On the Convergence of Log-Likelihood Values in Iterative Decoding

Gottfried Lechner and Jossy Sayir
Telecommunications Research Center Vienna (ftw.), A-1220 Vienna, Austria
Email: {lechner|sayir}@ftw.at

***Abstract*— A new method for visualizing the behavior of an iterative decoder is presented. This method is applied to regular and irregular low-density parity-check codes. Several types of decoding errors are investigated and suggestions are made for improving the iterative decoder to avoid such errors. Furthermore, structural properties of parity-check matrices that influence the decoding performance are identified.**

I. Introduction

ITERATIVE decoding is an essential component of modern communication systems. One class of codes that are suited for iterative decoding are low-density parity-check (LDPC) codes introduced by Gallager [1]. They can be described by a sparse parity check matrix that allows the application of the sum-product algorithm. This efficient algorithm passes messages on the associated bipartite graph [2]. The complexity of the sum-product algorithm grows linearly with the block length. Therefore it is possible to apply the algorithm to large block codes.

Much work has been done to analyze LDPC codes, especially for essentially infinite block lengths and ensembles of codes. In practical applications, the block length is limited and many of the assumptions made for infinite block lengths are not valid. This paper concentrates on the convergence behavior of the decoder for practical block lengths (about 1,000 to 10,000 bits). This behavior is visualized by animations, that illustrate the convergence of the sum-product algorithm in a graphical way. The aim of this work is to show the dependencies between graph properties and decoder convergence and to derive guidelines for the construction of LDPC codes.

The rest of this paper is organized as follows: first, the iterative decoding procedure is described in section II. In section III, we describe the method used for the graphical representation. Section IV and V applies this method to regular and irregular codes, respectively. Section VI gives an overview of the decoding errors that occur.

II. Iterative Decoding of LDPC Codes

LDPC codes can be decoded by applying the sum-product algorithm to the corresponding factor graph. The factor graph for a parity-check matrix is a bipartite graph[1], where every digit of the codeword is represented by a variable node and every parity check equation is represented by a check node. A connection between those nodes is made if and only if the digit of the codeword participates in the parity-check equation. A detailed description of factor graphs and the sum-product algorithm can be found in [2].

The representation of a parity-check matrix as a factor graph is unique, but a code can be described by many parity check matrices. A different matrix can be constructed by applying elementary row operations to the original matrix. The iterative decoder depends on this specific representation of the code. Structural parameters of the parity-check matrix, like column weight, cycle distribution and diameter of the corresponding graph influence the performance of the iterative decoder. In contrast, maximum a-posteriori (MAP) decoding is independent of the particular parity-check matrix. The performance of a MAP decoder depends on the distance spectrum—a property of the code. The performance of the iterative decoder depends on the properties of the parity-check matrix (the factor graph) as well as on the properties of the code. Because of the equivalence of parity check matrices and factor graphs, terms like column weight and variable-node degree are equivalent. We will use those terms interchangeably in the rest of this paper.

We tried to analyze the influence of the following parameters:

- the *column weight*. It is equal to the number of parity-checks in which one digit (column) is involved. Large column weight means that the corresponding digit of the codeword receives many messages after a small number of iterations. Intuitively, this digit is expected to converge faster.
- the *girth distribution*. Each variable node can be contained in one or more cycles. If a variable node is not directly involved in a cycle but connected via a path to a cycle, this node is involved in an *extended cycle*. The length of this extended cycle equals the length of the original cycle increased by the double path length. Figure 1 shows an example of such an extended cycle: variable nodes 2 and 3 are directly involved in a cycle of length 4. Variable node 1 is not directly involved in a cycle, but is connected to a cycle. Messages sent out by variable node 1 come back after passing 6 edges. By the extended definition of a cycle, we assign this node a cycle length of 6.

This work is part of Gottfried Lechner's diploma thesis at the Technical University of Vienna supervised by Prof. H. Weinrichter.

[1] A bipartite graph consists of two types of nodes, where no connection exist between nodes of the same type.

The *local girth* with respect to a node is defined as the smallest cycle in which the node is involved. By assigning every node a local girth, we can calculate the *girth distribution* of the graph.
The minimum local girth and the mean value of the girth distribution should be as large as possible to allow many independent iterations.

- the *local diameter*. We also extend the definition of the diameter of a graph to the diameter with respect to a node—called *local diameter*. This is the maximum path length needed to reach every other node in the graph. The local diameters should be small. If they are small, every node receives messages from every other node after a few iterations.

Clearly, these three parameters can not be optimized independently. For example, if the column weight is increased, the length of cycles decreases.

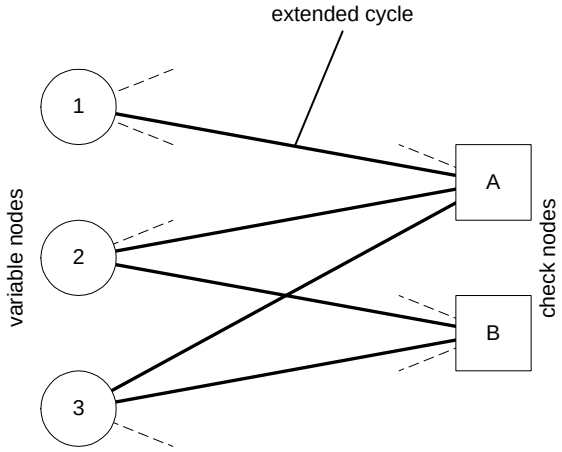

Fig. 1. extended cycle

The sum-product algorithm calculates the exact a-posteriori probabilities (APP) after a finite number of iterations if the factor graph is free of cycles (the number of iterations equals the diameter of the graph). Unfortunately, codes that can be represented by graphs with no cycles are poor codes in the sense of having a small minimum distance as shown in [3]. By applying the sum-product algorithm to graphs with cycles, the result is not equal to the APP. The algorithm does not terminate after a finite number of iterations and the estimated APP generally converge to zero or one. However, the performance of the decoder is still excellent—in the sense of a low bit error rate at transmission rates near capacity.

III. Visualization of the Decoding Process

After every iteration of the decoder, the current estimation of the log-likelihood ratio (LLR) of every bit is calculated by summing up all the incoming messages. The true log-likelihood ratios are defined as

$$L(x_i) = \log\left(\frac{P(x_i = 0|\boldsymbol{y})}{P(x_i = 1|\boldsymbol{y})}\right), \qquad (1)$$

where x_i is the i-th digit of the transmitted codeword, and $\boldsymbol{y}$ is the sequence received at the output of the channel.

The representations used in the following sections show the code digits on the horizontal and the estimated log-likelihood ratios on the vertical axis. For every iteration, such a figure is drawn and the sequence of the iterations is merged into an animation. This method gives a good insight into the behavior of the decoder. In this paper, we provide snapshots of the animations. All simulations are made by transmitting the all zero word, without loss of generality due to the linearity of the code. Therefore, a positive LLR represents a correct decision while a negative LLR represents a bit error.

IV. Regular Codes

All the columns in a regular LDPC code have the same weight. It is expected that the convergence behavior is equal for every variable node. At the beginning of the decoding process, every variable node sends its associated received message to the check nodes. If the graph is free of cycles of length c, then the first $\frac{c}{2}$ iterations are correct, i.e. no message is used twice. When the cycle length is reached, the cycle builds a positive feedback and the estimated LLR tends to infinity. Figure 2 shows the received LLR values before the decoder starts. Figure 3 shows the estimated LLR values after 5 iterations and figure 4 after 9 iterations. The code used has a block length $n = 1,000$, rate $R = 0.5$ and column weight $d_v = 3$. This simulation was made at a signal to noise ratio of 2.5dB.

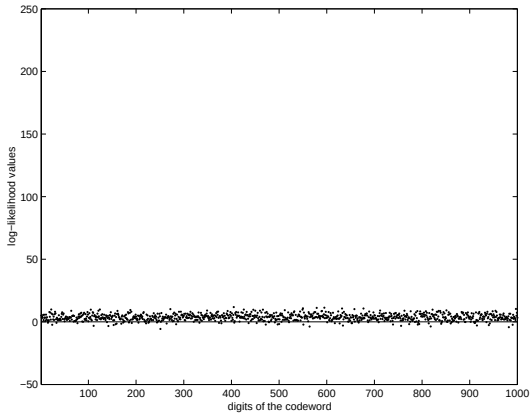

Fig. 2. regular code before decoding

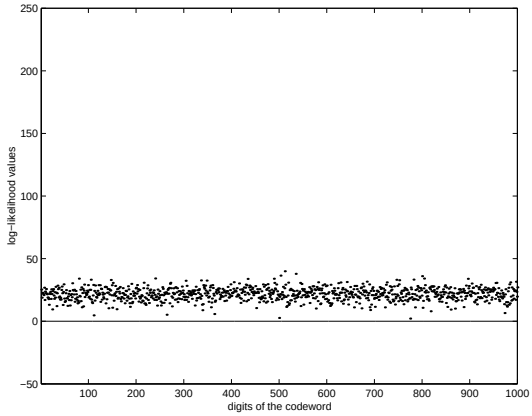

Fig. 3. regular code after 5 iterations

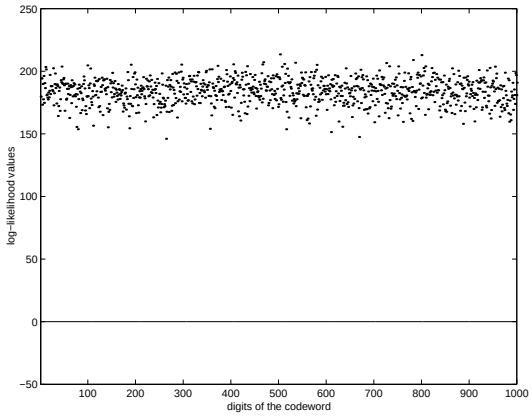

Fig. 4. regular code after 9 iterations

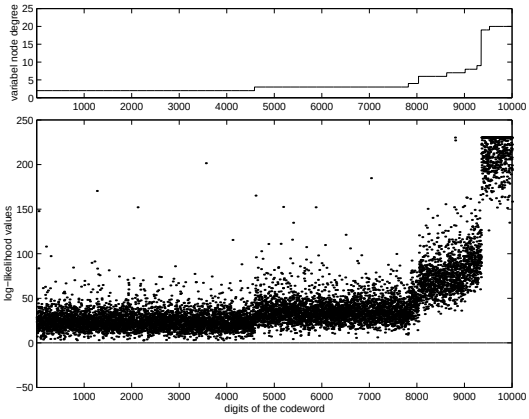

Fig. 6. irregular code after 30 iterations

V. Irregular Codes

In contrast with regular codes, irregular codes do not have a constant column weight. The number of variable nodes of a specific degree depends on the distribution function as described in [4]. The intuitive justification for using irregular codes is that variable nodes with higher degree will converge faster. These so called *elite nodes* will then provide reliable information to the other (low degree) nodes. We constructed an irregular code with block length $n = 10,000$, rate $R = 0.5$ and maximum variable degree $d_v = 20$, using the degree distribution provided in [4]. The variable nodes are ordered from left to right by degree, starting with the lowest. Figure 5 shows the estimated LLR values after 15 iterations and figure 6 after 30 iterations, at a signal to noise ratio of 1.0dB. The degree of the variable nodes is also shown in the figures above the animation snapshot.

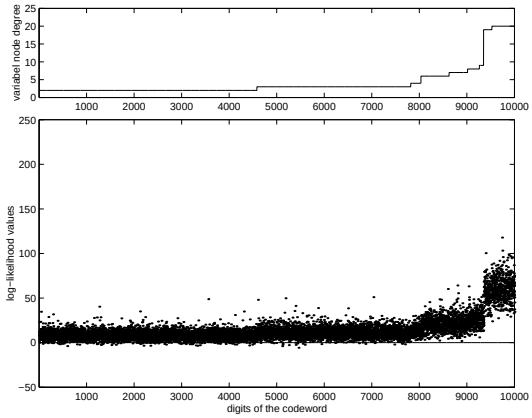

Fig. 5. irregular code after 15 iterations

The outer right nodes (degree 20) converge very fast, while the left nodes (degree 2) converge very slowly, confirming the previous considerations. (The ceiling appearing at the high degree nodes in figure 6 is caused by numerical limitations.) It should be noted that the probability of a bit error is higher for nodes with low degree. Therefore the systematic bits should be mapped onto the high degree nodes and the parity bits onto the nodes with lower degree!

VI. Decoding Errors

The animation of the decoding procedure can be used to observe the behavior of the decoder in the case of a decoding error. These observations can be used to determine which of the parameters of the system must be changed in order to improve the performance. Three kinds of errors were observed:

- *convergence to a wrong codeword*. The probability of this type of error is increased if the minimum distance of the code is small. The minimum distance is a property of the code and therefore this type of error can not be avoided by improving the decoder, i.e., using another parity-check matrix for the representation of the code. The minimum distance grows when the block length is increased. The probability for this type of error tends to zero for sufficiently long blocks. For block lengths larger or equal to 1,000 bits, we were not able to observe this type of error. For smaller block lengths (100 bits) we observed several errors of this type. In all cases, we found that the APP of the decoded codeword was higher than the APP of the transmitted codeword. Therefore, a maximum likelihood sequence decoder would also have failed in these cases.
- *convergence to a local maximum*. The estimated LLR values converge to an equilibrium which is not a codeword. Even though the graph contains cycles, the estimated LLR values do not tend to infinity.
- *oscillating LLR values*. The LLR values do not converge to an equilibrium. This could be caused by two or more cycles through one node. When the messages looping in the cycles are different (positive and negative) and arrive at different times at the variable node, the LLR value of this node is oscillating. This type of error could be avoided by modifying the scheduling algorithm used in the iterative decoder. In [5], a probabilistic scheduling algorithm is proposed and said to improve the bit error rates.

VII. Conclusion

We made several simulations and provided an illustration for the convergence behavior of an iterative decoder. Some parameters for iterative decoding systems were introduced and explained. The animations were used to confirm some intuitive considerations (specially for irregular codes). Furthermore, we

identified three types of decoding errors. Further research will be done to derive design guidelines from the observations of decoding errors and to precise the influence of the properties of the graph.

Simulation results, animations of the decoding process and more information about the progress of this research can be found on `http://www.ftw.at/ldpc`.

REFERENCES

[1] R. G. Gallager. *Low Density Parity Check Codes*. Number 21 in Research monograph series. MIT Press, Cambridge, Mass., 1963.

[2] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2):498–519, 2001.

[3] T. Etzion, A. Trachtenberg, and A. Vardy. Which codes have cycle-free tanner graphs? *IEEE Trans. Inform. Theory*, 45(6):2173 – 2180, September 1999.

[4] T. Richardson, A. Shokrollahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47:619–637, February 2001.

[5] Yongyi Mao and A.H. Banihashemi. Decoding low-density parity-check codes with probabilistic scheduling. *IEEE Commmunications Letters*, 5:414 – 416, October 2001.