

Randomised Construction and Dynamic Decoding of LDPC Codes

Joakim G. Knudsen and Matthew G. Parker

{joakimk, matthew}@ii.uib.no

University of Bergen, Selmer Centre, 2006

http://www.ii.uib.no/~joakimk/msc_knudsen.pdf

Abstract

The thesis [1] explores certain methods for the construction and decoding of Low-Density Parity-Check (LDPC) codes. To this end, we have developed a software package (written in C++) to implement and construct (design) these codes, to simulate the encoding and decoding of information using these codes, and to experiment with various new ideas; particularly with respect to decoding.

The focus of this text will be on the concept of error-correction and decoding. Details on LDPC construction can be found in the thesis.

1. Introduction

Low-Density Parity-Check (LDPC) codes were originally invented by Gallager in his 1963 thesis [2]. These optimum codes were the first to achieve Shannon's 1948 channel capacity—a theoretical performance bound which remains undisputed today. However, the hardware available at the time could not meet the requirements, and the codes were largely forgotten until rediscovered by Tanner as late as the 1980's [3]. The codes proved extremely apt to modern networking needs, and compete with Turbo codes in the resulting performance race towards capacity [4]. Recently, LDPC codes were elected the new standard for satellite broadcast, high-definition TV (DVB-2 and IEEE802.3an).

An LDPC code is essentially a random matrix, which is conventionally realized as a bipartite graph where this is convenient. Since these identically represent one and the same code, we shall use both descriptions in this text—in all cases referring to the underlying *code*. Fig. 1 shows a toy LDPC code.

1.1. Forward Error Correction

When information is conveyed across any medium—either transmitted across a physical or wireless channel, or written or read from a disk—it is subject to natural disturbances, or noise. Clever means must be applied at the source to facilitate identification and/or correction of (some) errors at the

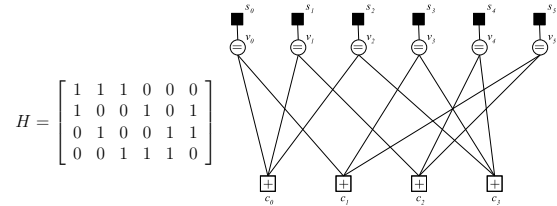


Figure 1: A toy LDPC code. The received vector is attached to 'input nodes,' s_i , while v_i are bit nodes and c_i check nodes.

receiving end. In an encoding procedure, parts of the information is repeated via a specific mathematical transformation, and sent along with the message. This redundancy is the key element in Forward Error Correction (FEC), which is an important field of research within Coding Theory (see [5] for a good introduction).

At the source, q -ary bits of information are modulated to distinguishable peaks of energy (channel symbols), so as to facilitate electrical transmission as a signal. Channel noise may deflect waves off-peak, such that the received vector (after sampling the signal) consists of real numbers spanning over a floating scale. Since less noise is more likely than more noise, this so called *soft information* can be thought of as an estimate of the corresponding bit-value. In short, these estimates are the complementing probabilities that a certain bit, v_i , is 0 or 1, given the corresponding channel symbol, y_i :

$$Pr(v_i = 0 | y_i) + Pr(v_i = 1 | y_i) = 1.$$

Traditionally, FEC has been dominated by complex, algebraic codes, which are typically decoded in one stage, immediately producing a decoder output. The 1993 discovery of Turbo codes sparked renewed interest in iterative coding and the concept of soft information.

In the algebraic setting, before decoding, the vector is immediately quantized back to q -ary bits by simply demodulating floats to their nearest energy peak; i.e., 'trusting' the estimates. This is known as hard decision quantization. The influence of noise may lead to wrong decisions, or bit-errors, which are hopefully corrected by the decoder.

1.2. Soft-Input, Soft-Output Decoding

Rather than discarding valuable gradient information by premature quantization, SISO (Soft Input, Soft Output) decoders benefit from operating directly on these estimates (channel symbols). Often, SISO decoders operate by iterating on a graphical structure, or network, in which pieces of soft information flow and interact according to rather simple, local decoding rules. When some stopping criterion is satisfied, the decoder outputs a hard decision on the final state. Our results add to the well-established performance gain of soft vs. hard decoding.

Iterative decoding algorithms typically employ some form of majority logic, where the global problem of identifying the received symbols most likely in error, is approached in a distributed manner. Within the receiver (decoder), a check node is *satisfied* if the sum of its adjacent bits equal zero, modulo 2. When all checks are satisfied, a valid codeword (which may still differ from the codeword intended by the source) is found in the bit-nodes. Hence, checks are called *constraint nodes*, and realize the actual decoding rules within the structure of the graph.

The Sum-Product Algorithm operates by passing messages on the edges of this graph, typically using 'Flooding' scheduling.¹ Each bit holds some estimate of its correct value, with some estimates less reliable (due to noise) than others. After sending out its current estimate on each connected edge, a bit receives in return a value from each direction—the product of which becomes the new estimate for that bit. Input from edge e is not allowed to affect the output along that same edge. This *extrinsic principle* is meant to counteract *feedback* where nodes 'influence with themselves,' ensuring that estimates converge towards a valid codeword—given moderate noise and a *good code* (e.g., few short cycles).

In this fashion, each check node attempts to adjust the local values of its set of adjacent bits in such a way that this check itself may be satisfied. However, since these sets of bits overlap (see Fig. 1), the process can be thought of as the balancing of conflicting interests towards a common goal: a valid codeword.

2. Our Results: Dynamic Decoding

Perhaps the most important insight from the Master's project is that the Sum-Product Algorithm (SPA) is very robust. This makes for an excellent framework for experimenting with new ideas within LDPC decoding. Using simulations on one and the same code,² we could com-

¹One Flooding iteration consists of the update of all bit nodes followed by all check nodes, thus creating an alternating 'wave' of information through the graph.

²Blocklength $N = 90$, redundancy $m = 60$, column and row weights 3 and 6, respectively, and, girth is 6 (no length-4, or shorter cycles in graph). This code was generated by our software, as based on [6].

pare the performance of novel decoder schemes to the standard, near-optimum 'Flooding' performance. We define Dynamic Decoding as the combination of selective (i.e., non-Flooding) updating of nodes, and/or structural change in the decoder graph—during decoding.

Important observations were that SPA is not dependant on Flooding—or any other particular node updating *order*—in order to converge, so long as the information is sufficiently propagated throughout the graph. Also, we learned that loss of extrinsic (soft) information during decoding is not necessarily critical to convergence.

2.1. Local Decisions

Individual nodes in the graph can be updated in a more selective manner, using local decision rules (see also [7]). In this sense, the standard Flooding schedule may be considered redundant and inefficient.

SISO decoding is sensitive to feedback in the underlying graph. The effect of the extrinsic principle is corrupted by the presence of cycles in LDPC graphs. However, acyclic graphs (tree codes) have poor performance (low *distance* between codewords, 'confusing' the decoder), so one typically tries to avoid or reduce the occurrence of short cycles during code construction—in particular, cycles of length 4.

By maintaining simple counters within nodes, we suggested a local updating rule which—in every iteration—identifies the nodes that would cause feedback in the graph, if updated in the next iteration. In other words, if you have a cycle: *don't go around it*. Fig. 2 shows performance compared to Flooding. Since 'avoid-4' iterations are local operations (does not update all nodes in every iteration) the simulation has been prolonged to produce comparably significant results.

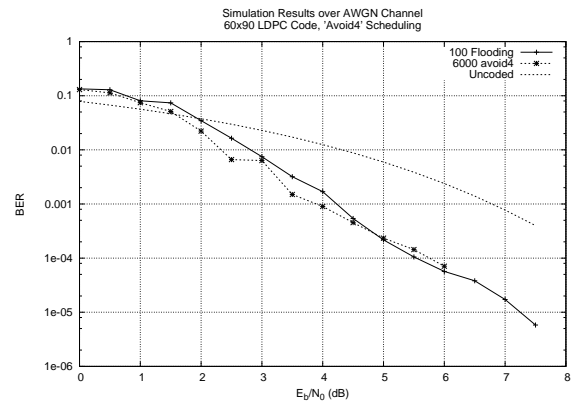


Figure 2: Avoiding 4-cycles, as compared to (max) 100 Flooding iterations. Some gain (drop in BER) is observed at the low-SNR (high noise) end.

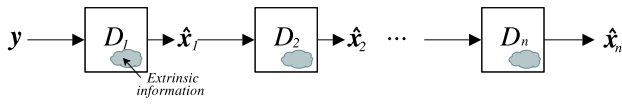


Figure 3: The conceptual serial decoder: the noisy channel vector, y , is decoded in n stages, with \hat{x}_n as the final output. Local distributions of extrinsic information are discarded between stages.

2.2. Serial Decoding

The observation on tolerance to loss of soft information derived from the idea of using several identical decoders, working in series; see Fig. 3. After running a fixed number of iterations, the final decoding state of decoder D_i becomes the input to D_{i+1} . The last decoder thus produces the final decoder output. This can be simulated using only one decoder, simply by resetting (neutralising) the extrinsic information; namely the messages currently on the edges.

Simulations were performed to observe the effects on the overall convergence, by resetting at regular intervals during decoding—see Fig. 4(a). Somewhat surprisingly, the results conclude that SPA successfully manages to maintain near-optimum performance, so long as the information content is not disturbed too frequently.

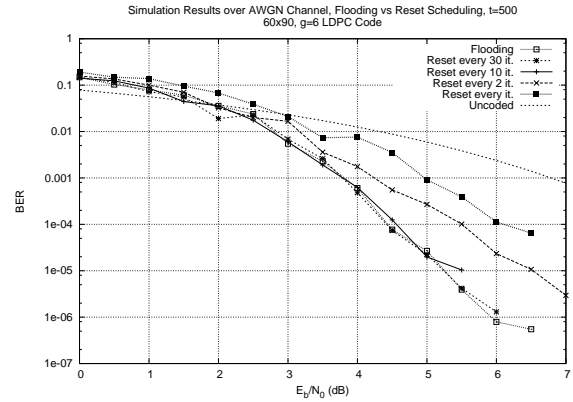
Fig. 4(b) shows a detailed view of the convergence *process* at SNR 3dB. The number of bit-errors remaining at iteration i is averaged against the total number of errors, producing a Density Evolution plot [8]. Flooding stabilises after an average of 120 iterations, so the left part of the plot illustrates the effect of disturbing the decoder before it has performed enough Flooding iterations to completely converge. Note the regular spikes in response to the decoder being reset.

2.3. Pivot

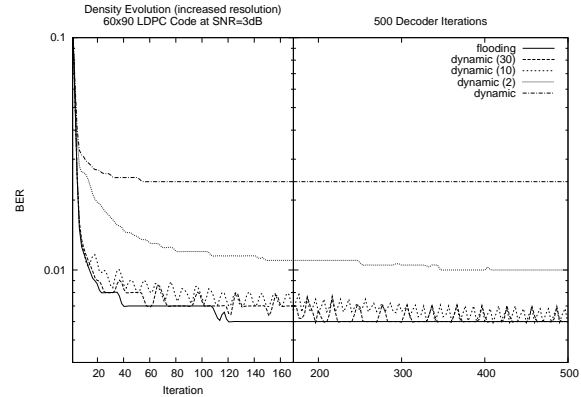
The operation of pivot is a local transformation (operating on an edge), transforming the structure of the graph while maintaining code equivalence. We may assume the LDPC graph is always bipartite, thus simplifying the description of pivot.

If 'local neighbourhood,' $n(u)$, is the nodes adjacent to node u , pivoting on edge (u, v) equals complementing the edge space between $n(u) \setminus \{v\}$ and $n(v) \setminus \{u\}$.³ By pivoting during decoding, we envisage the potential for performance gain due to the new independent paths of information created by pivot.

³This is the graphical equivalent of the matrix operation of the same name used in Gaussian reduction; where—obviously—the solution space of a set of linear equations is preserved.



(a) Bit-Error Rate, simulated over (max) $t = 500$ iterations.



(b) 'Dynamic (n)' refers to resetting the decoder every n 'th Flooding iteration.

Figure 4: Serial Decoding of a small LDPC code

Our initial ideas of dynamic decoding were often hindered by difficult bookkeeping concerns of how to ensure the validity of the extrinsic information content (on edges) during experimental decoding. For instance, if edges are removed (or inserted), what do we do with the presence of 'undelivered messages'? If an edge is redirected to a different node; is the previous recipient's message only irrelevant to the new destination, or even harmful?

The positive results of Serial Decoding is apparently an effective workaround for this concern, and leads to the interesting question of how can we further abuse the decoder when sensitive content is already stored safely within vertices?

Our early results indicate that the robust Sum-Product Algorithm is at least able to converge (if not improve) on a graph that is 'rotated' by pivot—see Fig. 5. Related ideas, by using overdefined codes, has recently been reported successful in [9].

3. Further Studies

The experimental nature of the Master's project has revealed many interesting and promising directions for further

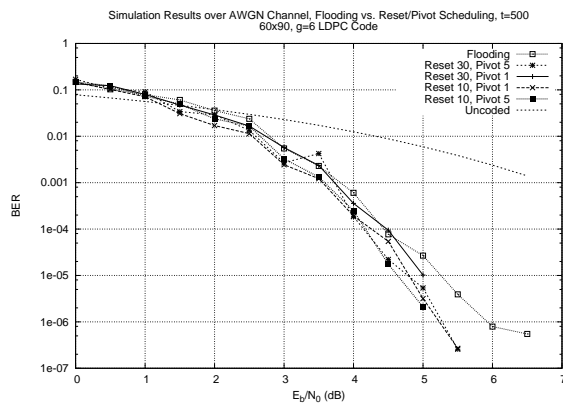


Figure 5: Effects of random pivot operation(s) while re-setting graph. 'Reset m , Pivot n ' refers to resetting the decoder every n 'th Flooding iteration, and performing m (random) pivot operations before resuming decoding. The results trace those of flooding quite closely, while suggesting a slight improvement at the high-SNR (low noise) end.

study in my PhD work. For instance, further negative consequences of pivot need to be worked out before this idea can be used in decoding. Since LDPC codes are only optimal when sparse,⁴ edge complementation due to random (unrestricted) pivot must create more edges than it removes, until at some point the density naturally stabilises at around 0.5. An important result from the thesis is that selective application of pivot (based on local decisions) enables us to limit and control the increase in density—details in [1].

Also, in real life, there are environments with networks that similarly change in structure over time, for instance mobile networks where each roaming cellular phone is viewed as a node. Using technology adapted to structural change (e.g., pivot scheduling), we suggest to interleave new functionality by message-passing on this implicit graph of telephones and base stations. Many unanswered questions of context and feasibility currently obscure this novel technology.

Additional ideas related to Dynamic Decoding are under investigation, but which are beyond the scope of this brief text. We thank the reader for taking an interest in our work!

References

- [1] J. G. Knudsen. Randomised construction and dynamic decoding of LDPC codes. Master's thesis, University of Bergen, 2006.

⁴Density well below 0.5 [10].

- [2] R. G. Gallager. *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
- [3] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27:533–547, 1981.
- [4] E. Guizzo. Closing in on the perfect code. *IEEE Spectrum*, 2004.
- [5] Shu Lin and Jr. Daniel J. Costello. *Error Control Coding*. Pearson, Prentice Hall, 2004.
- [6] J. Campello and D. Modha. Extended bit-filling and ldpc code design. *IEEE Trans. Inform. Theory*, 1:985–989, 2001.
- [7] H. Xiao and A. Banihashemi. Graph-based message-passing schedules for decoding ldpc codes. *IEEE Trans. on Commun.*, 52:2098–2105, 2004.
- [8] T. J. Richardson, A. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47:619–637, 2001.
- [9] T. R. Halford and K. M. Chugg. Random redundant soft-in soft-out decoding of linear block codes. *ISIT 2006, Seattle, USA, July 9 - 14*, pages 2230–2234, 2006.
- [10] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. *Cryptography and Coding 5th IMA Conf.*, pages 100–111, 1995.