Belief Propagation as a Dynamical System: The Linear Case and Open Problems

Björn S. Rüffer[a], Christopher M. Kellett[a,1], Peter M. Dower[b], Steven R. Weller[a]

[a] *School of Electrical Engineering & Computer Science, University of Newcastle, Callaghan, NSW 2308, Australia*
[b] *Department of Electrical & Electronic Engineering, University of Melbourne, Parkville, Victoria 3010, Australia*

Abstract

Systems and control theory have found wide application in the analysis and design of numerical algorithms. We present a discrete-time dynamical system interpretation of an algorithm commonly used in information theory called Belief Propagation. Belief Propagation (BP) is one instance of the so-called Sum-Product Algorithm and arises, e.g., in the context of iterative decoding of Low-Density Parity-Check codes. We review a few known results from information theory in the language of dynamical systems and show that the typically very high dimensional, nonlinear dynamical system corresponding to BP has interesting structural properties. For the linear case we completely characterize the behavior of this dynamical system in terms of its asymptotic input-output map. Finally, we state some of the open problems concerning BP in terms of the dynamical system presented.

Key words: large-scale discrete-time systems, iterative decoding, sum-product algorithm
2000 MSC: 93C55 (94B05, 94B35)

1. Introduction

The application of systems and control theory techniques has illuminated many interesting properties of iterative numerical methods. For example, viewing Newton's method as a Lur'e-type system allowed the derivation of novel stability criteria in [1]. Stability and convergence properties of numerical integration routines have been studied in [2] and [1] and dynamical systems that solve optimization problems were considered in [3]. Numerical integration and linear and quadratic programming are considered in [4] where it is shown that many numerical methods can be interpreted as proportional-derivative or proportional-integral controllers for certain dynamical systems. Control theoretic techniques are used to inform algorithm *design* in [5], where simple control-Lyapunov functions lead to improved methods for finding zeros of nonlinear functions. In this paper, we turn our attention to one of the most successful iterative algorithms in digital communications and demonstrate that a control theoretic approach provides important insights into its behavior.

Iterative algorithms are ubiquitous in state-of-the-art communications, especially in decoding of so-called turbo and Low-Density Parity Check (LDPC) codes. The (re-)discovery of these forward error correction codes was a major breakthrough in the 1990s as they approach Shannon's channel capacity within a fraction of a decibel, a milestone unachievable before. In the decoding context for LDPC codes, the so-called Belief Propagation (BP) algorithm and its variants have attracted much attention [6, 7, 8] not only in communications, but also in disciplines such as signal processing and machine learning. Variations of BP have promising applications well beyond communications; e.g., in Kalman filtering and expectation maximization [9, 10, 11, 7].

Although these algorithms usually work well, only limited theoretical insight is available today

*Corresponding author; phone +61 2 4921 6090; fax +61 2 4921 6993
Email addresses: Bjoern.Rueffer@newcastle.edu.au (Björn S. Rüffer), Chris.Kellett@newcastle.edu.au (Christopher M. Kellett), P.Dower@ee.unimelb.edu.au (Peter M. Dower), Steven.Weller@newcastle.edu.au (Steven R. Weller)

to explain why. These algorithms admit fast parallel implementations to efficiently approximate solutions to so-called marginalization problems [6, 12, 7]. In broad terms, marginalization can be — and most commonly is— used to approximate a-posteriori densities, a task familiar from Kalman filtering [13]. In general, however, it is unclear how close these approximations are to true marginals, or indeed whether these algorithms will converge for any given input. In the decoding context as per the setup of Figure 3, the goal of this marginalization problem is to calculate so-called maximum a-posteriori probabilities, based on a-priori probabilities for bits received over a noisy channel. This makes the otherwise computationally very hard *decoding* task feasible (in contrast to the inexpensive *encoding* task). Whilst these iterative algorithms have been reinvented many times and despite the wide area of existing and proposed applications, analysis and design methods still rely mostly on Monte Carlo simulations, instead of exploiting system theoretic results for the underlying highly structured dynamical system.

Although several authors have studied iterative decoding in a dynamical systems context, the underlying high dimensional system equivalent to the particular algorithm has been avoided. In part, this may be due to the fact that its dimension easily approaches the order 10^5 or more. It is known that iterative decoding exhibits nonlinear dynamics. Previous work [14, 15, 16, 17, 18, 19] investigating the dynamics of iterative decoding has involved examining lower dimensional models of the real dynamical system. These lower dimensional models are derived in a stochastic setting and rely on concentration theorem results made possible by assuming, for instance, that the dimension of the underlying dynamical system tends to infinity. While these previous approaches allow one to make statements about the behavior and performance of a "typical" system from a particular class, verifying the design of a specific system still relies on large-scale Monte Carlo simulations. Convergence criteria and uniqueness of fixed points for BP remains an active area of research [20, 21, 22, 23, 24, 25, 26].

This work is a first step away from such stochastic analysis where we study the full deterministic dynamical system defining the BP algorithm. It turns out that this system has interesting symmetry properties. Our contribution is to present and study a discrete-time nonlinear feedback dynamical system formulation equivalent to the so-called Belief Propagation (BP) algorithm. Here equivalence is understood in the sense that there is a one-to-one correspondence between trajectories of the dynamical system and the algorithm. With its long history of studying dynamical systems and improving their behavior towards desired outcomes using control strategies, we believe that the control community has much to offer to the understanding and ultimate improvement of iterative algorithms and BP in particular.

Without assuming any information theoretic background we introduce the dynamical system of interest. It is completely described by the so-called parity-check matrix and is essentially linear, up to one highly structured nonlinearity. We provide some of the existing results in the information theory literature in the language of deterministic dynamical systems. To illustrate the behavior of this system, we consider the special case in which the BP algorithm defines a linear system; i.e., when the feedback nonlinearity becomes linear. For this linear system, we can completely characterize the input-output behavior of BP.

The paper is organized as follows: Section 2 introduces the dynamical system central to this article. Here we also comment on input-output stability and describe a rather surprising result from the information theory literature, couched in the language of dynamical systems, related to the symmetry inherent in the BP algorithm. In information theory, this result is known as the all-zero-codeword assumption. Here it takes the form of a classification of the input-output sets. Then in Section 3 we consider the case when the dynamical system is linear. In an information theory context, this special case corresponds to repeat codes, which are the most easily understood error correction codes. In Section 4 we state some of the open problems regarding BP, translated into the language of control systems. Finally we conclude in Section 5. To make the paper self-contained, in the complementary Appendix we provide more of the coding theory background to aid readers in "decoding" some of the coding literature.

2. The dynamical system of Belief Propagation

2.1. Definitions

Figure 1 shows the basic structure of the BP algorithm, formulated as a deterministic dynamical

system as in (5), see below. A motivation from a decoding perspective is given in Appendix A. All blocks except the S-block are sparse and linear, and the nonlinearity S is highly structured and also sparse. All these blocks are completely described in terms of one sparse matrix H and are entirely deterministic. Let $\mathbb{N}$ denote the set of nonnegative

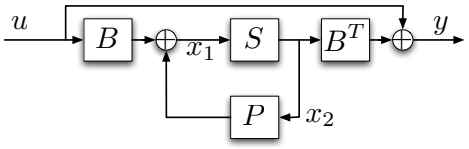

Figure 1: Block diagram of the BP algorithm as a dynamical system. The only nonlinear component is the operator S.

integers. For a positive integer n we denote by $\underline{n}$ the set $\{1, \dots, n\}$. Throughout, $\mathbb{F}_2$ denotes the binary field[1] with elements 0 and 1. Given a binary matrix $H = (h_{ij}) \in \mathbb{F}_2^{m \times n}$ denote by $q := |H|$ the number of nonzero entries in H. Enumerating rowwise from left to right and starting with the top row, let $\mathfrak{n}(k) = \mathfrak{n}_H(k) = (\mathfrak{n}_1(k), \mathfrak{n}_2(k))$ denote the coordinates $(i, j) \in \underline{m} \times \underline{n}$ of the kth non-zero entry in H, for $k \in \underline{q}$. Let $\mathcal{U}$ denote the set of constant functions u from $\mathbb{N}$ to $\mathbb{R}^n$. As usual we identify $u \in \mathcal{U}$ with points $u \in \mathbb{R}^n$.

Remark1. In practice, H is typically sparse, or in the terminology of information theory, has *low density*. H is termed a (low-density) *parity-check matrix*, cf. Appendix A.2. Usually, m is of the order of $R \cdot n$, where R is the so-called design rate of the code described by H. Typical values include $R = \frac{1}{2}$ or $\frac{1}{3}$. H is called regular if every row has the same number of nonzero entries and every column has the same number of nonzero entries. If these numbers, together with R, are kept fixed, q scales as $O(n)$. Typical values for n range between a few hundred to tens of thousands. To give an impression of the dimension of the dynamical system that we are about to define, some reasonable example numbers are $R = \frac{1}{2}$, $n = 20{,}000$, and $m = 10{,}000$, so that q would be of the order of 100,000.

[1] Addition and multiplication are given in the following tables. Sometimes addition is denoted by the symbol $\oplus$ instead of $+$. It coincides with the XOR operation.

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Define the matrix $B = B_H = (b_{ij}) \in \mathbb{R}^{q \times n}$ by

$$b_{ij} = \begin{cases} 1 & \text{if } \mathfrak{n}_2(i) = j \\ 0 & \text{otherwise}\,. \end{cases} \tag{1}$$

Define the maps $S = S_H : \mathbb{R}^q \to \mathbb{R}^q$ and $P = P_H : \mathbb{R}^q \to \mathbb{R}^q$ by

$$P_i(\xi) = \sum_{j \neq i:\, \mathfrak{n}_2(j) = \mathfrak{n}_2(i)} \xi_j, \quad \text{and} \tag{2}$$

$$S_i(\xi) = 2 \operatorname{atanh} \left(\prod_{j \neq i:\, \mathfrak{n}_1(j) = \mathfrak{n}_1(i)} \tanh \frac{\xi_j}{2} \right) \tag{3}$$

for $\xi \in \mathbb{R}^q$ (cf. Appendix A.7 for background on the particular form of (2) and (3)). The operator P can be represented as matrix-vector multiplication. With a slight abuse of notation we define the matrix $P = P_H = (p_{ij}) \in \mathbb{R}^{q \times q}$ to be

$$p_{ij} = \begin{cases} 1 & \text{if } j \neq i,\ \mathfrak{n}_2(i) = \mathfrak{n}_2(j) \\ 0 & \text{otherwise}\,, \end{cases} \tag{4}$$

so that we have $P(\xi) = P\xi$ for all $\xi \in \mathbb{R}^q$. In Section 3 we explicitly construct these matrices for a specific H.

In the following we will consider the time-invariant discrete-time dynamical system given by

$$\begin{aligned} x_1^+ &= P x_2 + B u \\ x_2^+ &= S(x_1) \\ y &= B^T x_2 + u\,, \end{aligned} \tag{5}$$

where $x_1, x_2 \in \mathbb{R}^q$, $u, y \in \mathbb{R}^n$. We write $x = (x_1^T, x_2^T)^T$. The structure of system (5) is depicted in Figure 1.

In this work, we are only interested in input signals that are constant; i.e., $u \in \mathcal{U}$. We do this because we are interested in how the BP algorithm iteratively processes a single input, the predominant case in applications. However, in the last section of the appendix we point out a case where it may be useful to consider non-constant inputs.

The information theoretic interpretation of a zero-initial condition is to start with an internal state of "zero knowledge", which matches with the motivation of the algorithm, which we later describe in the appendix. Also, as this dynamical system represents an algorithm implemented on a computer and not a physical system, we may always choose this initial condition. Hence from now on we assume $x(0) = 0$.

The input is a vector of a-priori log-likelihoods. The output at each iteration is an approximation to the a-posteriori log-likelihood vector. See the brief discussion in the next subsection or the appendix for a more detailed account, especially Appendix A.7.

Associated with H is a bipartite graph, called the *factor graph*, cf. Appendix A.5: Each row of H defines a *factor node*, each column a *variable node*. There is an edge between a factor and a variable node, if the corresponding entry in H is one.

Denote the extended reals by $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. Given the dynamical system (5) we define its *asymptotic input-output map* to be the set valued mapping $\mathcal{G} : \mathcal{U} \to 2^{\overline{\mathbb{R}}^n}$, $u \mapsto \left\{w \in \overline{\mathbb{R}}^n : y(t_k, u) \to w \text{ for some } \{t_k\}_{k\in\mathbb{N}} \text{ as } t_k \to \infty\right\}$.

2.2. Results

The derivation of the sum-product algorithm (SPA) or, respectively, belief propagation (BP), is based on an observation in the context of so-called marginalizations, cf. [6, 7, 27] or the appendix, which we formulate here as a lemma.

Lemma 1. *If the factor graph associated with H is a tree then for each input $u \in \mathcal{U}$, system (5) converges to a limit point in a finite number of steps. In particular, for the asymptotic input-output map we have $\mathcal{G}(u) \in \mathbb{R}^n$ for every $u \in \mathcal{U}$.*

In the tree case the asymptotic output (which is obtained after finitely many steps) yields the *exact* computation of so-called marginals. This motivates why BP (or SPA) is used even in the case that the factor graph contains cycles: One obtains only approximations to those marginals, but the hope is that these approximations are in some sense good.

At this point we should remark that for system (5), $x = 0, u = 0$ gives $y = 0$ for all times. We have the following stability property, which in particular implies a form of input-output stability of the origin.

Proposition 2. *Assume that the factor graph associated to H is a tree. Then there exists a continuous, non-decreasing function $\gamma : \mathbb{R}_+ \to \mathbb{R}_+$, satisfying $\gamma(0) = 0$, such that*

$$\|y(t, u)\| \le \gamma(\|u\|), \quad \forall t \ge 0.$$

Proof. If the factor graph is a tree, by Lemma 1, BP converges in a finite number of steps, denote this number by L. This number of steps L does not depend on the initial conditions or inputs, but only on the girth of the tree. For every finite $t \ge 0$, the mapping $f : u \mapsto y(t, u)$ is continuous and satisfies $f(0) = 0$. Hence the function

$$\gamma(r) := \max_{u:\, \|u\| \le r} \max_{t=0,\dots,L} \|y(t, u)\|$$

is continuous and non-decreasing. □

Concerning the interpretation of this result in a decoding context (cf. Appendix A.7), the inputs and outputs represent vectors of log-likelihood ratios. A log-likelihood ratio $r \in \mathbb{R}$ has the following interpretation: $r = 0$ denotes absolute uncertainty about whether a bit should be a zero or a one. Small non-zero values for r denote a slight but uncertain preference to either zero ($r > 0$) or one ($r < 0$). Large values for r, or even $r = \pm\infty$, denote strong confidence or certainty about the value of the corresponding bit. Hence Proposition 2 says that when the inputs to the algorithm contain little or no knowledge about the vector of bits, then this knowledge cannot be increased too much just by processing the available log-likelihood vector.

The null space of H is a vector space (over $\mathbb{F}_2$) which we denote by $\mathcal{N}_H$. The elements of $\mathcal{N}_H$ are called *codewords*. Define the *hard decision operator* $D : \operatorname{dom} D \to \mathbb{F}_2^n$, by

$$D_i(\xi) = \begin{cases} 0 & \text{if } \xi_i > 0 \\ 1 & \text{if } \xi_i < 0\,, \end{cases} \tag{6}$$

where $\operatorname{dom} D \subset \mathbb{R}^q$ is the set $\{\xi \in \mathbb{R}^q : \xi_i \ne 0\, \forall i\}$.

For each $c \in \mathcal{N}_H$ we let $Y_c \subset \mathbb{R}^q$ denote the open set given by $Y_c := \{\xi \in \operatorname{dom} D : D(\xi) = c\}$. The set Y_c is referred to as a *codeword set*. For fixed $c \in \mathcal{N}_H$ we are interested in the set of inputs $u \in \mathcal{U}$ so that the trajectory $y(t, u)$ eventually reaches Y_c. Denote these regions by

$$R_c = R_{c,H} = \{u \in \mathcal{U} : \exists t \ge 0 : y(t, u) \in Y_c\}\,. \tag{7}$$

Lemma 3. *For each $c \in \mathcal{N}_H$, the codeword set Y_c is invariant in the sense that $Y_c \subset R_c$.*

Proof. This is due to the output equation and the assumption, that our dynamical system always starts at $x(0) = 0$. □

The following result is a reformulation of what is commonly known in communications as the *all-zero codeword assumption*.

Theorem 4. *For all $c \in \mathcal{N}_H$ there exists a diagonal matrix $T = T(c) \in \mathbb{R}^{q\times q}$ such that $R_c = TR_0$,*

where the diagonal elements of T are given by

$$t_{ii} = \begin{cases} -1 & \text{if } c_{\mathfrak{n}_2(i)} = 1 \\ +1 & \text{otherwise}. \end{cases}$$

In other words, despite the fact that the BP algorithm is a *nonlinear* system, the regions $\mathcal{R}_c$ are the same size and shape for all $c \in \mathcal{N}_H$. Hence, since $0 \in \mathcal{N}_H$, coding theorists typically restrict their attention to the *all-zero codeword* when analyzing a particular H. A proof of this result can be found in [27, p.215, Lemma 4.90].

3. Linear case

Consider the case as per the $(n-1) \times n$ matrix in (8), where each row of H contains exactly two non-zero entries. Then the product term in (3) simplifies to $\tanh \frac{\xi_j}{2}$, where $j \neq i$. Thence, applying (3), $S_i(\xi) = \xi_j$, for some $j \neq i$. That is, S is a permutation operator. Note that (8) is a canonical form for matrices with two non-zero entries per row, provided that the factor graph is connected.

$$H = \begin{pmatrix} 1 & 1 & 0 & & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & \dots & 0 & 1 & 1 & 0 \\ 0 & \dots & & 0 & 1 & 1 \end{pmatrix}. \tag{8}$$

The error correction code defined by the matrix H is the simplest error correction code available. Imagine that we wish to transmit a binary value, 1 or 0, over a noisy communication channel. In an effort to make sure the value is received reliably, instead of transmitting the value once, we may, for instance, transmit the value three times. On the receiving end, we can then take a majority vote and, hopefully, be more confident that we correctly received what was originally intended. Presumably, the more times we repeat the transmission for a single bit message, the better the receiver's chances of correctly guessing what was intended by the transmitter. This scheme is referred to as a *repeat-n code*, where n is the number of transmitted bits. (More detail on the communication process is contained in the Appendix.)

Now since the map $x \mapsto 2\,\mathrm{atanh}(\tanh(x/2))$ is the identity, the operator S becomes linear and, again with slight abuse of notation, can be represented by matrix-vector multiplication $S(\xi) = S\xi$ with the matrix $S = S_H = (s_{ij}) \in \mathbb{R}^{q \times q}$ given by

$$s_{ij} = \begin{cases} 1 & \text{if } \mathfrak{n}_1(i) = \mathfrak{n}_1(j) \\ 0 & \text{else}. \end{cases} \tag{9}$$

System (5) reduces to the linear system

$$\begin{aligned} x^+ &= \begin{bmatrix} 0 & P \\ S & 0 \end{bmatrix} x + \begin{bmatrix} B \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} 0 & B^T \end{bmatrix} x + u\,. \end{aligned} \tag{10}$$

Since the factor graph for any repeat code represented by a parity-check matrix of the form (8) is a tree , we have according to Lemma 1, for each $u \in \mathcal{U}$ an equilibrium point $x^* = x^*(u)$ satisfying

$$x^* = \begin{bmatrix} 0 & P \\ S & 0 \end{bmatrix} x^* + \begin{bmatrix} B \\ 0 \end{bmatrix} u\,. \tag{11}$$

Lemma 5. *The matrices* $\begin{bmatrix} 0 & P \\ S & 0 \end{bmatrix}$ *and* PS *are nilpotent, and for* $k \geq 0$ *the* (i,j)*th entry of* $(PS)^k$ *is given by*

$$((PS)^k)_{ij} = \begin{cases} 1 & \text{if } i \text{ is even and } j = i + 2k \\ & \text{or if } i \text{ is odd and } j = i - 2k \\ 0 & \text{otherwise}. \end{cases} \tag{12}$$

Proof. Induction over k for equation (12); then nilpotency follows. □

Now equation (11) can be rewritten to

$$x^* = \begin{bmatrix} I & -P \\ -S & I \end{bmatrix}^{-1} \begin{bmatrix} B \\ 0 \end{bmatrix} u = \sum_{k=0}^{\infty} \begin{bmatrix} 0 & P \\ S & 0 \end{bmatrix}^k \begin{bmatrix} B \\ 0 \end{bmatrix} u\,. \tag{13}$$

Using basic linear algebra we obtain

$$y^* = \begin{bmatrix} 0 & B^T \end{bmatrix} \left(\sum_{k=0}^{\infty} \begin{bmatrix} 0 & P \\ S & 0 \end{bmatrix}^k \right) \begin{bmatrix} B \\ 0 \end{bmatrix} u + u \tag{14}$$

$$= B^T \Big(S \sum_{k \geq 0} (PS)^k \Big) Bu + u\,. \tag{15}$$

From Lemma 5 it follows that the infinite sums in (13)–(15) consist of only finitely many nonzero terms.

Example 1. *Let*

$$H = \begin{bmatrix} 1^1 & 1^2 & 0 & 0 \\ 0 & 1^3 & 1^4 & 0 \\ 0 & 0 & 1^5 & 1^6 \end{bmatrix}, \tag{16}$$

which corresponds to a repeat-4 code. The superscripts indicate the enumeration $\mathfrak{n}$ *defined in Section 2. For example,* $(\mathfrak{n}_1(1), \mathfrak{n}_2(1)) = (1, 1)$, $(\mathfrak{n}_1(2), \mathfrak{n}_2(2)) = (1, 2)$, $(\mathfrak{n}_1(3), \mathfrak{n}_2(3)) = (2, 2)$, *and so on. For the repeat-4 code given above, we have* $B, S,$ *and* P *given by, respectively,*

$$\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix}, \quad \begin{bmatrix} 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{bmatrix}$$

$$\textit{and} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then the asymptotic input-output map is given by the matrix

$$B^T\Big(S\sum_{k\geq 0}(PS)^k\Big)B + I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

An example trajectory of the dynamical system corresponding to the parity-check matrix (16) *is given in Figure 2, where* $u = (1, 2, 3, -4)^T$.

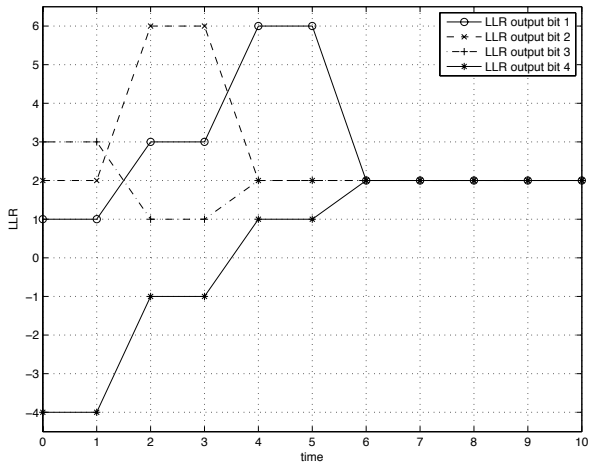

Figure 2: Dead-beat convergence of the per bit (output) log-likelihood ratios (LLRs) $y_i(t)$ $(t = 0, \ldots, 10)$ to the sum of the input LLRs u_i for Example 1.

The example provides the asymptotic input-output behavior of an iterative BP decoder for a repeat-4 code. The interpretation of this result might seem odd at first, since instead of averaging the inputs and assigning this average to every output, the sum of all inputs is assigned to every output. The explanation lies in the representation of the input per-bit probabilities: u_i represents a log-likelihood ratio, i.e., a number of the form $u_i = \log \frac{p_0^{(i)}}{p_1^{(i)}}$, where $p_0^{(i)} + p_1^{(i)} = 1$ (cf. Appendix A.7). If we now compute the probability that the first bit *and* the second bit *and* the third bit and so on are, say the zero bit, we compute a product, $r_0 = p_0^{(1)} p_0^{(2)} p_0^{(3)} \cdots$, and similarly, $r_1 = p_1^{(1)} p_1^{(2)} p_1^{(3)} \cdots$. Now $r = (r_0, r_1)$ in general will not satisfy $r_0 + r_1 = 1$, i.e., not represent a probability distribution, so that we would have to normalize to $\tilde{r} = \frac{1}{r_0 + r_1} \cdot r$. The log-likelihood ratio $l = \frac{r_0}{r_1}$ is not altered by this scaling, and it is this log-likelihood ratio that the asymptotic input-output map assigns to every output.

Of course, this result generalizes easily to repeat-n codes.

Theorem 6. *For a repeat-n code represented by a parity-check matrix* H *in canonical form* (8)*, the asymptotic input-output-map* $\mathcal{G}$ *is given by* $\mathcal{G}(u) = Eu$*, where* E *denotes the square matrix whose entries are all 1.*

Proof. From the explicit formula (12) for $(PS)^k$ in Lemma 5 we see that the $(i, j)th$ entry of the matrix $\sum_{k\geq 0}(PS)^k$ is a one if i is even and $j \in i + 2\mathbb{N}$, or i is odd and $j \in i - 2\mathbb{N}$, and zero otherwise. Multiplication by S from the left swaps every two consecutive rows, so that we obtain $\Big(S\sum_{k\geq 0}(PS)^k\Big)_{ij} = 1$ if i is odd and $j \in i+1+2\mathbb{N}$, or if i is even and $j \in i - 1 - 2\mathbb{N}$, and 0 otherwise. Multiplication by B^T from the left and B from the right and addition of the identity then immediately gives E, the all-one matrix. □

4. Open Problems

In this section we state a few problems that, to our knowledge, are still unanswered in information theory, but of great interest for applications and are indicative of the types of possible contributions for control theory in this area.

Problem P1 — Quantification of Decoding Performance

One measure of the quality of a decoder is related to how many (incorrectly) received transmis-

sions can be decoded to an actual codeword. For an iterative decoder this can be phrased as: What set of inputs u does the iterative decoder ultimately map to a codeword? In light of Theorem 4, one can reasonably restrict this question to: What set of inputs u does the iterative decoder ultimately map to the *all-zero codeword*?

As an alternative to the above general problem, which essentially asks for a computation of a controllable set, it would be of interest to know what fraction of the input space the iterative decoder maps to a valid codeword (i.e., an element of $\mathcal{N}_H$). Since this fraction is clearly overbounded by the inverse of the number of codewords, or $1/|\mathcal{N}_H|$, (based on Theorem 4 and completely partitioning the input space) one might consider the fitness measure

$$f_H := \lim_{r\to\infty} \frac{|\mathcal{N}_H|\lambda(R_0 \cap B_r(0))}{\lambda(B_r(0))}, \qquad (17)$$

as a measure of how close BP performs to maximum likelihood decoding, where $B_r(0)$ denotes a ball of radius r around the origin and λ the Lebesgue measure. Clearly, $f_H \in [0, 1]$ and ideally f_H approaches 1.

Problem P2 — Optimizing Parity-Check Matrices

A given code can be represented by different parity-check matrices. So it is natural to ask if a given H can be modified to enlarge the region R_0 or the fitness measure f_H. A related question involves quantifying the effects of cycles in the parity-check matrix, doubling factor nodes, adding linear combinations of existing rows, etc.

Problem P3 — Control

Naturally, the question arises as to whether or not it is possible to improve the performance of the iterative decoder via the judicious use of control. For example, is it possible (and computationally feasible) to change the feedback loop in the dynamical system (5) in order to force faster convergence? Furthermore, can we force faster convergence without shrinking the sets R_c? On the other hand, might it be possible to trade off speed of convergence for enlarging the measure (17)?

Problem P4 — Performance Measures in Information Theory

Most tools used to analyze performance of LDPC codes rely on large-scale Monte-Carlo simulations; e.g., bit-error-rate plots, so-called density evolution, and ExIT charts all require the processing of a very large set of random inputs u. Quantifying the fitness measure or similar quantities in terms of the so-called channel parameter, e.g, by replacing the Lebesgue measure in (17) by a different measure. This may provide a way to short-circuit lengthy Monte-Carlo runs.

5. Conclusions

The sum-product algorithm is in widespread use in signal processing applications from Kalman filtering to expectation maximization. In particular, under the moniker of *belief propagation* (BP) it has revolutionized error correction coding. Error correction codes have been demonstrated to perform within a fraction of a decibel of the fundamental limit predicted by Shannon's noisy coding theorem. However, these codes require hundreds of hours for both design and validation, particularly as design is currently more art than science, due in part to a paucity of results on the iterative processing of BP. Furthermore, the demonstrated high-performance codes typically are so long (i.e., contain so many bits per codeword) that they are completely impractical for most applications.

This work represents a first step in understanding the BP algorithm as a dynamical system with the aim of developing principled design tools. Toward this end, we have shown how to cast the BP algorithm as a dynamical system. Furthermore, we have considered a special case where the BP algorithm reduces to a linear system. In this special case, we are then able to completely characterize the convergence behavior of BP.

We identified some of the open problems in information theory and cast them in control theoretic terms. Control theory seems to be the natural setting to consider these type of problems and with its long history of considering iterative discrete-time control systems, it is likely to make useful contributions to the information theory field.

6. Acknowledgments

B.S. Rüffer, C.M. Kellett, and S.R. Weller have received support from the Australian Research Council (ARC) under grant DP0771131. P.M. Dower has received support from the ARC under grant DP0880494.

References

[1] K. Kashima, Y. Yamamoto, System theory for numerical analysis, Automatica 43 (7) (2007) 1156–1164.

[2] A. M. Stuart, A. R. Humphries, Dynamical systems and numerical analysis, Vol. 2 of Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, 1996.

[3] R. W. Brockett, Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems, Linear Algebra and its Applications 146 (1991) 79–91.

[4] A. Bhaya, E. Kaszkurewicz, Control perspectives on numerical algorithms and matrix problems, Vol. 10 of Advances in Design and Control, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.

[5] A. Bhaya, E. Kaszkurewicz, A Control-Theoretic Approach to the Design of Zero Finding Numerical Methods, IEEE Transactions on Automatic Control 52 (6) (2007) 1014–1026. doi:10.1109/TAC.2007.899109.

[6] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Transactions on Information Theory 47 (2) (2001) 498–519.

[7] H.-A. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, F. Kschischang, The factor graph approach to model-based signal processing, Proceedings of the IEEE 95 (6) (2007) 1295–1322.

[8] J. Yedidia, W. Freeman, Y. Weiss, Constructing free-energy approximations and generalized belief propagation algorithms, IEEE Transactions on Information Theory 51 (7) (2005) 2282–2312.

[9] J. Dauwels, S. Korl, H.-A. Loeliger, Expectation maximization as message passing, in: Proc. of the IEEE International Symposium on Information Theory ISIT, 2005, pp. 583–586. doi:10.1109/ISIT.2005.1523402.

[10] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. Roy. Statist. Soc. Ser. B 39 (1) (1977) 1–38.

[11] A. Eckford, The factor graph EM algorithm: applications for LDPC codes, in: 6th IEEE Workshop on Signal Processing Advances in Wireless Communications, 2005, pp. 910–914. doi:10.1109/SPAWC.2005.1506272.

[12] H.-A. Loeliger, An introduction to factor graphs, IEEE Signal Processing Magazine 21 (1) (2004) 28–41.

[13] R. E. Kalman, A new approach to linear filtering and prediction problems, Transactions of the ASME–Journal of Basic Engineering 82 (Series D) (1960) 35–45.

[14] D. Agrawal, A. Vardy, The turbo decoding algorithm and its phase trajectories, IEEE Transactions on Information Theory 47 (2) (2001) 699–722.

[15] M. Fu, Stochastic analysis of turbo decoding, IEEE Transactions on Information Theory 51 (1) (2005) 81–100.

[16] C. M. Kellett, S. R. Weller, Bifurcations and EXIT charts for the binary erasure channel, in: Proceedings of IEEE International Symposium on Information Theory, Seattle, WA, USA, 2006, pp. 2559–2563.

[17] C. M. Kellett, S. R. Weller, Bifurcations in iterative decoding and root locus plots, IET Control Theory and its Applications.

[18] L. Kocarev, F. Lehmann, G. M. Maggio, B. Scanavino, Z. Tasev, A. Vardy, Nonlinear dynamics of iterative decoding systems: Analysis and applications, IEEE Transactions on Information Theory 52 (4) (2006) 1366–1384.

[19] X. Zheng, F. C. M. Lau, C. K. Tse, S. C. Wong, Study of bifurcation behavior of LDPC decoders, Internat. J. Bifur. Chaos Appl. Sci. Engrg. 16 (11) (2006) 3435–3449.

[20] T. Heskes, On the uniqueness of loopy belief propagation fixed points, Neural Computation 16 (11) (2004) 2379–2413.

[21] A. Ihler, Accuracy bounds for belief propagation, in: Proceedings of the 23th Conference on Uncertainty in Artificial Intelligence UAI 2007, Vancouver, BC Canada, 2007.

[22] A. T. Ihler, J. W. Fisher, A. S. Willsky, Loopy belief propagation: Convergence and effects of message errors, Journal of Machine Learning Research 6 (2005) 905–936.

[23] J. M. Mooij, H. J. Kappen, Sufficient conditions for convergence of the sum-product algorithm, IEEE Transactions on Information Theory 53 (12) (2007) 4422–4437.

[24] N. Taga, S. Mase, Applications of Gibbs measure theory to loopy belief propagation algorithm, in: A. Gelbukh, C. A. Reyes-Garcia (Eds.), Proceedings of the 5th Mexican International Conference on Artificial Intelligence MICAI 2006: Advances in Artificial Intelligence, Vol. 4293, Apizaco, Mexico, 2006, pp. 197–207.

[25] S. C. Tatikonda, M. I. Jordan, Loopy belief propagation and Gibbs measures, in: Uncertainty in Artificial Intelligence, Morgan Kaufmann, 2002, pp. 493–500.

[26] Y. Weiss, W. Freeman, On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs, IEEE Transactions on Information Theory 47 (2) (2001) 736–744. doi:10.1109/18.910585.

[27] T. Richardson, R. Urbanke, Modern Coding Theory, Cambridge University Press, New York, 2008.

[28] Signal Processing Microelectronics, http://sigpromu.org/systemanalysis/.

[29] International ISBN Agency, ISBN Users' Manual International edition, available online at http://www.isbn-international.org/ (2005).

A. Background on iterative decoding

This appendix provides a gentle introduction to the elements of coding theory necessary to further investigate iterative decoding in the coding literature.

A deeper treatise of the material in this appendix can be found in the book [27]. Some example Matlab code implementing material of this appendix as well as of Sections 2 and 3 is available online at [28]. Regarding the notation there is to say that both the control and the information theory communities have their canonical uses for the letters X and Y, in both lower and upper case. To avoid introducing unnecessary non-standard notation, we switch officially at this point to the information theory meaning of these letters.

We start with an illustrative example of *channel coding*, i.e., how one might deliberately introduce

redundancy to be able to tell if a transmitted *message* has been corrupted during during transmission over a *channel*. Ultimately one would also like to be able to *correct* errors that have occurred during the transmission of a message.

Example 2. *The redundancy might be chosen such that certain constraints are satisfied, e.g., ISBN-13, the international standard book number [29] consists of thirteen decimal digits* $d_1 d_2 \dots d_{13}$*, where the last digit* d_{13} *is computed as*

$$d_{13} \equiv -\sum_{i=1}^{12} w_i d_i \ mod\ 10\,. \tag{18}$$

Here the weights w_i *are given by*

$$w_i = \begin{cases} 1 & \text{if } i \text{ is odd} \\ 3 & \text{if } i \text{ is even}\,. \end{cases}$$

We could equivalently write (18) *as a* parity-check equation *for all digits, i.e.,*

$$\sum_{i=1}^{13} w_i d_i \equiv 0 \ mod\ 10\,. \tag{19}$$

We observe that if two 13 digit numbers, $d = d_1 \dots d_{13}$ *and* $c = c_1 \dots c_{13}$*, satisfy parity-check equation* (19)*, then also their sum and multiples (that is, element-wise, modulo 10) will satisfy this equation, due to its linear nature. Given a 13 digit number, based on* (19) *we can tell whether or not this number is a valid ISBN, so within certain limitations we are able to tell that the transmission of the number over a channel (scanning a bar code in this case, e.g.) was successful or not. The set of solutions (i.e., the nullspace) of* (19) *is called a* code*, and individual solutions are* codewords*. These are the very basics of linear parity-check codes.*

Here is the general channel coding problem: To transmit a message m, say a binary vector of length k, the message m is augmented with a redundant vector $r(m)$, and the vector $x = (m^T, r(m)^T)^T$ is transmitted. Now the channel will corrupt the transmitted message x. The channel could be, e.g., transmission via radio frequency, or the result of scanning a bar-code. The receiver then can use the augmented (redundant) information to attempt to recover those parts of the message that have been lost or corrupted during transmission. The block diagram in Figure 3 shows the usual digital communications setup. We will consecutively consider each of the blocks in Fig. 3 in the sequel. The connection to the dynamical system formulation (5) is made in Section A.7.

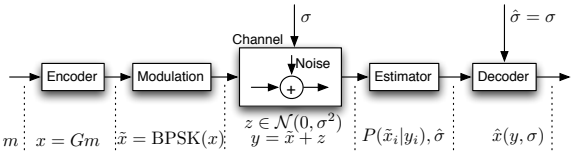

Figure 3: Channel coding essentials.

A.1. Notation

To ease notation, for functions $f : \mathbb{F}_2^n \to \mathbb{R}, x \mapsto f(x)$ we introduce the so-called *not-sum* or *summary* operation, cf. [6],

$$\sum_{\sim x_i} f(x) := \sum_{x_1,\dots,x_{i-1},x_{i+1},\dots,x_n} f(x)\,, \tag{20}$$

where $x = (x_1, \dots, x_n)^T$. For a given set A the indicator function 1_A is defined by

$$1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise}\,. \end{cases}$$

A.2. Linear block codes and parity-check matrices

A binary linear parity-check code is defined through a weighted sum; i.e., solutions $x = (x_1, \dots, x_n)^T$ of

$$\sum_{i=1}^{n} w_i x_i = 0\,, \tag{21}$$

with $w_i, x_i \in \mathbb{F}_2$, the binary field. Again, linearity can easily be seen. In fact, the code defined by (21) is an $(n-1)$-dimensional vector subspace of $\mathbb{F}_2^n$, provided at least one w_i is nonzero.

A linear *block code* C of length n and rate $R = k/n$, $k < n$, is a vector subspace of dimension k in $\mathbb{F}_2^n$. The dimension k determines that we can choose k bits arbitrarily and the remaining $n-k$ bits in each codeword will be determined by the structure of the code; i.e, we can consider $\mathbb{F}_2^k$ as the vector space of possible messages that we can send, before any parity-check information is appended. Clearly, there are exactly 2^k distinct possible messages.

The code C can be described as the nullspace of a *parity-check matrix* $H \in \mathbb{F}_2^{(n-k)\times n}$, so that $C = \{c \in \mathbb{F}_2^n : Hc = 0\}$, or by a *generator-matrix* $G \in$

$\mathbb{F}_2^{n\times k}$, that maps message vectors to codewords; i.e., $C = \{Gm : m \in F_2^k\}$.

In general, for every parity-check matrix there is no unique choice of generator matrix and *vice versa*. However it is always possible to calculate a parity-check matrix from a generator matrix and *vice versa* [27, p.36].

A binary *low-density parity-check* (LDPC) code is a binary linear block code C represented by a sparse parity-check matrix H. This sparsity is crucial to the decoding described in Section A.6: The factor graph corresponding to a sparse matrix H has only few edges, hence the message passing along those edges in Section A.6 will involve fewer messages.

A.3. BPSK and a-priori likelihoods

Now that we have a code and codewords, we still have to transmit binary vectors over a noisy channel. Take, e.g., the additive white Gaussian noise (AWGN) channel, a popular choice in information theory.

First binary bits are mapped to channel symbols. For the AWGN channel the channel symbols are real numbers, representing, e.g., the amplitude of a sent or received analogue signal at certain time instances. Here we can use so-called binary phase shift keying (BPSK), which maps $x \in \mathbb{F}_2^n$ to $\tilde{x} \in \mathbb{R}^n$ per

$$x_i \mapsto \tilde{x}_i = (-1)^{x_i} . \tag{22}$$

The channel then adds some noise realization, i.e., a sample drawn from a $\mathcal{N}(0, \sigma^2)$ distribution to each channel symbol, before the receiver sees it. So for each channel input $\tilde{x}_i \in \mathbb{R}$ the channel output is $y_i = \tilde{x}_i + z$, where $z \in \mathcal{N}(0, \sigma^2)$. Here σ^2 is a channel parameter, namely the variance of the channel noise, and is assumed to be known at the receiver.

If σ^2 is known on the receiver side, it is possible to compute $p_{Y_i|X_i}(y_i, x_i)$, the probability density corresponding to the cumulative conditional probability distribution $(x_i, y_i) \mapsto \frac{P(Y_i \le y_i, X_i = x_i)}{P(X_i = x_i)}$. Note that we have switched to a probabilistic setting in order to reflect that the receiver side doesn't know which codeword has been sent. The receiver only knows that some realization of a random variable X has been sent, where X takes values in the set of codewords. The ratio

$$r_i := \frac{p_{Y_i|X_i}(y_i, 0)}{p_{Y_i|X_i}(y_i, 1)} \tag{23}$$

is called the a-priori likelihood ratio. Its computation requires knowledge of σ^2. Under the assumption that $p_{X_i}(1) = p_{X_i}(0)$ the right hand side of (23) can be rewritten as

$$r_i = \frac{p_{X_i|Y_i}(0, y_i)}{p_{X_i|Y_i}(1, y_i)} . \tag{24}$$

A simple decoding approach assigns the binary-valued guess, $\hat{x}_i$, to 0 or 1 according to the likelihood (either $r_i > 1$ or $r_i < 1$). By verifying whether $H\hat{x} = 0$ or not, the receiver can see if the received vector is a valid codeword. If the computed estimate $\hat{x}$ is not a codeword, then some more involved decoding has to be performed, to hopefully recover the transmitted codeword.

A.4. Maximum a-posteriori probability (MAP) decoding

The idea of the MAP decoding approach is to compute a-posteriori probabilities/likelihoods for every bit and then choose that value for the bit which maximizes this probability.

Let us consider a simple example: A single parity check node. For the parity-check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{25}$$

the first check-equation imposed is

$$x_1 + x_2 + x_4 = 0 , \quad x_i \in \mathbb{F}_2 , \tag{26}$$

so that a valid codeword x has to satisfy equation (26).

We make the following assumptions:

A1 the channel is memoryless:
$$p_{Y|X}(y, x) = \prod p_{Y_i|X_i}(y_i, x_i)$$

A1 for all $y \in \mathbb{R}^n$ and $x_i \in \mathbb{F}_2$ we are given $p_{Y_i|X_i}(y_i, x_i)$

A1 uniform priors: $p_X(x^1) = p_X(x^2)$ for $x^1, x^2 \in C$, $p_X(x) = 0$ for $x \notin C$

Bit-wise *maximum a-posteriori* (MAP) decoding chooses for each bit the following best estimate (cf. [27, p.57]):

$$\hat{x}_i = \arg\max_{x_i \in \mathbb{F}_2} p_{X_i|Y}(x_i, y) . \tag{27}$$

By the law of total probability this is

$$= \arg\max_{x_i \in \mathbb{F}_2} \sum_{\sim x_i} p_{X|Y}(x, y) ,$$

where the sum is taken over all x_j, $j \neq i$, cf. (20). Application of Bayes' rule yields

$$= \arg\max_{x_i \in \mathbb{F}_2} \sum_{\sim x_i} p_{Y|X}(y,x) \frac{p_X(x)}{p_Y(y)}$$

and using assumption A3 and the fact that $p_Y(y)$ is a constant in this formula, we have

$$\begin{aligned} &= \arg\max_{x_i \in \mathbb{F}_2} \sum_{\sim x_i} p_{Y|X}(y,x) 1_C(x) \\ &= \arg\max_{x_i \in \mathbb{F}_2} \sum_{\sim x_i} \prod_j p_{Y_j|X_j}(y_j,x_j) 1_C(x)\,, \end{aligned} \tag{28}$$

where in the last step we have used assumption A1. The last expression (28) is efficiently computable using the factor graph message-passing approach introduced in the following sections.

A.5. Factor graphs

The statement $x \in C$, or that x is a codeword, can be rewritten as

$$\prod_{i=1}^{m} 1_{\mathcal{N}(h_i)}(x) = 1, \tag{29}$$

where h_i denotes the ith row of H and $\mathcal{N}(h_i)$ is the nullspace defined by this row vector. Clearly a codeword must be in the intersection of the nullspaces corresponding to the rows of the parity-check matrix.

In view of (29) it becomes clear why the graph defined in Section 2 is called *factor* graph: It describes the correspondence between the factors and arguments in (29). A factor node is connected to a variable node if and only if the value of that factor depends on the value of that variable.

An example factor graph corresponding to the parity check matrix (25) is given in Figure 4. We

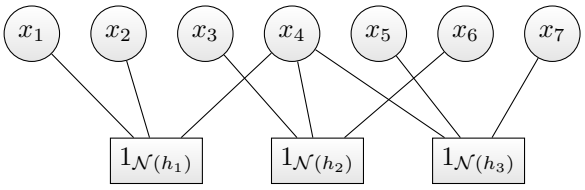

Figure 4: A factor graph.

will in the sequel interchangeably use the symbol of the variable to denote the corresponding variable node, as well as write h_j or $1_{\mathcal{N}(h_j)}$ for the corresponding factor node.

A.6. Message-passing

When we try to explicitly compute the *marginalizations*

$$S_i(x_i) := \sum_{\sim x_i} \prod p_{Y_j|X_j}(y_j,x_j) 1_C(x)\,, \tag{30}$$

in (28) for all $i = 1, \ldots, n$, by employing the distributive law, it turns out that certain intermediate terms keep reappearing, which are of the same shape as (30), but the sums are taken over fewer terms and products contain fewer factors. An efficient implementation would have to try to benefit from this observation.

Example 3. *Taking the parity-check matrix* (25) *as an example, we might compute*

$$\begin{aligned} S_1(x_1) &= \sum_{\sim x_1} \prod_j p_{Y_j|X_j}(y_j,x_j) 1_C(x) \\ &= \sum_{\sim x_1} \prod_j p_{Y_j|X_j}(y_j,x_j) \prod_k 1_{\mathcal{N}(h_k)}(x) \\ &= p_1(x_1) \sum_{x_2} p_2(x_2) \Bigg[\sum_{x_4} p_4(x_4) 1_{\mathcal{N}(h_1)}(\underline{x}_1) \\ &\quad \cdot \Bigg(\sum_{x_3,x_6} p_3(x_3) p_6(x_6) 1_{\mathcal{N}(h_2)}(\underline{x}_2) \Bigg) \\ &\quad \cdot \Bigg(\sum_{x_5,x_7} p_5(x_5) p_7(x_7) 1_{\mathcal{N}(h_3)}(\underline{x}_3) \Bigg) \Bigg] \end{aligned}$$

where we abbreviated $p_i(x_i) := p_{X_i|Y_i}(x_i,y_i)$, $\underline{x}_1 = (x_1,x_2,0,x_4,0,0,0)$, $\underline{x}_2 = (0,0,x_3,x_4,0,x_6,0)$, *and* $\underline{x}_3 = (0,0,0,x_4,x_5,0,x_7)$. *Observe that* $1_C(x) = \prod_j 1_{\mathcal{N}(h_j)}(\underline{x}_j)$. *Now, the expression for* $S_2(x_2)$ *is almost the same, namely*

$$\begin{aligned} S_2(x_2) &= p_2(x_2) \sum_{x_1} p_1(x_1) \Bigg[\sum_{x_4} p_4(x_4) 1_{\mathcal{N}(h_1)}(\underline{x}_1) \\ &\quad \cdot \Bigg(\sum_{x_3,x_6} p_3(x_3) p_6(x_6) 1_{\mathcal{N}(h_2)}(\underline{x}_2) \Bigg) \\ &\quad \cdot \Bigg(\sum_{x_5,x_7} p_5(x_5) p_7(x_7) 1_{\mathcal{N}(h_3)}(\underline{x}_3) \Bigg) \Bigg]. \end{aligned}$$

Clearly the sums over x_5, x_7 *and* x_3, x_6 *would only have to be computed once. Especially for very high dimensional but sparse matrices* H*, this observation leads to an efficient implementation to compute (or in general approximate) the functions* S_i.

It turns out that if the factor graph defined by H is a tree, then the functions S_i can be computed iteratively, according to the following message-passing rules.

Initially, each variable node x_i sends the message (that is, a function $\mathbb{F}_2 \to \mathbb{R}$) $p_i(\cdot)$ to all adjacent factor nodes. From now on we write $\mu_{x_i \to h_j}$ for the messages sent from variable node x_i to factor node h_j and $\mu_{h_j \to x_i}$ for the message in the opposite direction.

Then the following two steps are iterated. First update the factor-to-variable messages according to

$$\mu_{h_j \to x_i}(x_i) = \sum_{\substack{x_k \colon k \neq i \\ x_k \text{ adjacent to } h_j}} 1_{\mathcal{N}(h_j)}(\underline{x}_j) \prod_{\substack{l \neq i \colon \\ x_l \text{ adjacent to } h_j}} \mu_{x_l \to h_j}(x_l)\,. \tag{31}$$

Then update the variable-to-factor messages as

$$\mu_{x_i \to h_j}(x_i) = p_i(x_i) \cdot \prod_{\substack{h_l \text{ adjacent to } x_i \\ l \neq j}} \mu_{h_l \to x_i}(x_i)\,. \tag{32}$$

The intermediate marginalization after every iteration is given by

$$\tilde{S}_i(z) = p_i(z) \cdot \prod_{h_l \text{ adjacent to } x_i} \mu_{h_j \to x_i}(z)\,. \tag{33}$$

Example 4. *In Example 3, h_2 would send*

$$\mu_{h_2 \to x_4}(x_4) = \sum_{x_3, x_6} p_3(x_3) p_6(x_6) 1_{\mathcal{N}(h_2)}(\underline{x}_2)$$

to x_4, after receiving the initial messages

$$\mu_{x_3 \to h_2}(z) = p_3(z) \text{ and } \mu_{x_6 \to h_2}(z) = p_6(z).$$

Then x_4 sends

$$\begin{aligned}\mu_{x_4 \to h_1}(x_4) &= p_4(x_4) \cdot \mu_{h_2 \to x_4}(x_4) \cdot \mu_{h_3 \to x_4}(x_4) \\ &= p_4(x_4) \cdot \sum_{x_3, x_6} p_3(x_3) p_6(x_6) 1_{\mathcal{N}(h_2)}(\underline{x}_2) \\ &\quad \cdot \sum_{x_5, x_7} p_5(x_5) p_7(x_7) 1_{\mathcal{N}(h_3)}(\underline{x}_3)\end{aligned}$$

to h_1. In the next iteration h_1 in turn sends

$$\begin{aligned}\mu_{h_1 \to x_1}(x_1) = \sum_{x_2} p_2(x_2) \Bigg[&\sum_{x_4} p_4(x_4) 1_{\mathcal{N}(h_1)}(\underline{x}_1) \\ &\cdot \left(\sum_{x_3, x_6} p_3(x_3) p_6(x_6) 1_{\mathcal{N}(h_2)}(\underline{x}_2) \right) \\ &\cdot \left(\sum_{x_5, x_7} p_5(x_5) p_7(x_7) 1_{\mathcal{N}(h_3)}(\underline{x}_3) \right) \Bigg]\end{aligned}$$

to x_1. At this stage we obtain S_1 as

$$S_1(x_1) = \tilde{S}_1(x_1) = p_1(x_1) \cdot \mu_{h_1 \to x_1}(x_1)\,.$$

Observe that from this point on the message-updating rules do not alter the messages any more. We might say that the algorithm has converged.

In general the factor graph is not a tree, and this iterative procedure might not converge in the sense that messages become constant after some iteration. Nevertheless this algorithm is used very successfully for decoding even in the presence of cycles in the graph. A common stopping criterion is that the intermediate marginalizations give a codeword, which can be efficiently tested using the parity-check matrix.

The update rules (31),(32) also appear in other, more general, marginalization problems and gave rise to the name *sum-product algorithm*, due to the sum over products in (31). In the case of binary probability distributions the sum-product algorithm is also commonly referred to as belief propagation.

A.7. LLR reformulation yielding atanh-formula

Finally we show how one obtains the formulas (2),(3) from the update rules (31),(32). The basic idea is to represent each message μ, which —up to scaling— is a probability distribution, by a single number. This approach reduces the amount of storage needed for each message, and, if one chooses the representation carefully, also simplifies the message-update rules. Each message can be though of as a 2-vector, $\mu = (\mu(0), \mu(1))$, so we might define the *log-likelihood ratio* (LLR) of μ as

$$l_\mu = \log \frac{\mu(0)}{\mu(1)}\,. \tag{34}$$

If we now rewrite the message updating rules (31),(32) in terms of LLRs, we obtain from (32),

$$\begin{aligned} l_{\mu_{x_i \to h_j}} &= \log \frac{\mu_{x_i \to h_j}(0)}{\mu_{x_i \to h_j}(1)} \\ &\overset{(32)}{=} \log \left[\frac{p_1(0)}{p_i(1)} \cdot \prod_{\substack{h_l \text{ adjacent to } x_i \\ l \neq j}} \frac{\mu_{h_l \to x_i}(0)}{\mu_{h_l \to x_i}(1)} \right] \\ &= l_{\mu_{x_i \to h_j}} + \sum l_{\mu_{h_l \to x_i}}\,, \end{aligned} \tag{35}$$

which leads to the operator P in (5).

The derivation of

$$l_{\mu_{h_j\to x_i}} = 2\,\mathrm{atanh} \prod_{\substack{k\neq i:\\ x_k \text{ adjacent to } h_j}} \tanh\frac{l_{\mu_{x_k\to h_j}}}{2} \quad (36)$$

from (31) is slightly more involved and details are omitted due to space constraints. However, the derivation can be found in [27, pp. 58f]. The main ingredients are the following: Consider a factor node of degree $J+1$. For $x_i = 0$ in (31), the sum

$$\sum_{x_k:\, k\neq i} 1_{\mathcal{N}(h_j)}(\underline{x}_j)\cdot \text{some product}$$

can be reduced to a summation over those x_k, that add up to 0 (in $\mathbb{F}_2$), and similarly for $x_i = 1$. Then the identity

$$\prod_{i=1}^{I}(r_i+1)+\prod_{i=1}^{I}(r_i-1) = 2\sum_{\substack{x_1,\ldots,x_I:\\ x_1+\ldots+x_I=0}} \prod_{i=1}^{I} r^{(1-x_i)}$$

leads to

$$l_{\mu_{h_j\to x_i}} = \log\frac{1+\prod_i \frac{r_i-1}{r_i+1}}{1-\prod_i \frac{r_i-1}{r_i+1}},$$

where $r_i = \mu_{h_j\to x_i}(0)/\mu_{h_j\to x_i}(1)$. Now using the relations $l = \log r$ and $\frac{r-1}{r+1} = \tanh\frac{l}{2}$, formula (36) follows, leading to the operator S in the dynamical system formulation (5). The input u in (5) is the vector of log-likelihood ratios $u_i = \log\frac{p_{Y_i|X_i}(y_i,0)}{p_{Y_i|X_i}(y_i,1)}$.

We see that instead of the computationally costly sum of products in (31) and a product in (32), the computational burden reduces essentially to one product in (36) and one sum in (35), as in practice the hyperbolic tangent and its inverse could be implemented using lookup tables.

Instead of LLRs, other single-number representations of messages are possible, leading to further equivalent formulations of the BP algorithm [6].

A.8. Turbo estimation: Non-constant inputs

Our last point is to argue why non-constant inputs to our system (5) are worthwhile to consider:

In Section A.3 we have assumed that the receiver would know σ^2. In practice that is not the case and the receiver will have to make a guess $\hat{\sigma}^2$ for σ^2. This guessing is called *channel estimation*. To make a good guess, the receiver might employ information obtained from the decoder after, say, a few iterations, but possibly before a valid codeword has been found. The receiver might then utilize this additional knowledge in the estimation of the channel parameter. Conversely, the updated knowledge of the channel parameter changes the input to the decoder. At this point the decoder could either restart, or continue but with changed probabilities p_i, i.e., we have non-constant inputs. This procedure is termed *turbo-estimation* (turbo as in turbo charger in a car, where the exhaust warms up the air fed into the engine), and ongoing research indicates that it improves the decoding performance even further.