Soft decoding of Raptor codes over AWGN channels using Probabilistic Graphical Models

Rian Singels, J.A. du Preez and R. Wolhuter
Department of Electrical and Electronic Engineering
University of Stellenbosch
Email: rsingels@gmail.com

***Abstract*—Raptor codes are a class of Fountain codes that can reach data transmission rates at the capacity of the Binary Erasure Channels (BEC's) and has, therefore, been extensively researched and refined for hard-decoding over these channels. These Raptor codes are ideal for packet transmission over the internet as they are a real world realization of the BEC's. This article extends the Raptor codes for soft-decoding, investigates its performance over Additive White Gaussian Noise (AWGN) channels, which can potentially be used for wireless broadcast with asynchronous data access. We start by reviewing the conventional problem formalization for Fountain codes and consider the constraints common for both hard- and soft-decoding. We explain how the traditional belief propagation (BP) update rules may be transformed in order to avoid computation over large distributions. The loopy BP algorithm (a.k.a. sum-product algorithm) as well as 2 improvements, i.e. expectation propagation and inactivation decoding, are applied for soft-decoding in order to maximise the usage of the information available at the decoder. Simulation results over the AWGN channel of the decoding methods are compared.**

I. Introduction

The pursuit for data transmission at rates close to the channel capacity has long been sought and, until recently, has merely been considered as theoretically feasible [1]. Then, in May 1993, a revolution occurred in coding theory with the release of a paper [2] that, almost by accident, combined sparse-graph codes with low-complexity iterative decoding thereby changing the way we approach error correcting codes. Today we have many codes, including turbo codes [2] and variations of low-density parity-check (LDPC) codes [3], [4] that perform close to channel capacity and may be implemented with *low-complexity* decoding algorithms. However, these codes have design difficulties due to their fixed code rates. Furthermore, they require that all packages be received and are received in order. These restrictions render these codes unsuitable for broadcasting [5].

Another class of codes, called fountain codes, does not suffer from these restrictions. The "digital fountain approach" concept was first introduced in [6] for communication over an erasure channel with unknown erasure probability. What makes fountain codes appropriate for broadcasting is that they are naturally *rateless*, i.e. for a given finite set of source packets, a fountain code can theoretically generate an infinite sequence of coded output packets. This is done while still retaining low complexity for both the encoding and decoding and therefore, remain viable codes for time constraint applications.

The first practical realization of a fountain code was the LT-code as introduced by M. Luby in [7] for Binary Erasure Channels (BEC). Initially, the LT-code divides the data file into equal packets, called source symbols, and encodes these into output symbols. Each of these output symbols is a linear combination of a random subset of the source symbols, as defined by an *output degree distribution*. These output symbols are then transmitted over the channel and decoded at the receiver to render the original source symbols. Due to the randomisation of the dependencies of the output symbol to the source symbols, the receiver may receive the output symbols in any order. Furthermore, as long as enough output symbols are received, we can successfully decode even when some output symbols are lost completely. This is ideal for broadcasting as explained in [5]. Unfortunately it turns out that, for reliable decoding, the length of the LT-code increases greatly with smaller increases of the code's block length.

In this paper we will focus on an extension of the LT-code called the Raptor (*rap*id-*tor*nado) code. Raptor codes first encode the initial source symbols with a sparse-graph code (usually LDPC codes), which we refer to as the pre-code, and thereafter continues to encode these new symbols (which we will refer to as input symbols) with the LT-code. This solves the problem of super-linear growth and yields linear time encoders and decoders [8]. This article also extends the Raptor codes for soft-decoding, which can potentially be used for wireless broadcast with asynchronous data access.

II. Raptor codes

A. Graph structure of Raptor codes

The factor graph of an example Raptor code is presented in Fig. 1 and consists of 2 parts. The first, uppermost part of the graph is the pre-code with the input symbol gained after the initial source symbols were encoded with this pre-code. The pre-code factors define their XOR relationship to each other. In the lower part of the graph we have the LT-code to encode the output symbols as defined by the LT-factors, which define each output symbols' XOR relationship to the input symbols. This graph structure applies to the transmitter and receiver alike, although at the receiver only the output symbols are received and the original input symbols are inferred from them, using this graph and some approximate inference algorithm.

B. Information transition over factors

We can define the decoding problem, from a probabilistic graphical model point of view, as the distribution of collected information over the factor graph. Initially the input symbol are undetermined at the receiver before decoding and each output symbol contains some information of the rest of the graph. Once we have collected enough output symbols we

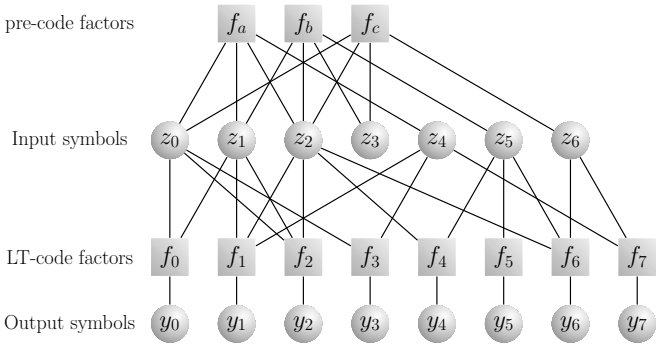


Fig. 1. A schematic diagram of a Raptor code. In this example, $D = 4$ source symbols was encoded at the transmitter to $N = 7$ input symbols using a (7,4) Hamming pre-code [9]. These packets were then encoded into $M = 8$ output symbols with a weakened LT-code. These output symbols are then transmitted to the receiver. At the receiver the input symbols are unknown, thus it uses the received output symbols to infer (decode) the input symbols. Note that the LT-code has failed to connect z_3 to any LT-factor, however this symbol can be recovered through the pre-code.

will have enough information to determine the correct values of the input symbols, with high certainty.

However, having enough information in the global sense is not the only necessity for successful inference. We also require enough localised information to successfully transfer the information in the output symbols to the input symbols. Specifically, we need enough information over each factor in order to perform inference over it.

Take, for example, the LT-factor f_0 in Fig. 1. The relationship between its connected symbols is

$$y_0 \oplus z_0 \oplus z_1 = 0$$

where the modulo-2 addition (XOR) is represented by the symbol $\oplus$. Now let us assume that we know $y_0 = 1$ and nothing of z_0 and z_1. The previous equation then simplifies to

$$z_0 \oplus z_1 = 1$$

This still leaves us with 2 possible solutions. Thus, knowing only one symbol is not enough information to obtain a unique solution. In fact, for a factor of N dependent symbols, we need to know the value of $N - 1$ symbols to obtain a unique solution.

This problem is true for soft-decision inference as well. In fact, the probability of the symbol we wish to compute will be more or less equal to the symbol with the lowest probability. Moreover, if 1 of the symbols other than the one being computed has a probability distribution of $\{0.5, 0.5\}$, the resulting distribution will be the same. It is left for the reader to prove that this is true.

We therefore require a very specific graph structure in order to successfully propagate the information in each output symbol across the entire graph. For larger codes we may construct such a code by using an output degree distribution.

C. Degree distributions for fountain codes

As was just shown, to ensure a high probability of successful decoding, we need the output symbols' dependencies to obtain such a structure that it ensures successful inference over all LT-factors. Although the output symbols' dependencies are randomised, we can try to choose the input degree of these LT-factors in such a way that no redundancy exists in the structure of the graph, while ensuring proper inference. We would like to be able to transfer information to at least 1 new input symbol that does not yet contain any information at every iteration. For the theoretical Raptor code with endless output symbols, this can be achieved by the *ideal soliton distribution* [7], [10] defined as follows

$$\rho(d) = \begin{cases} \frac{1}{N} & \text{for } d = 1 \\ \frac{1}{d(d-1)} & \text{for } d = 2, 3, \ldots, N \end{cases}$$

where d is the input degree of the LT-factor and N is the total number of input symbols.

In practice, however, we will always have only a finite number of output symbols and in this case the ideal soliton distribution performs poorly. In [7] some adjustments were made to improve the distribution's performance in practice. The result is called the *Robust soliton distribution* and is defined as follows

$$\tau(d) = \begin{cases} \frac{S}{dN} & \text{for } d = 1, 2, \ldots, (\frac{N}{S} - 1) \\ \frac{S}{N} \ln(\frac{S}{\delta}) & \text{for } d = \frac{N}{S} \\ 0 & \text{otherwise} \end{cases}$$

where $S \equiv c \ln(\frac{N}{\delta})\sqrt{N}$ for some suitable choice of c and δ. For more on how to choose these parameters the reader is referred to [7].

D. Encoding of Raptor codes

Let us assume a data file of length L which we wish to transmit. To minimise packet overhead we choose the symbol size l and divide the data file into $D = \frac{L}{l}$ symbols. These symbols are then encoded using some sparse code such as LDPC codes (in Fig. 1 we used the Hamming (7,4) code). We then generate the output symbols on-the-fly according to the following steps:

1) Choose the degree d of the new output symbol randomly from a degree distribution such as the Robust soliton distribution.
2) Choose uniformly d different random input symbols as dependencies of the new symbol.
3) The output symbol will be the result of the XOR on the d chosen input symbols to satisfy even parity.

Output symbols are produced until decoding is successful.

III. Belief propagation

The decoding of Raptor codes are most often done using the Belief-Propagation (BP) algorithm (a.k.a the sum-product algorithm). In the following section we explain how the traditional BP update rules can be transformed in order to avoid computation over large distributions and how they are used to decode the Raptor code.

A. Conversion of update rules

The BP algorithm uses message-passing over bipartite Tanner graphs [11] (such as the one in Fig. 1) to decode a transmitted code word. More specifically, these graphs consist of 2 disjointed sets; variable nodes and factor nodes. A message passed along the edge between an arbitrary symbol x_n (which may be any input or output symbol) and factor f_m is one of the two following possible update rules, as defined in [9], [12]:

Updating rule from variable to factor:

$$\mu_{x_n \to f_m}(x_n) = \prod_{f_i \in \mathbf{f}_n \setminus f_m} \mu_{f_i \to x_n}(x_n) \quad (1)$$

Updating rule from factor to variable:

$$\mu_{f_m \to x_n}(x_n) = \sum_{\mathbf{x}_m \setminus x_n} \left(f_m(\mathbf{x}_m) \prod_{x_i \in \mathbf{x}_m \setminus x_n} \mu_{x_i \to f_m}(x_i) \right) \quad (2)$$

Considering that Raptor codes consist of discrete variables, the number of entries in the distribution of $f_m(\mathbf{x}_m)$ is exponential to the number of symbols in $\mathbf{x}_m$ i.e. $2^{|\mathbf{x}_m|}$. For sparse graphs such as those of LDPC codes, the factors remain small and thus these equations suffice for low complexity and reliable inference. However, due to the Robust soliton distribution introduced in Section II we are almost guaranteed to have factors of high density. These dense factors greatly increases the complexity of decoding and computations done over their distributions become intractable to the extent that decoding is NP-hard.

Fortunately we can avoid computation over these large distribution tables by making the observation that our factors enforce even parity over their dependencies. This allows us to re-express the BP algorithm for the special case of even parity, using the Log Likelihood Ratio (LLR) form [12], [13].

Firstly we would like to define a function to describe the even parity constraint. We know that even parity is the equivalent of the modulo-2 addition (XOR). For the case of an 2 independent Binary Random Variables (BRVs) x and y, with probability distributions $P(x) = \{p_0^{(x)},\ p_1^{(x)}\}$ and $P(y) = \{p_0^{(y)},\ p_1^{(y)}\}$, the probability for even parity is:

$$\begin{aligned} P(x \oplus y = 0) &= p_0^x p_0^y + p_1^x p_1^y \\ &= (p_0^x + p_1^x)(p_0^y + p_1^y) + (p_0^x - p_1^x)(p_0^y - p_1^y) \\ &\quad - (p_0^x p_0^y + p_1^x p_1^y) \\ &= \frac{1}{2}((p_0^x + p_1^x)(p_0^y + p_1^y) + (p_0^x - p_1^x)(p_0^y - p_1^y)) \end{aligned}$$

By induction we can extend this for an arbitrary finite amount of BRV's x_i for $i = 0, 1, 2 \ldots, N$. Then we have:

$$P\left(\sum_{i=0}^{N} \oplus\ x_i = 0\right) = \frac{1}{2}\left(\prod_{i=0}^{N}(p_0^{(i)} + p_1^{(i)}) + \prod_{i=0}^{N}(p_0^{(i)} - p_1^{(i)})\right) \quad (3)$$

$$P\left(\sum_{i=0}^{N} \oplus\ x_i = 1\right) = \frac{1}{2}\left(\prod_{i=0}^{N}(p_0^{(i)} + p_1^{(i)}) - \prod_{i=0}^{N}(p_0^{(i)} - p_1^{(i)})\right) \quad (4)$$

Notice that the term $\prod_{i=0}^{N}(p_0^{(i)} + p_1^{(i)}) = 1$, however we will not implement this simplification as this will not be the case once we convert these equations to the LLR domain.

We will define the LLR of the BRV x as $L(x) = \ln\left(\frac{P(x=0)}{P(x=1)}\right)$. This also implies the following relationships:

$$P(x = 0) = \frac{e^{L(x)}}{1 + e^{L(x)}} \quad (5)$$

$$P(x = 1) = \frac{1}{1 + e^{L(x)}} \quad (6)$$

Furthermore, considering that the incoming message in equation (2) are of independent symbols in LLR form, it is useful to use the symbol $\boxplus$ as the notation for the addition defined as:

$$L(x_1) \boxplus L(x_2) \boxplus \ldots \boxplus L(x_N) \equiv L(x_1 \oplus x_2 \oplus \ldots \oplus x_N) \quad (7)$$

To retain mathematical integrity the following additional rules also apply [13]:

$$\begin{aligned} L(x) \boxplus \ \infty &= \ L(x) \\ L(x) \boxplus -\infty &= -L(x) \\ L(x) \boxplus \ 0 \ &= \ 0 \end{aligned}$$

Using equations (3) to (6) we can extend equation (7) such that

$$\begin{aligned} \sum_{i=0}^{N} \boxplus\ L(x_i) &\equiv L\left(\sum_{i=0}^{N} \oplus\ x_i\right) \\ &= \ln\left(\frac{\prod_{i=0}^{N}\left(e^{L(x_i)} + 1\right) + \prod_{i=0}^{N}\left(e^{L(x_i)} - 1\right)}{\prod_{i=0}^{N}\left(e^{L(x_i)} + 1\right) - \prod_{i=0}^{N}\left(e^{L(x_i)} - 1\right)}\right) \end{aligned}$$

Using the identity $\tanh(\frac{x}{2}) = \frac{e^x - 1}{e^x + 1}$ we can simplify

$$\sum_{i=0}^{N} \boxplus\ L(x_i) \equiv \ln\left(\frac{1 + \prod_{i=0}^{N} \tanh\left(\frac{L(x_i)}{2}\right)}{1 - \prod_{i=0}^{N} \tanh\left(\frac{L(x_i)}{2}\right)}\right)$$

furthermore, if we define $\kappa \equiv \prod_{i=0}^{N} \tanh\left(\frac{L(x_i)}{2}\right)$ we have

$$\begin{aligned} \tanh\left(\frac{1}{2}\sum_{i=0}^{N} \boxplus\ L(x_i)\right) &= \frac{e^{\ln\left(\frac{1+\kappa}{1-\kappa}\right)} - 1}{e^{\ln\left(\frac{1+\kappa}{1-\kappa}\right)} + 1} \\ &= \frac{(1+\kappa) - (1-\kappa)}{(1+\kappa) + (1-\kappa)} \\ &= \kappa \end{aligned}$$

Therefore we conclude that

$$\sum_{i=0}^{N} \boxplus\ L(x_i) \equiv 2 \tanh^{-1}\left(\prod_{i=0}^{N} \tanh\left(\frac{L(x_i)}{2}\right)\right)$$

Let us now consider the BP updating rules (1) and (2). To help distinguish between the linear and LLR domains we will define the following relationship

$$\begin{aligned} \lambda_{x_n \to f_m}(x_n) &\equiv L(\mu_{x_n \to f_m}(x_n)) \\ \lambda_{f_m \to x_n}(x_n) &\equiv L(\mu_{f_m \to x_n}(x_n)) \end{aligned}$$

It is thus easy to show that the updating rule for variable to factor message (1) now becomes:

$$\begin{aligned} L(\mu_{x_n \to f_m}(x_n)) &= L\left(\prod_{f_i \in \mathbf{f}_n \setminus f_m} \mu_{f_i \to x_n}(x_n)\right) \\ \lambda_{x_n \to f_m}(x_n) &= \sum_{f_i \in \mathbf{f}_n \setminus f_m} \lambda_{f_i \to x_n}(x_n) \end{aligned} \quad (8)$$

The LLR transformation of the updating rule for factor to variable is somewhat more elaborate. As mentioned before,

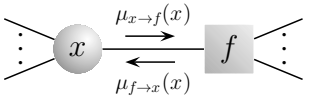


Fig. 2. A sub-graph of a arbitrary factor graph, showing a symbol x, factor f and the two types of messages sent along the edges of a graph that are involved in the BP algorithm. x will only transmit $\mu_{x\to f}(x)$ once it has received a message from all other factors it shares an edge with. Similarly, factor f will not transmit $\mu_{f\to x}(x)$ before it has received all messages from all the symbols it is connected to.

$f_m(\mathbf{x}_m)$ enforces even parity and thus the summation over this distribution can be represented as a modulo-2 addition, such that

$$\sum_{\mathbf{x}_m\setminus x_n} f_m(\mathbf{x}_m) \prod_{x_i\in\mathbf{x}_m\setminus x_n} P(x_i) \equiv P\left(\sum_{x_i\in\mathbf{x}_m\setminus x_n}{\oplus}\, x_i\right)$$

With this knowledge we may continue our transformation of the updating rule for factor to variable messages (2):

$$\begin{aligned}
&L(\mu_{f_m\to x_n}(x_n)) \\
&= L\left(\sum_{\mathbf{x}_m\setminus x_n}\left(f_m(\mathbf{x}_m)\prod_{x_i\in\mathbf{x}_m\setminus x_n}\mu_{x_i\to f_m}(x_i)\right)\right) \\
&\lambda_{f_m\to x_n}(x_n) \\
&= L\left(\sum_{x_i\in\mathbf{x}_m\setminus x_n}{\oplus}\prod_{x_i\in\mathbf{x}_m\setminus x_n}\mu_{x_i\to f_m}(x_i)\right) \\
&= \sum_{x_i\in\mathbf{x}_m\setminus x_n}{\boxplus}\,\lambda_{x_i\to f_m}(x_i)
\end{aligned}$$

which leaves us with the new updating rule

$$\lambda_{f_m\to x_n}(x_n) = 2\tanh^{-1}\left(\prod_{i=0}^{N}\tanh\left(\frac{\lambda_{x_i\to f_m}(x_i)}{2}\right)\right) \quad (9)$$

B. Message passing

The BP algorithm was originally designed for tree-like Tanner graphs and, unfortunately, Raptor codes in general contain many loops. However, these loops are generally large enough that the code is tree-like over the span of localised sub-graphs which allows us to use the LBP algorithm.

LBP is an iterative process where the messages (equations (8) and (9)) are passed across the graph until convergence. This algorithm will not produce exact inference as in the case of tree-like graphs, nor does it guarantee convergence. However, despite these shortfalls it has been proved to achieve good decoding performance and thus remains useful in practice [14].

The message passing schedule applied in this paper is called *message flooding*. We start by initiating all messages from variables symbols to become

$$\text{for all } n \text{ and } m\text{:}\quad \lambda_{x_n\to f_m}(x_n) = 0$$

which is often referred to as lazy initialisation. This causes all factor nodes to receive a message across all of their edges, enabling them to transmit a new message back to those same variable symbols in turn. If Fig. 2 is considered, the message $\lambda_{x\to f}(x)$ will initially be 0. The factor f receives this message and all other messages across all other edges connected to that. It then updates it's messages using equation (9) and returns message $\lambda_{f\to x}(x)$ and all other messages across all other edges. Symbol x goes through a similar process using equation (8). This process continues until convergence occurs.

IV. Improvements on Belief propagation

A. Tree-structure Expectation propagation

As mentioned before, one fundamental problem with fountain codes is that enough input degree 1 ($\mathcal{N}=1$) LT-factors are required in order to initiate the BP algorithm. Conventionally this problem is overcome by introducing very large fountain codes. However with very little or no increase in complexity we can use the relationship of the input and output symbols, as defined by the LT-factors, to change the structure of the factor graph in order to obtain the necessary $\mathcal{N}=1$ LT-factors.

An algorithm called Tree-structure Expectation Propagation (TEP) is proposed and analysed in [15]. It changes the graph structure to decode LDPC codes over the Binary Erasure Channel (BEC). In [16] this algorithm was analysed for the LT-code over the BEC. Here we will adapt the algorithm for Raptor/LT code over the Additive White Gaussian Noise (AWGN) channels (i.e. soft-decoding).

TEP is only triggered once the BP fails as described in Section II-B. The algorithm starts by searching for the first $\mathcal{N}=2$ LT-factor (f_0) that is a function of the two input symbols (z_0 and z_1) as depicted in Fig. 3. Once found, this factor is added to a list which will exclude it from searches in future iterations of the algorithm. Thereafter, one of the two input symbols is chosen (z_1). All LT-factors, other than the originally chosen factor (f_0), that shares an edge with both z_0 and z_1 have both edges removed. Each LT-factor sharing an edge with z_1, but not with z_0, has its link to z_1 removed and a link to z_0 added. Furthermore, all output symbols connected to f_0 (y_0) are connected to each factor that had its links to z_1 and/or z_0 removed. Finally, once the graph restructuring is done, the algorithm will try to implement BP again and if decoding is not successful the process will be repeated.

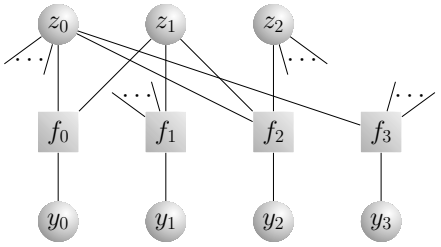
In the example of Fig. 3 the TEP algorithm is successful in producing a $\mathcal{N}=1$ factor (f_2). In fact the TEP algorithm will only be successful if and only if a $\mathcal{N}=3$ LT-factor share both the input symbols of the chosen $\mathcal{N}=2$ LT-factor with it. However, it is shown in [15] that the probability of producing a $\mathcal{N}=1$ factor increases exponentially after each iteration of the algorithm.

The TEP algorithm is based upon a basic observation that the relationship as defined by the LT-factor can be viewed as simultaneous XOR equations. Thus, for the instance in Fig. 3a we have,

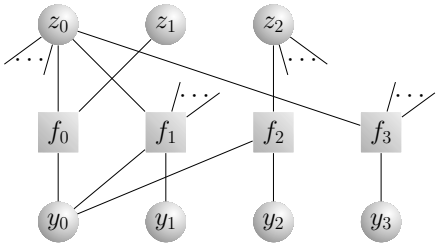
$$\begin{aligned}
y_0 &= z_0 \oplus z_1 & y_2 &= z_0 \oplus z_1 \oplus z_2 \\
y_1 &= z_1 \oplus \dots & y_3 &= z_0 \oplus \dots
\end{aligned}$$

The order of the variables does not matter, thus we may also write the first equation as $z_1 = y_0 \oplus z_0$. We can now substitute this into the other equation to remove z_1, leaving us with

$$\begin{aligned}
y_0 &= z_0 \oplus z_1 & y_0 \oplus y_2 &= z_0 \oplus z_0 \oplus z_2 \\
y_0 \oplus y_1 &= z_0 \oplus \dots & y_3 &= z_0 \oplus \dots
\end{aligned}$$

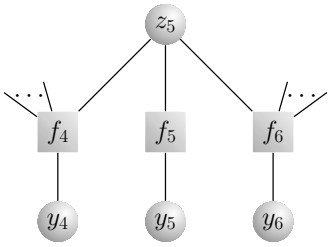


(a) Before

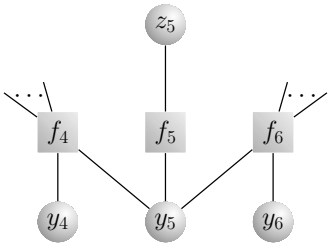


(b) After

Fig. 3. A sub-graph of the Raptor code in Fig. 1. Note that it only consists of the LT section of the code. (a) The sub-graph before 1 iteration of TEP, with f_0 as the chosen $\mathcal{N} = 2$ factor and z_1 is to be isolated. Here f_2 shares both input symbols with f_0, where f_1 and f_3 only shares one of the 2 symbol. (b) The sub-graph after 1 iteration of TEP where we can see that f_2 has now become a $\mathcal{N} = 1$ factor. f_1's edges have also been altered, whereas f_3 remains the same.



(a) Before



(b) After

Fig. 4. Another sub-graph of the Raptor code in Fig. 1. Fig (a) shows the sub-graph before 1 iteration of ID, with z_5 as the symbol to be recovered. Fig (b) shows the sub-graph after ID. This is possible because factor f_5 defines $y_5 = z_5$, therefore replacing z_5 with y_5 in factors f_4 and f_6 retains the validity of their distributions.

Note that a variable XOR'ed with itself is 0, therefore the third equation reduces to $y_0 \oplus y_2 = z_2$. These final equations now represent the relationships in Fig. 3b.

B. Inactivation decoding

The following algorithm, termed *Inactivation decoding* (ID), was developed in order to combine the decoding success rate of Gaussian elimination with the low complexity of belief propagation [17] and is analysed for BEC's in [5]. Once again we will make minor adaptations to the algorithm for soft decoding over AWGN channels.

Before the algorithm starts, all input symbols are considered active symbols. Only active symbols are included when calculating $\mathcal{N}$ of a LT-factor. The algorithm starts by searching for a LT-factor of $\mathcal{N} = 1$. Once found it performs Gaussian elimination on the LT-constraint matrix (or equivalent operations on the factor graph), thus eliminating the active input symbol from all other LT-factors. Just as with TEP, we add a link between all output symbols connected to the $\mathcal{N} = 1$ LT-factor and those which had the variable removed. The corresponding input symbol of the $\mathcal{N} = 1$ LT-factor is now considered *recovered* (and thus is no longer included when calculating $\mathcal{N}$) and is excluded from the rest of the algorithm until BP is applied to the final factor graph. A simple example is depicted in Fig. 4, where input symbol z_5 is to be recovered via LT-factor f_5. ID replaces the edges z_5 to f_4 and z_5 to f_6 with the edges y_5 to f_4 and y_5 to f_6, respectively.

This process continues until no $\mathcal{N} = 1$ exists. If this is the case, the algorithm *inactivates* an input symbol that has not yet been recovered. By doing so the inactivated symbol will be ignored when choosing a factor to eliminate, i.e. all factors connected to the inactivated symbol have their input degree reduced by one. After a symbol has been inactivated, the algorithm once again searches for a LT-factor of $\mathcal{N} = 1$. If none is found another input symbol is inactivated. This continues until all input variables are ether recovered or inactivated. This adds another constraint to the ID: there needs to be at least as many LT-factors (thus output symbols) as there are input symbols. Finally, BP is applied to the new factor graph.

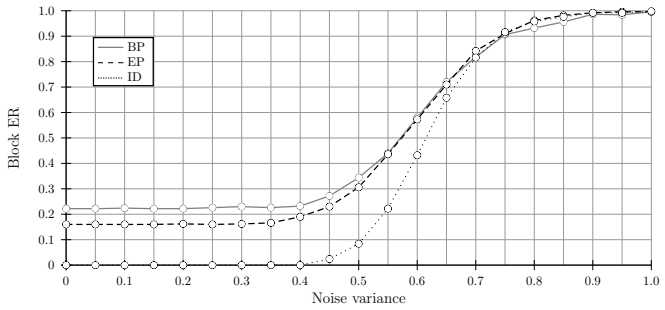
In many ways this algorithm is very similar to TEP, in fact the instance depicted in Fig. 3 can be viewed as a special case of recovery of z_1, where z_0 is inactivated. However ID manipulates the graph structure to a greater extent and does not wait for BP to fail before initiating. Therefore both these algorithms end up with vastly different factor graphs and, as we shall see, vastly different decoding performances.

V. Performance Evaluation

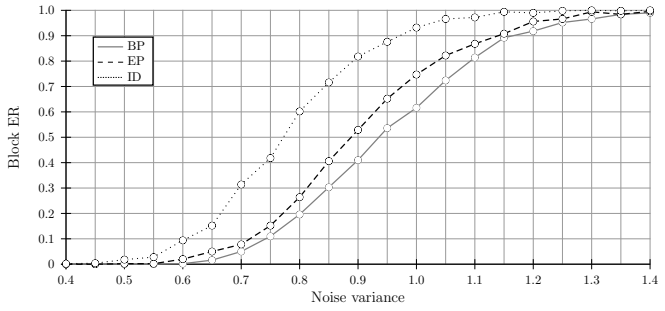
A. Simulation parameters

For testing transmission over the binary AWGN channel, we used binary inputs $\{-1, +1\}$ to represent input bit values $\{0, 1\}$ respectively. We tested the 3 decoding algorithms described in this paper on a Raptor code with the precode of the R10 code (a standard code used in the industry [5]). A small block size of $B = 50$ bits was used to reduce simulation time. Although these small codes do not fare as well as larger codes, they suffice to depict the behaviour of the 3 algorithms. The symbol size over all simulations was set to 1 bit as we assumed synchronous communication between sender and receiver, thereby negating the necessity for package headers. Subsequently the same performance will be obtained for larger symbol sizes.

Simulations were run to compare the rate of block decoding failure (block error rate) against the noise variances σ. σ is incremented in steps of 0.05 from 0 to 1, and each data point is the result over 500 iterations. The overhead was chosen as 30% and 100%, where the overhead is measured as the percentage of output symbols compared to the input symbols. Unfortunately, the differences between hard- and soft-decoding are of such a nature that comparing them is nonsensical, thus we do not compare our simulation result over AWGNC to that of conventional BEC.



(a) 30% overhead



(b) 100% overhead

Fig. 5. The 2 graphs display the simulation results for the cases of 30% and 100% overhead, respectively. In fig. (a) all 3 algorithms perform well. Here ID outperforms BP and TEP considerably, with almost no block errors at low noise levels. One may also observe that all 3 algorithms' decoding performances decline rapidly after $\sigma = 0.5$. In fig. (b) all 3 algorithms show improvements in performance compared to that of fig. (a). However, ID's increase in performance is far less significant than those of BP and TEP and as a result they now outperform ID.

B. Simulation results

In Fig. 5a we show the results for transmission with an overhead of 30% for $\sigma = 0$ (absence of noise) to $\sigma = 1.0$. As we can see, TEP marginally improves on the performance of BP for low levels of noise, as is to be expected. However ID greatly outperforms them both, reaching very small error rates for σ as high as 0.4.

Considering only this evidence, we may deem ID the far superior decoding algorithm, which is the case for BECs. However Fig. 5b shows interesting results that suggest otherwise. Here we show the results for transmission with an overhead of 100% for $\sigma = 0.4$ to $\sigma = 1.4$. Although all 3 algorithms' performance increase with the overhead, BP and TEP shows a far greater improvement than ID to the extent that they outperform it for higher levels of noise. This slow improvement of the decoding performance of ID in the presence of noise can be contributed to 2 phenomena. The first being the dense loops created between the output symbols (that retain the initial information) when the graph is restructured. This implies that the assumption of localised tree-like sub-graphs is no longer valid and therefore, BP fails. The second cause is due to the fact that far more weight is placed upon output symbols involved early during the ID algorithm, thereby not utilising all given information fully.

VI. Conclusions and future work

In this paper we considered the Raptor codes for soft decoding and investigated its performance over binary AWGN channels. We started by reviewing fountain codes and considered its constraints common to both hard- and soft-decoding. We explained how to construct the BP update rules in order to avoid computation over large distributions. We continued to apply and simulate the loopy BP, EP and ID algorithms on a Raptor code for soft-decoding and compared the results.

It is clear that, although ID has a good performance at low overhead, it is outperformed by BP and TEP at higher overhead levels. In contrast, BP and TEP has a poorer performance at low noise levels, but has a greater resistance against increasing noise levels if a large overhead is provided.

Future work include investigating the junction tree algorithm to avoid loopy inference and the necessity for $\mathcal{N} = 1$ LT-factors. This may be combined with techniques to approximate the very large clusters of junction trees. Furthermore, damping techniques might be used on the messages in order to reduce the oscillations of message values to in effect, improve message convergence and thus decoding performance.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 2, may 1993, pp. 1064 –1070 vol.2.

[3] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 599 –618, feb 2001.

[4] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 619 –637, feb 2001.

[5] A. Shokrollahi and M. Luby, "Raptor codes," *Foundations and Trends in Communications and Information Theory*, vol. 6, no. 3-4, pp. 213–322, 2011. [Online]. Available: http://dx.doi.org/10.1561/0100000060

[6] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 56–67, October 1998. [Online]. Available: http://doi.acm.org/10.1145/285243.285258

[7] M. Luby, "Lt codes," *Foundations of Computer Science, IEEE Annual Symposium on*, vol. 0, p. 271, 2002.

[8] P. Pakzad and A. Shokrollahi, "Design principles for raptor codes," in *Proceedings of the IEEE Information Theory Workshop*, 2006, pp. 165–169.

[9] D. J. C. MacKay, *Information theory, inference and learning algorithms*, 4th ed. Cambridge University Press, March 2003.

[10] ——, "Fountain codes," in *The IEE Seminar on Sparse-Graph Codes*. London: IEE, 2004, pp. 1–8.

[11] T. Richardson and R. Urbanke, *Modern Coding Theory*. New York, NY, USA: Cambridge University Press, 2008.

[12] F. R. Kschischang, B. J. Freyy, and H.-A. Loeliger, "Factor graphs and the sum product algorithm," vol. 47, no. 2, pp. 498–519, February 2001.

[13] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[14] Y. Weiss, "Belief propagation and revision in networks with loops," Massachusetts institute of technology, Tech. Rep., November 1997.

[15] P. Olmos, J. Murillo-Fuentes, and F. P. andrez Cruz, "Tree-structure expectation propagation for decoding ldpc codes over binary erasure channels," in *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, June 2010, pp. 799 –803.

[16] M. A. Guede, "Optimization of the belief propagation algorithm for luby transform decoding over the binary erasure channel," Master's thesis, Delft University of Technology, Augustus 2011.

[17] A. Shokrollahi, S. Lassen, and R. Karp, "Systems and processes for decoding chain reaction codes through inactivation," *U.S. Patent number 6 856 263*, February 2005.

Rian Singels received his bachelors degree in Electrical & Electronic Engineering from the University of Stellenbosch, South Africa in 2010. He is currently completing his masters degree in the same field at the University of Stellenbosch. His current studies focuses on the assessment of Raptor codes on soft-decision inference with probabilistic graphical models.