AN EFFICIENT MESSAGE-PASSING SCHEDULE FOR LDPC DECODING

Eran Sharon *Simon Litsyn* *Jacob Goldberger*

Tel-Aviv University
{eransh,litsyn}@eng.tau.ac.il

University of Toronto
jacob@cs.toronto.edu

ABSTRACT

An efficient decoding schedule for low-density parity-check (LDPC) codes that outperforms the conventional approach both in terms of complexity and performance is presented. Conventionally, in each iteration all symbol nodes and subsequently all the check nodes send messages to their neighbors ("flooding schedule"). In contrast, in the proposed method, the updating of nodes is performed according to a serial schedule which propagates the information twice as fast. A Density Evolution (DE) algorithm for asymptotic analysis of the new schedule is derived, showing that when working near the code's capacity, the decoder converges in approximately half the number of iterations. In addition a Concentration Theorem is proved, showing that for a randomly chosen serial schedule, code graph, and decoder input, the decoder's performance approaches its expected one as predicted by the DE algorithm, when the code length grows

1. INTRODUCTION

The most attractive feature of LDPC codes is their ability to achieve a significant fraction of the channel capacity at relatively low complexity using iterative decoding algorithms. In this work we deal with message-passing iterative decoding algorithms, in particular the belief-propagation algorithm (BP). These algorithms rely on a graph-based representation of codes, where the decoding can be understood as message passing between nodes in the graph. The order of passing the messages between the nodes is referred to as an updating rule or a schedule. The standard message-passing schedule for decoding LDPC code is a version of the so-called *flooding* schedule [5], in which in each iteration all the symbol nodes, and subsequently all the check nodes, pass new messages to their neighbors. It was pointed out by Forney [1]: "An open question is whether different schedules could improve iterative decoding algorithms". Here we provide an affirmative answer to this question by pointing out a schedule that is better than the flooding schedule.

Some versions of non-flooding schedules were proposed earlier. In [6] Mao *et al* suggested a probabilistic variant of the flooding schedule, taking into account the loop structure of the code's graph, at which different symbol nodes update their outgoing messages in accordance with the length of the shortest cycle through them. The proposed schedule was shown to reduce the error floor and the number of undetected errors. In [2],[4], a serial decoding schedule was proposed, based on a serial update of symbol nodes' messages which can be considered as shuffling of the flooding schedule. Both papers provide empirical results indicating fast convergence of the serial schedule, though no analytical analysis is provided.

In this work we propose some new serial scheduling strategies, among them we rediscover the serial decoding schedule proposed in [2],[4]. The proposed (dual) schedule based on a serial update of check nodes' messages is more efficient than the one in [2],[4], since it satisfies better memory requirements, while preserving the fast convergence property of the symbol nodes serial schedule. Moreover, we provide a rigorous analysis for the suggested algorithms confirming the fast convergence of the serial schedule.

2. EFFICIENT SCHEDULING

An LDPC code can be defined by a bipartite graph, a.k.a. Tanner graph or factor graph, constructed from the parity-check matrix. The graph consists of two types of nodes, symbol nodes and check nodes. A symbol node (denoted by v) represents a symbol in the transmitted codeword and a check node (denoted by c) represents a parity check constraint. Each check node is connected by an edge to the symbol nodes it checks. In each round of the belief propagation (BP) iterative algorithm each node in the graph passes messages to its neighbors along the edges it is incident to. Denote the message passing from a symbol node v to a check node c by Q_{vc} and the message passing from c to v by R_{cv}. Let $p_v(0)$ be the probability that the transmitted bit v is zero. LDPC decoding algorithm can be described in the log-likelihood-ratio (LLR) domain with a reduced complexity and a better numerical stability. Denote $P_v = \log \frac{p_v(0)}{p_v(1)}$. In BP the messages from the symbol nodes to the check nodes are:

$$Q_{vc} \leftarrow P_v \quad + \sum_{c' \in N(v) \setminus c} R_{c'v} \tag{1}$$

such that $N(\cdot)$ is the set of neighboring nodes in the graph. The messages from the check nodes to the symbol nodes are:

$$R_{cv} \leftarrow 2 \tanh^{-1} \left(\prod_{v' \in N(c) \setminus v} \tanh(\frac{Q_{v'c}}{2}) \right) \tag{2}$$

The computations at the check nodes can be simplified further by performing them in the log domain as follows:

$$R_{cv} \leftarrow \varphi^{-1} \left(\sum_{v' \in N(c) \setminus v} \varphi(Q_{v'c}) \right) \tag{3}$$

such that $\varphi(x) = (\text{sign}(x)\,, -\log \tanh(\frac{|x|}{2}))$,
$\varphi^{-1}(\text{sign}, x) = (-1)^{\text{sign}} \times -\log \tanh(\frac{x}{2})$ and sign(x) is the indicator function $1_{\{x<0\}}$. The soft decoding result for each symbol v is obtained as follows:

$$Q_v \leftarrow P_v \quad + \sum_{c \in N(v)} R_{cv} \tag{4}$$

The order of passing the messages between the nodes is called updating rule or schedule.

In the case of cycle-free graphs the situation is simple. Given any iterative schedule in which messages are sent in both directions of all

the edges in each iteration, the BP algorithm will converge after a finite number of iterations to the exact a-posterior probabilities. The number of iterations is bounded by $D/2$ where D is the length of the longest path in the graph. In case of cycle-free graph we can even construct an optimal schedule [5] in the sense of the minimum number of messages that needed to be sent till it converges. It is precisely $2E$, where E is the number of graph edges, and exactly one message will pass in each direction on each edge, hence a single iteration is needed for convergence.

In graphs with cycles the behavior of iterative decoding is more involved and less clear. The algorithm may produce inaccurate a-posterior probabilities. In this case, use of a specific schedule not only influences the convergence rate, but can also affect the algorithm convergence at all.

The standard message-passing schedule for decoding LDPC code, described by Gallager [3], is a version of the so-called *flooding* schedule [5], in which in each iteration all the symbol nodes, and subsequently all the check nodes, pass new messages to their neighbors. For LDPC, though the flooding schedule is popular, there is no evidence that it is optimal.

In [2],[4], a serial decoding schedule was proposed. It is based on a serial update of symbol nodes' messages which can be considered as shuffling of the flooding schedule. Here, we propose a dual schedule, which is based on a serial update of check nodes' messages. The dual schedule is more efficient in terms of memory requirements while maintaining fast convergence. Instead of sending all messages from symbol nodes to check nodes, and then all messages from check nodes to symbol nodes, as done in the flooding schedule, we go over the check nodes in some order and for each node we send all messages into the node and then all messages out from the node:
Let C and V denote the sets of check nodes and variable nodes respectively. Each check node $c \in C$ sends the following messages:

1. Compute Q_{vc} for each $v \in N(c)$ (i.e. send all Q_{vc} messages into the node c).

2. Compute R_{cv} for each $v \in N(c)$ (i.e. send all R_{cv} messages from check node c).

This schedule enables immediate propagation of the messages unlike the flooding schedule where the updated messages can propagate only in the next iteration.

The decoding schedule can be efficiently implemented with a significant reduction of memory. This is achieved by using Q_v output messages and R_{cv} messages to compute the Q_{vc} messages on the fly, thus avoiding the need to keep an additional memory for the Q_{vc} messages. Additionally, the same memory which is initialized with the channel messages P_v is used to store the iteratively updated Q_v output messages. The efficient message computation is derived by substituting (1) into (3); it gives the following updating formula for the incoming and outgoing messages from a check node c:

$$R_{cv}^{new} \leftarrow \varphi^{-1}\left(\sum_{v' \in N(c)\setminus v} \varphi(Q_v - R_{cv'})\right) \quad (5)$$

$$Q_v^{new} \leftarrow Q_v^{old} - R_{cv}^{old} + R_{cv}^{new} \quad (6)$$

The complete algorithm is given in Table 1.

Iterations of the flooding schedule can be completely parallelized, i.e. all symbol and check node messages can be updated simultaneously. The serial decoding is inherently serial, however, sets of check nodes' messages can be updated in parallel. If the check nodes are partitioned into sets $B_1, \ldots, B_m$ such that no two check nodes in a set are connected to the same symbol node, i.e.

$$\forall i \in \{1, \ldots, m\} \;\; \forall c, c' \in B_i \qquad N(c) \cap N(c') = \phi \quad (7)$$

then the check nodes in each set can be updated simultaneously. Since the fully parallel implementation is usually impossible due to the complex interconnect fabric between the computation nodes, the partially serial nature of the serial schedule is not limiting. In addition, it turns out that when the check nodes are partitioned into enough sets, even if the sets do not maintain the property (7), the performance is very close to the performance of the serial schedule as shown in Section 3. A serial schedule involving a serial update of sets of nodes we address as a *semi-serial schedule*. For instance, partition the check nodes into m equal sized sets $B_1, \ldots, B_m$, and perform an iteration by updating all the check nodes in B_1 simultaneously, then updating all the check nodes in B_2 simultaneously, and so on until we have processed set B_m. Note that the the semi-serial schedule is a hybrid of the serial schedule and the flood schedule. Actually, if $m = 1$ the semi-serial schedule coincides with the flood schedule, and if $m = |C|$ or the m sets satisfy (7) the semi-serial schedule coincides with the serial schedule.

The proposed serial schedule has many advantages: (a) The serial schedule converges in approximately half the number of iterations needed by the flooding schedule, as shown by simulation results (see Fig. 1a), and by the analysis of Section 3 (see Fig.2) (b) The performance of the serial schedule is equal to that of the flooding schedule when the decoders are allowed to perform many iterations, however, in practice since the maximal number of decoder iterations is limited, the serial schedule will perform better than the flooding schedule, as shown by simulation (Fig.1b). (c) The memory size (RAM) needed in an architecture based on the serial schedule is smaller then the memory required for implementing the flooding schedule or the symbol nodes serial schedule proposed in [2],[4] (even when implemented efficiently). (d) The serial schedule utilizes an efficient data structure, containing $N(c)$ entries for all $c \in C$, for storing the graph structure, while for the flooding schedule we also need a data structure for keeping $N(v)$ entries for all $v \in V$ (though an efficient implementation of the flooding schedule can avoid it). (e) The proposed serial schedule fits naturally implementing a decoder for the popular right regular LDPC codes, since it requires only check node computation units that will have constant degree. (f) The proposed serial schedule allows an extremely simple convergence testing which is performed as a by-product of the decoding process by checking that the sign of the variable S (see Table 1) is positive for $|C|$ consecutive checks.

initialization:
for all $v \in V$, $c \in C$ $\quad R_{cv} \leftarrow 0$
for all $v \in V$ $\quad Q_v \leftarrow P_v$

iteration:
for all $c \in C$
  $S \leftarrow \sum_{v \in N(c)} \varphi(Q_v - R_{cv})$
  for all $v \in N(c)$
    $Q_{temp} \leftarrow Q_v - R_{cv}$
    $R_{cv} \leftarrow \varphi^{-1}(S - \varphi(Q_{temp}))$
    $Q_v \leftarrow Q_{temp} + R_{cv}$
  end of loop
end of loop

Table 1: Efficient decoding algorithm

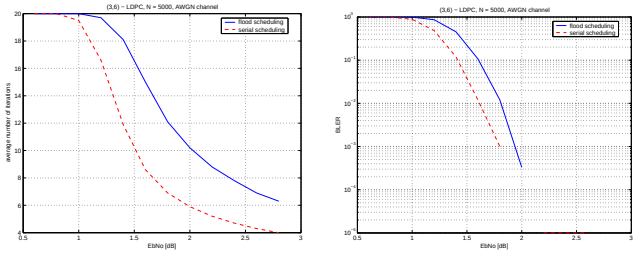

Fig. 1: Simulation results for a regular(3,6)-LDPC over the Gaussian channel with maximum of 20 iterations: (a) average iterations Vs $\frac{Eb}{No}$ (b) Block Error Rate Vs $\frac{Eb}{No}$

3. CONVERGENCE ANALYSIS

This section includes analysis of the convergence rate of the semi-serial and the serial schedules. We present here an analysis for the symbol nodes serial schedule proposed in [2] and [4], since we believe it is simpler to understand. However the same analysis applies to the check nodes serial schedule proposed in this paper. For the sake of simplicity, we provide the analysis only for regular (d_v, d_c)-LDPC codes, which are represented by regular bipartite graphs with constant symbol node degree d_v and constant check node degree d_c. We denote by $\mathcal{C}_n^{(d_v,d_c)}$ the ensemble of regular (d_v, d_c)-LDPC codes of length n.

3.1. Average computation tree depth

Consider a message passing along an edge e, depending on the iteration number. It is a function of the sub-graph of the code's graph that evolves from the edge and of the received symbols that are contained in the sub-graph. We refer to this sub-graph as the computation graph of e. Computation graphs which are cycle-free are computation trees. Note that a computation graph with cycles can also be spanned as a tree with replicated nodes.

Let D_e^l denote the depth of the computation graph spanned from edge e after l iterations. The depth of a computation graph of any edge for a flooding schedule grows by 2 at each iteration, thus for the flooding schedule $D_e^l = 2l$. For the serial schedule the depth of the computation graph after l iterations is not fixed and depends on the code's graph, the serial schedule by which the nodes are updated and the specific edge. However, the average depth of the computation graph for the serial schedule, assuming a cycle-free graph can be computed based on the following theorem:

Theorem 3.1.1 *For any regular cycle-free bipartite graph and for any serial schedule the average computation tree depth of a symbol node grows by 4 each iteration:*
$\overline{D_e}^{(l)} = \overline{D_e}^{(l-1)} + 4$

Note, that the cycle-free graph case provides an upper bound on the growth of the computation tree depth. In case of cycles the computation tree depth will grow slower. Theorem 3.1.1 shows that the average computation tree depth grows twice as fast in the serial schedule compared to the flooding schedule, implying that the serial schedule propagates information twice as fast.

3.2. Density Evolution analysis

LDPC codes exhibit a threshold phenomenon - as the codeword length tends to infinity, the bit error probability can be made arbitrarily small if the noise level is smaller than some constant threshold. Richardson *et al* developed an algorithm for determining the threshold for various message passing decoders based on the flooding schedule called Density Evolution [7]. The algorithm is based on iterative calculation of the densities of the message passed by the decoder in each iteration, assuming the bipartite graph that represents the code is cycle-free. This assumption is justified by a general concentration theorem [7] showing that the decoder's performance on random graphs converges exponentially with the code length to its expected performance. This expected performance in the limit of infinitely long codes can be determined from the corresponding cycle-free bipartite graph.

Note that under the cycle-free graph assumption all possible schedules converge to the exact posterior value after a finite number of iterations. Hence, the threshold of the flooding and serial schedules is the same and for threshold computation one can use the flooding DE. However, we are interested in the DE algorithm for prediction of the average number of iterations needed for convergence, which depends on the specific schedule that is used even under the cycle-free graph assumption.

For the sake of simplicity, we assume that the symbol nodes are divided into m equal size sets $B_1, B_2, \ldots, B_m$ and that the decoding iteration is performed by simultaneously updating all symbol nodes in set B_1, then set B_2 and so on till set B_m. For the semi-serial schedule, there is no restriction on the partition into the sets, however, for the serial schedule the sets are assumed to satisfy (7). A trivial partition into such sets is that each symbol node constitutes a different set.

Similarly to the flooding DE, the analysis is based on the assumption of a cycle-free graph, justified by the general concentration theorem proved in the next section. Denote by f_P the probability density function (pdf) of a channel LLR message P, assuming the zero codeword was transmitted. Denote by $f_{Q_i}^{(l)}$ the pdf of a LLR message sent from a symbol node belonging to the i'th set at the l'th iteration , assuming the zero codeword was transmitted. All pdf's are averaged over all tree-like graphs from the ensemble and over all decoder inputs. Let $\mathcal{DE}$ denote the DE operator expressing the pdf of an output message from a symbol node as a function of the pdfs of an input message to the symbol nodes and of the channel message [7]. The $\mathcal{DE}$ operator is a function of the LDPC ensemble for which the analysis is performed (the details are omitted).

Theorem 3.2.1 *Density Evolution of a semi-serial schedule with m sets for a given LDPC codes ensemble is described by:*

$$f_{Q_i}^{(l)} = \mathcal{DE}(f_P, \frac{i-1}{m}\sum_{j=1}^{i-1} f_{Q_j}^{(l)} + \frac{m-i+1}{m}\sum_{j=i}^{n} f_{Q_j}^{(l-1)})$$

$$f_Q^{(l)} = \frac{1}{m}\sum_{i=1}^{m} f_{Q_i}^{(l)}$$

where

$$f_{Q_i}^{(0)} = f_P \qquad i = 1, \ldots, m$$

Theorem 3.2.2 *Density Evolution of a serial schedule with m sets satisfying (7) for a given LDPC codes ensemble is described by:*

$$f_{Q_i}^{(l)} = \mathcal{DE}(f_P, \frac{i-1}{m-1}\sum_{j=1}^{i-1} f_{Q_j}^{(l)} + \frac{m-i}{m-1}\sum_{j=i+1}^{n} f_{Q_j}^{(l-1)})$$

$$f_Q^{(l)} = \frac{1}{m}\sum_{i=1}^{m} f_{Q_i}^{(l)}$$

where

$$f_{Q_i}^{(0)} = f_P \qquad i = 1, \ldots, m$$

The same results are obtained by the serial schedule DE algorithm regardless of the number of sets m that is used (this is not the case for the semi-serial schedule). Hence, one can use $m = 2$ when running the DE algorithm for the serial schedule.

Applying the DE algorithm for various ensembles of LDPC codes for various channels, we see that when working near the decoder's capacity, asymptotically (in the code length) the average number of iterations needed for convergence of a serial schedule is approximately half of what is needed for convergence of the flooding schedule. Furthermore, we see that the convergence rate of the semi-serial schedule approaches the convergence rate of the serial schedule as the number of sets increases.

The fraction of erroneous messages as a function of the number of iterations as predicted by the DE algorithm for the flooding schedule, the serial schedule and the semi-serial schedule for various values of m for a (3,6)-LDPC code over a Gaussian channel with channel parameter $Eb/No = 1.11dB$ is shown in Figures 2.

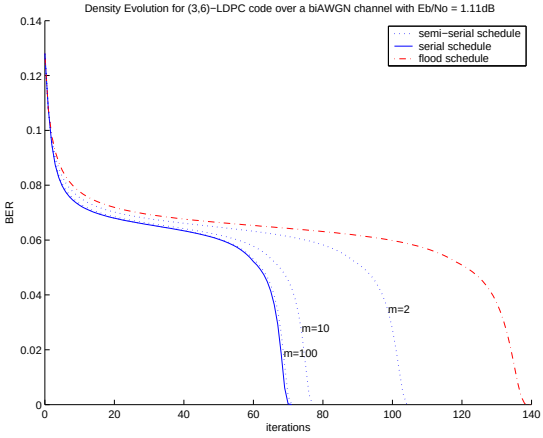

Fig. 2: Density Evolution analysis over Gaussian channel

3.3. Concentration Theorem

In the previous section, we saw how the average behavior of a given semi-serially or serially scheduled message passing decoder can be determined, assuming that the graph does not contain cycles. In this section, we follow [7] and prove a general concentration theorem for the semi-serial and serial schedules, showing that for a randomly chosen serial schedule, graph and decoder input the fraction of incorrect messages passed at the l'th decoding step is within an arbitrary ϵ of the value predicted by the proposed DE algorithm with probability that approaches 1 as the code length increases.

For a semi-serial schedule, the depth of a computation graph after l iterations is not fixed and depends on the graph, the serial schedule according to which the sets of symbol nodes are updated and the specific edge. However, it is easy to see that the depth of computation graph after l iterations is bounded by $D_e^l \leq 2ml$ (where m is the number of sets). Since for a semi-serial schedule the depth of computation graph is bounded, a similar concentration theorem to the one proved in [7] for the flooding schedule applies to the semi-serial schedule requiring minor changes in its proof:

Theorem 3.3.1 *(Concentration theorem for the semi-serial schedule). Let Z be the number of incorrect messages among all $d_v n$ symbol-to-check messages sent in the l-th decoding step of a semi-serial schedule with m symbol node sets. Further, let p be the expected number of incorrect messages passed along an arbitrary edge with a tree-like computation neighborhood at the l-th decoding step. Then there exist positive constants β, γ such that for any $\epsilon > 0$ and $n > \frac{2\gamma}{\epsilon}$.*

$$Pr\left(|Z/(nd_v) - p| > \epsilon\right) \leq 2e^{-\beta\epsilon^2 n} \tag{8}$$

The constant γ can be chosen as $M_{ml}^2 + \frac{d_v}{d_c} C_{ml}^2$ where, $M_{ml} = \sum_{i=0}^{ml} (d_v - 1)^i (d_c - 1)^i$ and $C_{ml} = 1 + (d_v - 1) \sum_{i=0}^{ml-1} (d_v - 1)^i (d_c - 1)^i$. The constant β can be chosen as $\frac{d_v^2}{(512 d_v + 32)(d_c d_v)^{2ml}}$.

Unfortunately, since for a serial schedule the number of sets m is not fixed and depends on the code length, the depth of the computation graph of an edge cannot be bounded by a constant even after a single iteration. Because the computation tree for a serial schedule at the l-th decoding step cannot be bounded by a constant, the flooding concentration theorem proved in [7] cannot be applied directly to the serial schedule and some additional arguments are needed to show that the probability that the computation tree is not bounded approaches zero as the code length increases:

Lemma 3.3.2 . *Let e be an edge in a randomly chosen element of $\mathcal{C}_n^{(d_v,d_c)}$ decoded using a serial schedule and let G_e^l be the computation graph of e at the l-th iteration, then for large n*

$$Pr\left(G_e^l \text{ is not tree-like with depth smaller then } \frac{4l\ln(n)}{\ln(\ln(n))}\right) \leq \frac{o(n)}{n} \to 0$$

Using Lemma 3.3.2, we are then able to prove the concentration theorem for the serial schedule.

Theorem 3.3.3 *(Concentration theorem for the serial schedule). Let Z be the number of incorrect messages among all $d_v n$ symbol-to-check messages sent in the l-th decoding step. Further, let p be the expected number of incorrect messages passed along an arbitrary edge with a tree-like computation neighborhood at the l-th decoding step. Then for any $\epsilon > 0$ and sufficiently large n.*

$$Pr\left(|Z/(nd_v) - p| > \epsilon\right) \leq \frac{2}{n^{1-o(1)}} \tag{9}$$

4. REFERENCES

[1] G.D. Forney Jr., "On iterative decoding and the two-way algorithm," Proc. Int. Symp. on Turbo Codes and Related Topics, pp. 12-15, 1997.

[2] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," The Proceedings 36th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA, November 2002.

[3] R.G. Gallager, "Low-density parity-check codes," IRE Trans. Info. Theory, vol. IT-8, pp. 21-28, 1962.

[4] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," submitted to Physical Review E, July 2002.

[5] F.R. Kschischang, B.J. Frey and H. Loeliger, "Factor Graphs and the sum-product algorithm," IEEE Trans. on Info. Theory, 2001.

[6] Y. Mao and A.H. Banihashemi, "A new schedule for decoding low-density parity-check codes," Proc. IEEE GlobeCom, Texas, 2001.

[7] 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.