Efficient Encoding of Low-Density Parity-Check Codes[*]

Tom Richardson [†] Rüdiger Urbanke [‡]

March 6, 2001

Abstract

Low-density parity-check codes can be considered serious competitors to turbo codes in terms of performance and complexity and they are based on a similar philosophy: constrained random code ensembles and iterative decoding algorithms.

In this paper we consider the encoding problem for low-density parity-check codes. More generally, we consider the *encoding* problem for codes specified by sparse parity-check matrices. We show how to exploit the sparseness of the parity-check matrix to obtain efficient encoders. For the (3,6)-regular low-density parity-check code for example, the complexity of encoding is essentially quadratic in the block length. However, we show that the associated coefficient can be made quite small, so that encoding codes even of length $n \simeq 100,000$ is still quite practical. More importantly, we will show that "optimized" codes actually admit linear time encoding.

Keywords: turbo codes, parity-check, decoding, encoding, sparse matrices, random graphs, binary erasure channel.

1 Introduction

Low-density parity-check (LDPC) codes were originally invented and investigated by Gallager [1]. The crucial innovation was Gallager's introduction of iterative decoding algorithms (or message-passing decoders) which he showed to be capable of achieving a significant fraction of channel capacity at low complexity. Except for the papers by Zyablov and Pinsker [2], Margulis [3] and Tanner [4] the field then lay dormant for the next 30 years. Interest in LDPC codes was rekindled in the wake of the discovery of turbo-codes and LDPC codes were independently rediscovered by both MacKay and Neal [5] and Wiberg [6].[1] The past few years have brought many new developments in this area. First, in several papers Luby, Mitzenmacher, Shokrollahi, Spielman and Stemann

[*]This work was performed while both authors were at Bell Labs.

[†]Flarion Technologies, Bedminster, NJ 07921, USA, email: `richardson@flarion.com`

[‡]EPFL, LTHC-DSC, CH-1015 Lausanne, email: `Rudiger.Urbanke@epfl.ch`

[1]Similar concepts have also appeared in the physics literature [7, 8].

introduced new tools for the investigation of message-passing decoders for the binary erasure channel (BEC) and the binary symmetric channel (BSC) (under hard decision message-passing decoding) [9, 10], and they extended Gallager's definition of LDPC codes to include *irregular* codes, see also [5]. The same authors also exhibited sequences of codes which, asymptotically in the block-length, provably achieve capacity on a BEC. It was then shown in [11] that similar analytic tools can be used to study the asymptotic behavior of a very broad class of message passing algorithms for a wide class of channels and it was demonstrated in [12] that LDPC codes can come extremely close to capacity on many channels.

In many ways LDPC codes can be considered serious competitors to turbo codes. In particular, LDPC codes exhibit an asymptotically better performance than turbo codes and they admit a wide range of trade-offs between performance and decoding complexity. One major criticism concerning LDPC codes has been their apparent high *encoding* complexity. Whereas turbo codes can be encoded in linear time, a straightforward encoder implementation for a LDPC code has complexity quadratic in the block length. Several authors have addressed this issue:

1. It was suggested in [13] and [9] to use cascaded rather than bipartite graphs. By choosing the number of stages and the relative size of each stage carefully one can construct codes which are encodable and decodable in linear time. One drawback of this approach lies in the fact that each stage (which acts like a subcode) has a length which is in general considerably smaller than the length of the overall code. This results, in general, in a performance loss compared to a standard LDPC code with the same overall length.

2. In [14] it was suggested to force the parity check matrix to have (almost) lower triangular form, i.e., the ensemble of codes is restricted not only by the degree constraints but also by the constraint that the parity check matrix have lower triangular shape. This restriction guarantees a linear time encoding complexity but, in general, it also results in some loss of performance.

It is the aim of this paper to show that, even without cascade constructions or restrictions on the shape of the parity check matrix, the encoding complexity is quite manageable in most cases and provably linear in many cases. More precisely, for a $(3,6)$-regular code of length n the encoding complexity seems indeed to be of order n^2 but the actual number of operations required is no more than $0.017^2 n^2 + O(n)$, and, because of the extremely small constant factor, even large block lengths admit practically feasible encoders. We will also show that "optimized" irregular codes have a *linear* encoding complexity and that the required amount of preprocessing is of order at most $n^{3/2}$.

The proof of these facts is achieved in several stages. We first show in Section 2 that the encoding complexity is upper bounded by $n + g^2$, where g, the *gap*, measures in some way to be made precise shortly, the "distance" of the given parity check matrix to a lower triangular matrix. In Section 3 we then discuss several greedy algorithms to triangulate matrices and we show that for these algorithms, when applied to elements of a given ensemble, the gap concentrates around its expected value with high probability. As mentioned above, for the $(3, 6)$-regular code the best greedy algorithm which we discuss results in an expected gap of $0.017n$. Finally, in Section 4 we prove that for all known "optimized" codes the expected gap is actually of order less than $\sqrt{n}$, resulting in the promised linear encoding complexity. In practice the gap is usually a small constant. The $\sqrt{n}$ bound can be improved but it would require a significantly more complex presentation.

We finish this section with a brief review of some basic notation and properties concerning LDPC codes. For a more thorough discussion we refer the reader to [1, 11, 12].

Low-density parity-check codes are linear codes. Hence, they can be expressed as the null space of a *parity check* matrix H, i.e., x is a codeword if and only if

$$Hx^T = 0^T.$$

The modifier 'low-density' applies to H; The matrix H should be sparse. For example, if H has dimension $\frac{n}{2} \times n$, where n is even, then we might require H to have 3 ones per column and 6 ones per row. Conditioned on these constraints we choose H at random as discussed in more detail below. We refer to the associated code as a $(3, 6)$-regular LDPC code. The sparseness of H enables efficient (sub-optimal) decoding, while the randomness ensures (in the probabilistic sense) a good code [1].

Example 1. [Parity-Check Matrix of $(3, 6)$-Regular Code of Length 12] The following matrix H will serve as an example.

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}. \quad (1)$$

$\square$

In the theory of LDPC codes it is customary and useful not to focus on particular codes but to consider ensembles of codes. These ensembles are usually defined in terms of ensembles of *bipartite graphs* [13, 15]. E.g., the bipartite graph which represents the code defined in Example 1 is shown in Fig. 1. The *left* set of nodes represents the *variables* whereas the *right* set of nodes represents the *constraints*. An ensemble of bipartite graphs is defined in terms of a pair of *degree distributions*. A degree distribution $\gamma(x) = \sum_i \gamma_i x^{i-1}$ is simply a polynomial with non-negative real coefficients satisfying $\gamma(1) = 1$. Typically, γ_i denotes the fraction of edges in a graph which are incident to a node (variable or constraint node as the case may be) of degree i. In the sequel, we will use the shorthand $\int\gamma$ to denote $\sum_{i\geq 1} \frac{\gamma_i}{i} = \int_0^1 \gamma(x)\, dx$. This quantity gives the inverse of the average node degree. Associated to a degree distribution pair (λ, ρ) is the *rate* $r(\lambda, \rho)$ defined as

$$r(\lambda, \rho) := 1 - \frac{\int\rho}{\int\lambda}. \tag{2}$$

E.g., for the degree distribution pair (x^2, x^5), which corresponds to the regular $(3, 6)$ Gallager code, the rate is $\frac{1}{2}$.

Given a pair (λ, ρ) of degree distributions and a natural number n, we define an *ensemble* of bipartite graphs $\mathcal{C}^n(\lambda, \rho)$ in the following way. All graphs in the ensemble $\mathcal{C}^n(\lambda, \rho)$ will have *left* nodes which are associated to λ and *right* nodes which are associated to ρ. More precisely, assume that $\lambda(x) := \sum_{i\geq 1} \lambda_i x^{i-1}$ and $\rho(x) := \sum_{i\geq 1} \rho_i x^{i-1}$. We can convert these degree distributions into *node perspective* by defining $\tilde{\lambda}_i := \frac{\lambda_i}{i\int\lambda}$ and $\tilde{\rho}_i := \frac{\rho_i}{i\int\rho}$. Each graph in $\mathcal{C}^n(\lambda, \rho)$ has $n\tilde{\lambda}_i$ left nodes of degree i and $(1 - r(\lambda, \rho))n\tilde{\rho}_i$ right nodes of degree i. The order of these nodes is arbitrary but fixed. Here, to simplify notation, we assume that (λ, ρ) and n are chosen in such a way that all these quantities are integers. A node of degree i has i *sockets* from which the i edges emanate and these sockets are *ordered*. So in total there are

$$s := \sum_{i\geq 1} in\tilde{\lambda}_i = \sum_{i\geq 1} n\frac{\lambda_i}{\int\lambda} = \frac{n}{\int\lambda} = \frac{(1 - r(\lambda, \rho))n}{\int\rho} = (1 - r(\lambda, \rho)) \sum_{i\geq 1} n\frac{\rho_i}{\int\rho} = (1 - r(\lambda, \rho)) \sum_{i\geq 1} in\tilde{\rho}_i$$

ordered sockets on the left as well as on the right. Let σ be a permutation on $[s] := \{1, \cdots, s\}$. We can associate a graph to such a permutation by connecting the i-th socket on the left to the $\sigma(i)$-th socket on the right. Letting σ run over the set of permutations on $[s]$ generates a set of graphs. Endowed with the uniform probability distribution this is the ensemble $\mathcal{C}^n(\lambda, \rho)$. Therefore, if in the future we choose a graph at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$ then the underlying probability distribution is the uniform one.

It remains to associate a code to every element of $\mathcal{C}^n(\lambda, \rho)$. We will do so by associating a parity check matrix to each graph. At first glance it seems natural to define the parity check matrix associated to a given element in $\mathcal{C}^n(\lambda, \rho)$ as the $\{0,1\}$-matrix which has a non-zero entry at row i and column j iff the i-th right node is connected to the j-th left node. Unfortunately the possible presence of multiple edges between pairs of nodes requires a more careful definition. Since the encoding is done over the field GF(2), we define the parity check matrix H as that $\{0,1\}$-matrix which has a non-zero entry at row i and column j iff the i-th right node is connected to the j-th left node an *odd* number of times. As we will see the encoding is accomplished in two steps, a *preprocessing* step, which is an offline calculation performed once only for the given code, and the actual encoding step which is the only data dependent part. For the preprocessing step it is more natural to work with matrices which contain the multiplicities of edges and therefore we define the *extended parity check matrix* $\hat{H}$ as that matrix which has an entry d at row i and column j iff the i-th right node is connected to the j-th left node by d edges. Clearly, H is equal to $\hat{H}$ modulo 2. In the sequel we will also refer to these two matrices as the *adjacency matrix* and the *extended adjacency matrix* of the bipartite graph. Since for every graph there is an associated code, we will use these two terms interchangeably so we will, e.g., refer to codes as elements of $\mathcal{C}^n(\lambda, \rho)$.

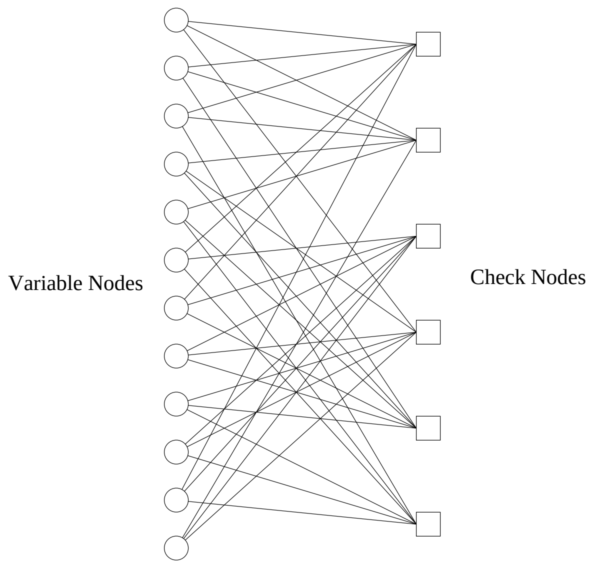

Figure 1: Graphical representation of a $(3, 6)$-regular LDPC code of length 12. The left nodes represent the variable nodes whereas the right nodes represent the check nodes.

Most often LDPC codes are used in conjunction with *message-passing decoders.* Recall that there is a received message associated to each variable node which is the result of passing the corresponding bit of the codeword through the given channel. The decoding algorithm proceeds in *rounds*. At

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.