Abstract—We consider an iterative message-passing decoder that can choose its decoding rule among a group of decoding algorithms at each iteration (for example: a software decoder). Each available decoding algorithm may have a different computation time and performance. We first show that with proper choice of algorithm at each iteration, decoding latency can significantly be reduced. We call such a decoder a gear-shift decoder because it changes its decoding rule (shifts gear) in order to guarantee both convergence and minimum decoding-latency. We also prove that the optimum gear-shift decoder (the one with the minimum decoding-latency) has a decoding threshold equal to or better than the best decoding threshold of the available algorithms. We use extrinsic information transfer charts and dynamic programming to find the optimum gear-shift decoder.

I. INTRODUCTION

Codes defined on graphs together with iterative message-passing algorithms are capable of approaching the capacity of many channel-types, e.g. [1], [2], [3], [4]. There are many different message-passing algorithms with different performance and complexity.

High-complexity decoding algorithms such as sum-product can correct more errors created by the channel noise, so they are very attractive when complexity and computation time is not an issue. Low-complexity decoding algorithms, however, are more attractive when we need fast decoders for delay-sensitive applications or high-throughput systems. Low-complexity decoding rules have two main drawbacks. First, they have a worse threshold of decoding compared to high-complexity algorithms so, to ensure convergence, a lower code rate should be used. Second, due to their relatively poor performance, more iterations of such algorithms are required to achieve a given target bit error rate. Hence, it is not obvious if the overall computations are any less.

Now consider the following scenario: A long-length regular (4,8) low-density parity-check (LDPC) code is used to protect data transmitted over a Gaussian channel whose signal to noise ratio (SNR) is 2.50 dB. Four different decoding algorithms—sum-product, min-sum, Algorithm E (erasure in the decoder) [1] and Gallager’s Algorithm B [5], [1]—are implemented in software. Based on these implementations, the computation time of these algorithms are 5.2, 2.8, 0.9 and 0.39 μsecond per bit per iteration (μs/b/iter) respectively. A routine analysis shows that we need 9 iterations of sum-product decoding to achieve a target message error rate of less than $10^{-7}$. That is to say, the decoding time is 46.8 μs/b. Using min-sum decoding, density evolution analysis shows that 20 iterations are required, hence the decoding time will be 56.0 μs/b. It can be easily verified that using Algorithm E or Gallager’s Algorithm B, the decoding will fail. Notice that although min-sum is a lower-complexity algorithm, its overall decoding time is longer.

Now assume that the decoder is allowed to vary its decoding strategy at each iteration. Using density evolution, one can verify that performing 4 iterations of sum-product, followed by 5 iterations of Algorithm E and followed by 4 iterations of Algorithm B the decoder will achieve the same target error rate in just 26.86μs/b. This is less than 58% of the time required by a sum-product decoder and is more than twice as fast as a min-sum decoder. It also makes use of Algorithms E and B, which are very fast decoding algorithms, but which could not be used on their own at this channel condition.

We call the strategy of switching from one algorithm to another “gear-shift decoding” as the decoder changes gears (its algorithm) in order to speed up the decoding. In this paper, given a set of decoding algorithms and their computation time, we show how to find the optimum combination in the sense of minimizing the decoding time. Gear-shift decoding was introduced in [6], but the problem was studied only in the special case that all the algorithms have equal complexity. Here, we generalize the idea to the case of algorithms with varying complexity and performance.

This paper is organized as follows. In Section II we introduce extrinsic information transfer (EXIT) charts as a decoding analysis method which will be used throughout this paper. In Section III we define and solve the gear-shift decoding optimization problem for minimum decoding-latency. Section IV presents some examples and Section V concludes the paper.

II. EXIT CHARTS

Message-passing iterative decoders can be analyzed using density evolution [2], which tracks the evolution of the probability density function (pdf) of extrinsic messages iteration by iteration. This analysis becomes intractable when the constituent codes are complex, e.g., in turbo codes. Another approach for analyzing iterative decoders, including codes with complicated constituent codes, is to use extrinsic information transfer charts [7], [8], [9].

Masoud Ardakani, Frank R. Kschischang
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto, Toronto, Ontario, CANADA, M5S 3G4
Email: {masoud,frank}@comm.utoronto.ca
Telephone: 1 (416) 946–8809
In EXIT chart analysis, the idea is to track the evolution of a single parameter—a measure of the decoder's success—iteration by iteration. For example, one can track the SNR of the extrinsic messages [8], [9], their error rate [10] or the mutual information between messages and decoded bits [7].

For many applications, EXIT charts are very accurate. For instance in [7], EXIT charts are used to approximate the behavior of iterative turbo decoders on a Gaussian channel very accurately. In [10], using EXIT charts, the threshold of convergence for LDPC codes is approximated within a few thousandths of a dB of the actual value. In addition, when the pdf of messages can truly be described by a single parameter, e.g., in an erasure channel, EXIT chart analysis is equivalent to density evolution.

As we will see, using EXIT charts, the gear-shift decoding problem (to be defined) can be solved as a dynamic program. Hence, we will work with the EXIT chart of each algorithm. In addition, we are most interested in EXIT chart based on message error probability. This approach reflects typical design goals, as one can request a target message error rate rather than a target SNR or mutual information. Thus, in the remainder of this paper, we assume that the EXIT charts are tracking the extrinsic message error rate. An EXIT chart can be thought as a function

\[
p_{\text{out}} = f(p_{\text{in}}, p_0),
\]

where \(p_0\) is the intrinsic message error rate, \(p_{\text{in}}\) is the extrinsic message error rate at the input of the iteration and \(p_{\text{out}}\) is the extrinsic message error rate at the output of the iteration.

Usually EXIT charts are presented by plotting both \(f\) and its inverse \(f^{-1}\) for a fixed \(p_0\). This makes the visualization of the decoder's performance easier (see Fig. 1). Thus EXIT charts allow one to study how many iterations are required to achieve a target message error rate.

If the "decoding tunnel" of an EXIT chart is closed, i.e., if for some \(p_{\text{in}}\) we have \(p_{\text{out}} > p_{\text{in}}\), convergence does not happen. In such cases we say that the EXIT chart is closed.

The convergence threshold \(p_0^*\) is the worst channel condition, for which the tunnel is open, i.e.,

\[
p_0^* = \arg \sup_{p_0} \{ f(p_{\text{in}}, p_0) < p_{\text{in}}, \text{ for all } 0 < p_{\text{in}} \leq p_0 \}.
\]

Fig. 1 shows the EXIT chart for a regular (3,6) LDPC code on a Gaussian channel under sum-product decoding and compares it with the actual results. This comparison shows that such analyses can be highly accurate.

III. GEAR-SHIFT DECODING

Gear-shift decoding, simply put, means allowing the decoder to change its decoding algorithm during the process of decoding. Gear-shift decoding for LDPC codes, interestingly, dates back to Gallager and his original work on LDPC codes [5]. For binary message-passing decoders, Gallager noticed that, by allowing a decision threshold to be changed during the process of decoding, the decoder’s performance and convergence threshold improves significantly. This gear-shifting algorithm of Gallager is referred to as Algorithm B in [1].

The importance of using different decoding algorithms in the case of binary message-passing has inspired other work. In hybrid decoding [11], the variable nodes are partitioned, with nodes in different partitions performing a different algorithm. This results in an average behavior which can be better than the convergence behavior of every single algorithm.

In soft decoding, however, it is well known that sum-product decoding minimizes error probability when the factor graph of the code is cycle free. Thus, gear-shift decoding cannot help to improve the threshold of convergence. However, as we discussed earlier, the complexity of decoding can be reduced by gear-shift decoding.

In this section, assuming that an EXIT chart analysis is accurate, we show how to find the optimum gear-shift decoder.

A. Definitions

Consider a set of \(N\) different decoding algorithms numbered from 1 to \(N\) whose EXIT charts are given as \(f_i(p_{\text{in}}, p_0)\), for \(1 \leq i \leq N\). The computation time for one iteration of algorithm \(i\) is \(t_i\), which is assumed to be independent of \(p_{\text{in}}\) and \(p_0\). A gear-shift decoder is denoted with a sequence of integers \(S = (a_1, a_2, \ldots, a_i(S))\), \(a_i \in \{1, 2, \ldots, N\}\), which means algorithm number \(a_1\) is used in the first iteration, algorithm number \(a_2\) is used in the second iteration, etc. The computation time for such a gear-shift decoder is \(T_S = \sum_{i=1}^{l(S)} t_{a_i}\), and the output message error rate of such a decoder is

\[
P_S(p_0) = f_{a_1(S)}(f_{a_2(S)}(\ldots f_{a_l(S)}(f_{a_1}(p_0, p_0), p_0), \ldots), p_0).
\]

Notice that the initial extrinsic message error rate is \(p_0\).

For a given target message error rate \(p_t\), we say a gear-shift decoder \(S\) is admissible if \(P_S(p_0) \leq p_t\). The "optimum gear-shift decoder", corresponds to the sequence \(S^*\) whose computation time is less than all other admissible sequences.

We assume that EXIT charts are increasing functions of \(p_{\text{in}}\). This is because, a greater \(p_{\text{in}}\), i.e., having more extrinsic
message errors at the input of an iteration, usually results in more message errors at the output of the iteration, i.e., a greater $p_{\text{out}}$. We also assume that the output messages of each algorithm are compatible with the input of each algorithm.

**B. Equally-complex algorithms**

If all the decoding rules have equal computation time, i.e., $t_i$ is a constant, it is clear that the optimum gear-shift decoder at each iteration uses the algorithm whose output extrinsic message error rate is the smallest of all.

Here we prove that the greedy strategy of choosing the algorithm with the best performance in each iteration results in the fastest convergence.

**Theorem 1:** Given a channel condition $p_0$ and a set of equally complex algorithms, a greedy algorithm achieves the target error rate in the least time.

**Proof:** Given a gear-shift decoder $S = \{a_1, a_2, \ldots, a_t\}$, $1 \leq a_t \leq N$, using (2), one can compute $P_S(p_0)$, i.e., the message error rate after $l$ iterations.

Now assume that $S^* = \{a_1^*, a_2^*, \ldots, a_t^*\}$ represents the sequence associated with the greedy algorithm of the same length $l$. If $S^*$ is not the best sequence for $l$ iterations, we call the best sequence $\hat{S} = \{a_1, a_2, \ldots, a_t\}$. Now change $\hat{S}$ as follows. At the first place where $S$ differs with $S^*$, say in iteration $i$, switch $a_i$ with $a_i^*$, leaving the rest of $\hat{S}$ unchanged and call the updated sequence $\hat{S}^*$. Since $\hat{S}^*$ is based on a greedy decision,

$$f_{a_i^*}^{(i-1)}(p_{\text{out}}; p_0) < f_{a_i}^{(i-1)}(p_{\text{out}}; p_0),$$

where $p_{\text{out}}^{(i-1)}$ shows the output message error probability after the first $i-1$ iterations. Notice that these first $i-1$ iterations are the same in both $\hat{S}^*$ and $\hat{S}$ so $p_{\text{out}}^{(i-1)}$ is the input error rate to both $\hat{S}$ and $\hat{S}^*$ at iteration $i$.

Since all EXIT charts are assumed to be increasing functions of $p_{\text{in}}$, it is evident that after $l$ iterations $\hat{S}^*$ offers a better $p_{\text{out}}$ than $\hat{S}$, which contradicts the assumption that $\hat{S}$ is the best decoding sequence. Therefore, $\hat{S}^*$ has the minimum message error rate among all sequences of length $l$. Since $l$ was chosen arbitrarily, $\hat{S}^*$ achieves a better error rate for any $l$ iterations and hence meets the target error rate in the least number of iterations.

**C. Algorithms with varying complexity**

When the computation times of different algorithms are not equal, it is not clear which combination of algorithms will result in the fastest decoding. Choosing the algorithm with the best performance may cost a lot of computation while for the same amount of computation another algorithm could produce better results. In this section we show that, finding the sequence of algorithms with the minimum computational complexity can be cast as a dynamic program. We first define the problem.

We use the general setup of Section III-A. At a fixed $p_0$, for each EXIT function $f_i(p_{\text{in}}, p_0)$, we define a quantized version $\hat{f}_i(p_{\text{in}})$ whose domain and range is the set $\mathcal{P} = \{p_0, p_1, \ldots, p_n\}$. Here, $p_0$ is the intrinsic message error rate, $p_0 > p_1 > \cdots > p_n$ and $p_n$ is equal to $p_t$ the target message error rate. For $p_{\text{in}} = p_m$, $0 \leq m \leq n$ we define

$$\hat{f}_i(p_{\text{in}}, p_0) = \begin{cases} p_0 & \text{if } f_i(p_{\text{in}}, p_0) \geq p_0 \\ p_k & \text{otherwise} \end{cases},$$

where $k$, $k \in \{0, \ldots, n\}$ is the largest integer for which $f_i(p_{\text{in}}, p_0) \leq p_k$. This way $\hat{f}_i$ is a pessimistic approximation of $f_i$, hence an admissible gear-shift decoder based on $\hat{f}_i$’s is guaranteed to achieve $p_t$.

We form a trellis whose states are the elements of $\mathcal{P}$ and whose branches are formed by $\hat{f}$. From each state we have $N$ branches associated with the $N$ available decoding algorithms. Each branch, associated with algorithm $i$, connects state $p_m$ in time $T$ to state $p_k$ in time $T + 1$ if and only if $\hat{f}_i(p_{\text{in}}) = p_k$. The weight of such a branch is $t_i$. We refer to this trellis as the “trellis of decoding performance”.

Any path in this trellis which starts at $p_0$ and ends at $p_n$ defines an admissible gear-shift decoder whose computation time is the sum of the weights of branches on that path. The optimum gear-shift decoder is associated with the path of minimum weight, which can be found by dynamic programming.

It should be mentioned that the number of branches in this trellis is proportional to $N$ and $n$ so increasing $n$, in order to reduce the quantization error, linearly increases the number of trellis branches. As a result having even thousands of states can be effectively handled.

Notice that we are interested only in the minimum weight path from $p_0$ to $p_n$, hence we can remove all branches that connect a state $p_m$ at time $T$ to a state $p_k$ at time $T + 1$ when $p_k \geq p_m$. This further simplifies the dynamic program. An example trellis of decoding performance is shown in Fig. 2.

**D. Convergence threshold of the gear-shift decoder**

For equally complex algorithms, it is clear that the convergence threshold of the gear-shift decoder is equal to or better than the best of the available decoders. Note that the decoder chooses the most open EXIT chart at each iteration. The resulting EXIT chart is then the lower hull of all the EXIT charts and hence has an equal or even better threshold.

In the case of algorithms with different complexities, the optimum gear-shift decoder chooses the algorithms based on their complexity and performance. Hence, it might use an algorithm with a worse performance due to its lower complexity. Therefore, it is not obvious how this might affect the threshold of convergence. The following theorem shows
that even in this case, the threshold of convergence can only be improved by gear-shift decoding.

**Theorem 2:** The convergence threshold of the gear-shift decoder is better than or equal to the best of the available decoding algorithms.

**Proof:** Let us denote the best convergence threshold of available algorithms with $p_0^*$. We need to show that when the channel condition is $\epsilon$ better than $p_0^*$, the optimum gear-shift decoder can converge successfully.

Since $p_0$ is the convergence threshold of at least one algorithm, say algorithm $m$, at least this algorithm has an open EXIT chart at $p_0^* - \epsilon$. In other words, for all $p_{in}$, $p_n \leq p_{in} \leq p_0^* - \epsilon$ we have $p_{out} = f_m(p_{in}, p_0^* - \epsilon) < p_{in}$. Therefore, in the trellis of decoding performance, there is a path between $p_0$ and $p_n$ and hence there is a path of minimum weight between them.

Notice that if for some $p_{in}$, all decoding rules except algorithm $m$ have a closed EXIT chart, they all result in a $p_{out} > p_{in}$ Therefore, in the trellis of decoding performance, we can remove the branches associated with them at this particular $p_{in}$. Hence, the optimum gear-shift decoder chooses the only existing branch, i.e., algorithm $m$, towards convergence. This is also true when other decoding rules have a tight EXIT chart. As we see in some examples, in tight regions the optimum gear-shift decoder uses more complex decoders.

**IV. Examples**

In this section we show the effect of gear-shift decoding on the decoding latency through some examples. The available algorithms are assumed to be: sum-product, min-sum, Algorithm E and Algorithm B.

Algorithm E and its update rules can be found in [1], but the version that we use here has a fixed weight of $w = 1$ for the intrinsic message. In this algorithm the message alphabet is $\{ -1, 0, +1 \}$. A $-1$ message can be thought as a vote for a '1' bit, a $+1$ message is a vote for a '0' bit and a 0 message is an undecided vote. Assuming that the all-zero codeword is transmitted, the error rate of this algorithm can be defined as $Pr(-1) + \frac{1}{2} Pr(0)$ to be used for plotting the EXIT chart.

Another issue is the compatibility issue. Since the message alphabet of the above mentioned algorithms are not the same, when transition from one algorithm to another occurs, we have to make the output of one algorithm compatible with the input of the next algorithm. Table I shows how we change the messages at the output of one algorithm for different transitions. The "not-compatible" entries refer to transitions that are not allowed. We avoid transitions from hard algorithms E and B to soft algorithms because it requires us to use a new EXIT chart for the soft algorithms as they are now fed with hard information. The performance of the soft decoder after such transitions is very sensitive to the mapping used for messages. In Example 2, we allow such transitions to investigate their impact on the decoder’s performance.

**Example 1:** In this example we consider a (3,6) regular LDPC code, used to achieve a message error rate of $10^{-6}$ on a Gaussian channel with SNR=1.75 dB. This is almost 0.65 dB and 0.05 dB better than the threshold of this code under sum-product and min-sum decoding respectively. It is also more than 1.3 dB and 3.1 dB worse than the threshold under Algorithms E and B respectively.

Figure 3, shows the EXIT charts of this code under sum-product, min-sum and Algorithm E decoding. It also shows a decoding trajectory using different decoding algorithms. To avoid confusion, each EXIT chart is plotted only in the regions in which it has an open decoding tunnel. The EXIT chart of Algorithm B is closed everywhere and hence is not plotted. Based on our implementations for a (3,6) code, one iteration of sum-product, min-sum and Algorithm E takes 4.1 $\mu$s/b, 1.7 $\mu$s/b and 0.5 $\mu$s/b respectively. These numbers arrived at using a Pentium IV processor clocked at 1 GHz and are intended as illustrations only. Quantizing $p_{in}$ from $p_0 = 0.1106$ to $p_1 = 10^{-6}$ in 4000 points uniformly spaced in a log scale, the optimum gear-shift decoder is: 2 iterations of min-sum followed by 5 iterations of sum-product followed by 10 iterations of min-sum. The whole decoding takes 40.9 $\mu$s/b.

Using only sum-product, a routine density-evolution analysis shows that 15 iterations are required to achieve the same message error rate and hence the decoding takes a total of 61.5 $\mu$s/b. This is more than 50% longer than the optimum gear-shift decoder. Using only min-sum, we need 31 iterations which takes 52.7 $\mu$s/b, which is more than 28% longer than the optimum gear-shift decoder.

Another important benefit of the gear-shift decoder over min-sum decoder is the decoding threshold. If the channel SNR is over estimated by only 0.05 dB, the min-sum decoder would fail to converge, no matter how many iterations of decoding we allow. However, as Fig. 3 suggests, when the decoding tunnel of the min-sum decoder is very tight (or even closed due to over estimation of channel SNR), the gear-shift decoder switches to sum-product. Hence, it is more robust to channel estimation, while its decoding latency is also significantly less.

**Example 2:** Consider the previous example, but this time
we resolve the “not compatible” cases. To do so, we map a message \( m \) in Algorithm E (Algorithm B) to \( K \cdot \text{sign}(m) \) \((K \cdot (2m - 1))\) to be used in sum-product or min-sum algorithms.

Choosing \( K = 10 \), the result for the optimum gear-shift decoder is 3 iterations of min-sum followed by 4 iterations of sum-product followed by another 7 iterations of min-sum, followed by 4 iterations of Algorithm E and finally followed by another 3 iterations of min-sum. The decoding time is 40.5 \( \mu \text{s}/\text{b} \), which is only 1% better than that of the previous example. Since for variable nodes of degree 3, Algorithm E and B both have poor performance even at low extrinsic message error rates, they did not have an impact on the speed of the gear-shift decoder. Nevertheless, the gear-shift decoding by only sum-product and min-sum is dramatically better than using single-algorithm decoding.

**Example 3:** Consider the example studied in the introduction section, but allow only sum-product and min-sum decoding. The optimum decoder is one iteration of min-sum followed by 4 iterations of sum product followed by another 5 iterations of min-sum, which takes 37.6 \( \mu \text{s}/\text{b} \). This is 40% longer than the optimum gear-shift decoder using all 4 algorithms. Comparing with the case of the (3,6) code, it shows that for a (4,8) code the hard decoding rules can have a significant impact on the speed of the optimum gear-shift decoder. This is also true for higher degree variable nodes, since the EXIT chart of Algorithm B for variable nodes of degree 4 and larger approaches zero with a slope of zero [12].

**V. CONCLUSION**

We introduced gear-shift decoding, a method for reducing the decoding latency of iterative decoders without sacrificing the code rate. In this method, the decoder changes its decoding rule during the process of decoding.

Given a set of decoding rules, we showed that the optimum gear-shift decoder for minimum decoding-latency can be found using dynamic programming. We also proved that the convergence threshold of the optimum gear-shift decoder is better than or equal to the best convergence threshold of the available algorithms. Through some examples we showed that, gear-shift decoding can significantly reduce decoding latency.

The idea of gear-shift decoding is very general and can be used for other iterative decoders. Gallager’s Algorithm B is a good example of gear-shift decoding for irregular LDPC codes [12], [14]. However, when the complexity of different algorithms varies, it is not clear how an (irregular LDPC code, gear-shift decoder) pair should be specifically designed for each other. Notice that irregular LDPC codes are usually highly optimized for a certain decoding algorithm. It has been noticed in many papers, e.g. [3], [15], that the EXIT chart of the highly optimized irregular code for a decoding rule is very tight for all \( \mu \text{b}/\text{s} \)’s, so the EXIT chart of a code optimized for sum-product decoding, for example, under any other decoding is closed everywhere, and so gear-shift decoding is not applicable. Nevertheless, for many codes, gear-shift decoding has great potential to reduce decoding complexity.

**REFERENCES**


