}_n, \{\hat{q}_k\}_{k\in\bar{\mathcal{M}}(n)}\right) \propto p(y_n|x_n) \times \Pr\left(\mathcal{C}_n \mid x_n, \boldsymbol{y}_{\setminus n}, \{\hat{q}_k\}_{k\in\bar{\mathcal{M}}(n)}\right). \tag{3}$$

In the similar way as in the BP, the check node likelihoods are computed in an iterative process. First, according to [7], the check likelihoods can be computed as

$$\Pr\left(\mathcal{C}_n \mid x_n, \boldsymbol{y}_{\setminus n}, \{\hat{q}_k\}_{k\in\bar{\mathcal{M}}(n)}\right) = \prod_{m\in\mathcal{M}(n)} \hat{p}_{mn}(x_n), \tag{4}$$

where

$$\hat{p}_{mn}(x_n) = \begin{cases} \frac{1}{2} - \frac{1}{2} \prod_{k\in\mathcal{N}(m)\setminus n} (1 - 2\hat{q}_k(x_n)); & \text{for } x_n = 1 \\ \frac{1}{2} + \frac{1}{2} \prod_{k\in\mathcal{N}(m)\setminus n} (1 - 2\hat{q}_k(x_n)); & \text{for } x_n = 0 \end{cases} \tag{5}$$

Once we have computed the a posteriori values $\{\hat{p}_n\}_{n=1\ldots N}$ using the expression (3), they are are assign to the a priori probabilities $\{\hat{p}_n\}_{n=1\ldots N}$ and new iteration starts. As an initial prior probabilities the channel likelihoods $\Pr(y_n|x_n)$ are taken. Note that the check node likelihoods depend on $\boldsymbol{y}_{\setminus n}$ only in the first iteration.

Recall that in APP decoders two types approximations for the check likelihood computation are done:

- Only the local constraints (i.e. in the direct neighborhood) are directly taken into account, and
- The check likelihoods are computed using the a priori probabilities which are obtained through the iterative process.

In the following paragraphs we propose the parameterized version of the APP decoder called uniformly reweighted APP decoder (URAPP). The drawbacks of the APP decoder are handled using a reweighting parameter α for improving the correctness of a check node likelihood computation. The URAPP is derived from the equations (3) and (4), in a way that the probability $\hat{p}_{mn}(x_n)$, given by (5), is replaced with its α escort distribution [8]:

$$\hat{p}_{mn}(x_n) \leftarrow \hat{p}^{(\alpha)}_{mn}(x_n) = \frac{\hat{p}^{\alpha}_{mn}(x_n)}{\sum_{x_n} \hat{p}^{\alpha}_{mn}(x_n)}; \quad \alpha \in [0, 1]. \tag{6}$$

Accordingly, the URAPP decoder runs as follows.

1) For all nodes v_n initialize a priori values to $\hat{q}_n(x_n) = \Pr(x_n \mid y_n)$;
2) For all nodes v_n compute a posteriori probabilities:

$$\hat{p}_n(x_n) \propto p_n(x_n) \times \prod_{m\in\bar{\mathcal{M}}(n)} \hat{p}_{mn}(x_n)^{\alpha}, \tag{7}$$

   where $\hat{p}_{mn}(x_n)$ is computed according to (5);
3) For all nodes v_n set $\hat{q}_n \leftarrow \hat{p}_n$ and go to 2 until a convergence criterion has been reached.

In the limit cases, $\alpha = 0$ and $\alpha = 1$, the escort distribution reduces to the uniform and to the original one, respectively. Accordingly, as the α becomes smaller, the escort distribution becomes closer to the uniform distribution. Put differently, in the case of small α the check likelihood will be smoothed and the equation (7) will be determined by the channel likelihood. Oppositely, in the case of $\alpha = 1$ the URAPP reduces to APP decoder. The optimal value for α is between 0 and 1. If properly optimized, the parameter α can significantly improve the accuracy of the a posteriori probability estimation. In Section V we present the experimental results for the URAPP when the α is determined by a brute force optimization.

IV. Memory efficient implementation of URAPP decoder

For the implementation purposes, in order to avoid the computation of the normalization coefficient in the equation (7) and to reduce the number of multiplications, the APP decoders are commonly represented in log-ratio domain message passing version [1]. The message passing version can be directly derived from the previously defined steps 1-3, if we set

$$\hat{\gamma}_n = \log \frac{\hat{p}_n(1)}{\hat{p}_n(0)} \quad \text{and} \quad \mu_{m\to n} = \log \frac{\hat{p}_{mn}(1)}{\hat{p}_{mn}(0)}$$

as follows.

Initialization: Variable-nodes are initialized to *a priori* values $(\gamma_1, \gamma_2, \ldots, \gamma_n)$, which are in the case of BI-AWGN channel equal to the received sequence $(y_1, y_2, \ldots, y_N)$, prior to the first iteration of the APP decoder:

$$\tilde{\gamma}_n^{(0)} = \gamma_n = \frac{p(x_n = 0|y_n)}{P(x_n = 1|y_n)}. \tag{8}$$

Iterative processing:

1) Check-node processing: consists in computing the check-to-variable messages $\mu_{m\to n}^{(k)}$, for all check-nodes m and their neighbor variable-nodes v_n;

$$\mu_{m\to n}^{(k)} = \boxplus_{k\in\mathcal{N}(m)\setminus n} \tilde{\gamma}_k^{(k-1)}, \tag{9}$$

where $\boxplus_{i\in\mathcal{N}(m)\setminus n}$ stands for the summation over the set $\mathcal{N}(m)\setminus n$ induced by the box-sum operation defined as

$$x \boxplus y = \log \frac{1 + \mathrm{e}^x \mathrm{e}^y}{\mathrm{e}^x + \mathrm{e}^y} \tag{10}$$

2) A posteriori information update: consists in computing the a posteriori messages $\hat{\gamma}_n^{(k)}$, for all variable-nodes v_n,

$$\tilde{\gamma}_n^{(k)} = \gamma_n + \alpha \sum_{m\in\mathcal{M}(n)} \mu_{m\to n}^{(k)}. \tag{11}$$

3) Hard decision: Estimated (binary) values of sent bits, $\hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$, according to the rule: $\tilde{\gamma}_n^{(k)} > 0$ then $x_n^{(k)} = 0$, otherwise $x_n^{(k)} = 1$. The decoder stops when either $\hat{\boldsymbol{x}}$ is a codeword or a maximum number of decoding iterations is reached.

Check to variable messages requires the computation of all partial sums $\boxplus_{k\in\mathcal{N}(m)\setminus n} \tilde{\gamma}_k^{(k-1)}$, which can efficiently be computed using the inverse operation for $\boxplus$ called minus-box operator:

$$x \boxminus y = \log \frac{1 - \mathrm{e}^x \mathrm{e}^y}{\mathrm{e}^x - \mathrm{e}^y} \tag{12}$$

It is easy to check that $x \boxplus y \boxminus y = x$. Using the $\boxminus$ operator, the sum

$$\Psi_m^k = \boxplus_{k\in\mathcal{N}(m)\setminus n} \tilde{\gamma}_k^{(k-1)} \tag{13}$$

can be computed once per iteration and node, and all the messages can be computed for all $n \in \mathcal{N}(m)$ as

$$\mu_{m\to n}^{(k)} = \Psi_m^{(k)} - \hat{\gamma}_n^{(k)}. \tag{14}$$

Algorithm 1: Semi-parallel URAPP decoder

Input: $\boldsymbol{y} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ▷ received word
Output: $\hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$ ▷ estimated codeword

Initialization:

for each $\{v_n\}_{n=1,\ldots,N}$ **do** $\gamma_n = \log \frac{Pr(x_n = 0|y_n)}{Pr(x_n = 1|y_n)}$;
for each $\{v_n\}_{n=1,\ldots,N}$ **do** $\hat{\gamma}_n^0 = \gamma_n$;

Iteration loop: $k > 0$
Total check-sum computation
for each $\{c_m\}_{m=1,\ldots,M}$ **do** $\Psi_m^{k-1} = \boxplus_{n\in\mathcal{N}(m)} \hat{\gamma}_n^{k-1}$

partial a posteriori update
for each $\{c_m\}_{m=1,\ldots,M}$ **do**
  for each $v_n \in \mathcal{H}(c_m)$ **do**

$$\mu = \Psi_m^{k-1} \boxminus \hat{\gamma}_n^{k-1}$$
$$\hat{\gamma}_n^k = \hat{\gamma}_n^k + \alpha\mu$$

a posteriori update
for each $\{v_n\}_{n=1,\ldots,N}$ **do** $\hat{\gamma}_n^{k-1} = \hat{\gamma}_n^k$
hard decision
for each $\{v_n\}_{n=1,\ldots,N}$ **do** $\hat{x}_n = (1 - \mathrm{sign}(\tilde{\gamma}_n))/2$
if $\hat{\boldsymbol{x}}$ is codeword **then** exit the iteration loop

End iteration loop

In a common, parallel message passing implementation of the APP decoder, all the variable nodes take the message at same time, the a posteriori values are computed, which completes one iteration. Although this version provides high throughput, it suffers from the high memory requirements, proportional to the number of edges in the Tanner graph, since an iteration requires the storing of all check to variable messages for one iteration.

In the **Algorithm 1** we present semi-parallel, memory efficient implementation of the URAPP decoder, based on the ideas proposed in [9] for the BP algorithm. Instead of the messages, we store only the values $\Psi_m^{k-1} = \boxplus_{n\in\mathcal{N}(m)} \hat{\gamma}_n^{k-1}$ which are used for the computation of the posterior values $\hat{\gamma}_n^k$, and at one iteration, all variable nodes can be partially updated during the computations in all variable nodes. Both implementations provide exactly the same output after each iteration and have the same computational requirements. Although the former one has the smaller throughput, since two check nodes might try to access the same variable node to update its a posteriori value, it needs the storing only the values in variable nodes. As a result it have the complexity proportional to the number of nodes, which is a significant saving.

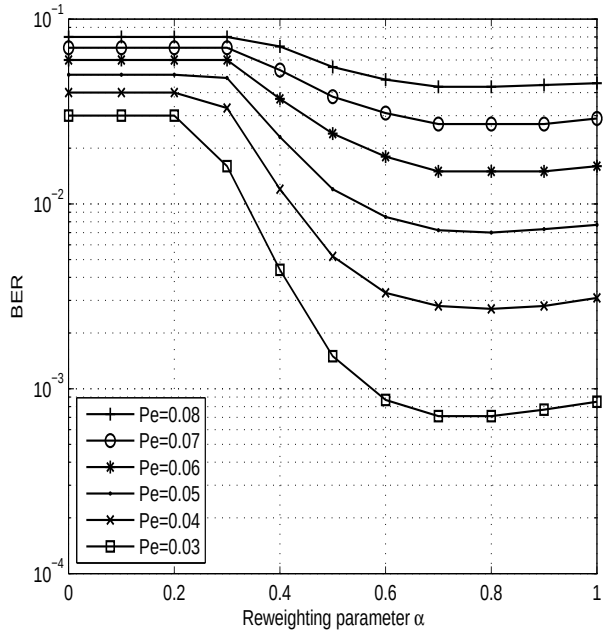

Fig. 1: BER as a function of the reweighting factor for different values of crossover probability for Tanner $(3, 5)$ code, with the maximal numer of iteration set to 5.

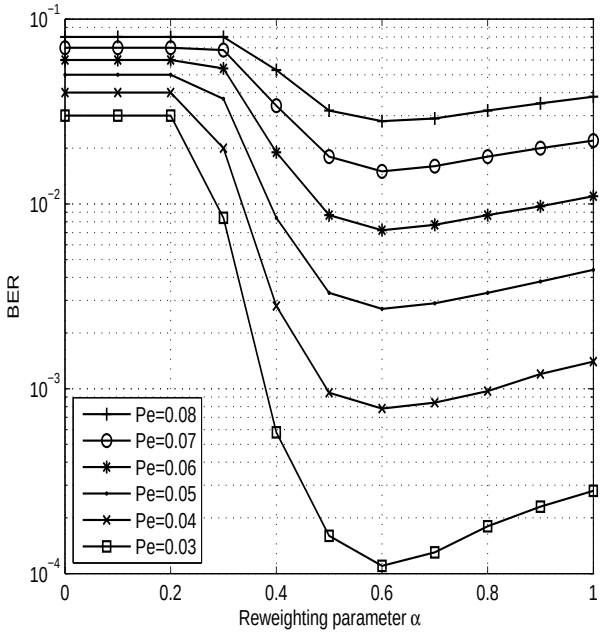

Fig. 2: BER as a function of the reweighting factor for different values of crossover probability for Tanner $(3, 5)$ code, with the maximal numer of iteration set to 10.

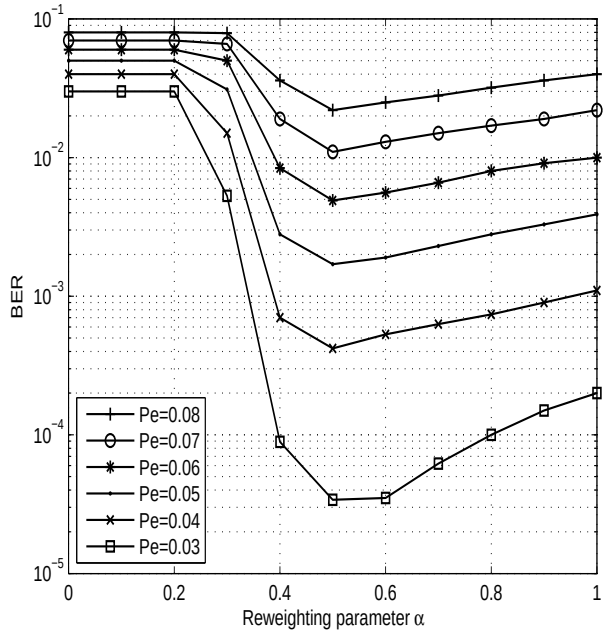

Fig. 3: BER as a function of the reweighting factor for different values of crossover probability for Tanner $(3, 5)$ code, with the maximal numer of iteration set to 20.

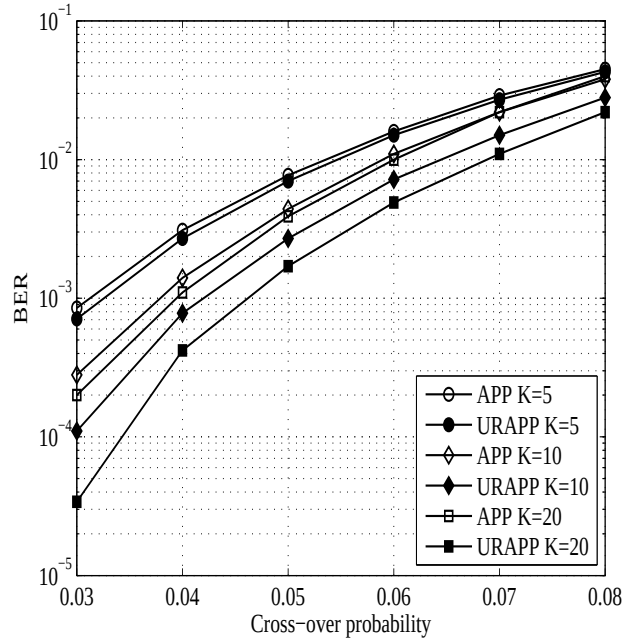

Fig. 4: BER as a function of crossover probability. APP decoders are marked with empty markers, URAPP decoders are marked with the black markers.

V. Reweighting factor optimization and simulation results

In this section we present simulation results for the URAPP. We consider Tanner $(3, 5)$ code [6]. Decoders with small number of iterations and suitable for high-throughput systems [5]. Optimal reweighted parameter is obtained through simulations. The results of the optimization procedure, for the maximal number of iterations are set to 5, 10 and 20, are shown in Figures 1, 2 and 3. They show that the optimum value of reweighted factor does

not depend on the cross-over probability, but only on the maximal number of iterations. As discussed in Section III, the reweighted factor slows down the convergence. This agrees with the simulation results, which show that the value of the factor is inversely proportional to the maximum number of iterations. Fig. 4 shows the bit error rate (BER) performances of standard and uniformly reweighted APP decoders, with reweighting parameters obtained from the optimization procedure, for the maximal number of iterations $K = 5$, $K = 10$ and $K = 20$. For $K = 5$, optimal value of the reweighting factor α is 0.7 and there is not a significant BER improvement over the unweighted APP decoder. As the K becomes larger the effect on α is larger, specially in the error floor region, since the reweighting factor emphasizes puts a higher importance on the channel likelihoods.

VI. Conclusion

In this paper we proposed memory efficient uniformly reweighted APP decoder (URAPP) for decoding of LDPC codes, with the memory complexity proportional to the number of nodes in the Tanner graph of the code. The algorithm can be seen as an suboptimal variant of the tree reweighted belief propagation [2], [3] and operates as a parameterized Bayesian inference algorithm, improving the performances of previously propose APP decoder [1]. Subotpimallity of the APP decoder comes from the fact that in each iteration and for each variable node only the neighboring check constraints are taken into account. We introduced the reweighting parameter which deals with this source of suboptimallity and empirically showed the significant performance improvements even in the case when the a constant value is used for the correction parameter. Taking into account the graph structure and dynamics of the URAPP decoder for the parameter estimation seem like a promising direction, which is the part of our future work on the parameter adaptive version of the reweighted APP decoder. The simulation results on binary simetric channel for Tanner $(3, 5)$ code [6] show that the reweighting factor does not depend on the cross-over probability, but only on the maximal number of iterations taken in to decoder. Currently, we are exploring its dependence on LDPC code parameters.

References

[1] M. Fossorier, M. Mihaljević, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *Communications, IEEE Transactions on*, vol. 47, no. 5, pp. 673–680, May 1999.

[2] H. Wymeersch, F. Penna, and V. Savic, "Uniformly reweighted belief propagation for estimation and detection in wireless networks," *Wireless Communications, IEEE Transactions on*, vol. 11, no. 4, pp. 1587–1595, April 2012.

[3] J. Liu and R. de Lamare, "Low-latency reweighted belief propagation decoding for ldpc codes," *Communications Letters, IEEE*, vol. 16, no. 10, pp. 1660–1663, October 2012.

[4] K. Karplus and H. Krit, "A semi-systolic decoder for the pdsc-73 error-correcting code," *Discrete Applied Mathematics*, vol. 33, no. 1–3, pp. 109 – 128, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X91901119

[5] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High throughput low-density parity-check decoder architectures," in *Proc. of Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, vol. 5, 2001, pp. 3019–3024 vol.5.

[6] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, May 1981.

[7] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[8] C. Beck and F. Schögl, *Thermodynamics of Chaotic Systems: An Introduction*, ser. Cambridge Nonlinear Science Series. Cambridge University Press, 1995. [Online]. Available: http://books.google.rs/books?id=GyPpZ-Lg6KAC

[9] F. Guilloud, E. Boutillon, J. Tousch, and J.-L. Danger, "Generic description and synthesis of ldpc decoders," *Communications, IEEE Transactions on*, vol. 55, no. 11, pp. 2084–2091, Nov 2007.

[10]