# Linear Programming-Based Decoding of Turbo-Like Codes and its Relation to Iterative Approaches

Jon Feldman
Laboratory for Computer Science
MIT, Cambridge, MA, 02139
jonfeld@theory.lcs.mit.edu

David R. Karger*
Laboratory for Computer Science
MIT, Cambridge, MA, 02139
karger@theory.lcs.mit.edu

Martin Wainwright
Electrical Engineering and Computer Science
UC Berkeley, CA, 94720
martinw@eecs.berkeley.edu

**Abstract**

In recent work (Feldman and Karger [8]), we introduced a new approach to decoding turbo-like codes based on linear programming (LP). We gave a precise characterization of the noise patterns that cause decoding error under the binary symmetric and additive white Gaussian noise channels. We used this characterization to prove that the word error rate is bounded by an inverse polynomial in the code length. Furthermore, for any turbo-like code, our algorithm has the *ML certificate* property: whenever it outputs a code word, it is guaranteed to be the maximum-likelihood (ML) code word.

In this paper we extend these results and give an iterative decoder whose output is equivalent to that of the LP decoder. We also extend the ML certificate property to the more efficient iterative *tree reweighted max-product* message-passing algorithm developed by Wainwright, Jaakkola, and Willsky [13]: we show that whenever this algorithm converges to a code word, it must be the ML code word.

Finally, we demonstrate experimentally that the noise patterns that cause decoding error in the LP decoder also cause decoding error in the standard iterative sum-product and max-product (min-sum) message-passing algorithms. Consequently, the deterministically constructible interleaver used by the LP decoder to achieve its bounds on error rate is useful in practice not only for the LP decoder, but for these standard iterative decoders as well.

## 1 Introduction

The introduction of turbo codes [3] surprised the field of coding theory by achieving error probabilities that far outperformed any other code at the time. A great deal of subsequent work has focused on design, implementation, and analysis of turbo codes and their variants [12], called “turbo-like” codes. One of the goals of this research has been to achieve a theoretical understanding of the remarkably good performance of turbo-like codes.

A lot of progress has been made [7, 2, 11] using “code ensembles,” where all interleavers are considered simultaneously, leading to an understanding of the asymptotic behavior of a randomly chosen interleaver. The natural extension of this work is to identify the precise noise patterns under which decoding error occurs, derive fixed interleavers, and prove tight bounds on the error rate using these interleavers.

*Research supported by NSF contract CCR-9624239 and a David and Lucille Packard Foundation Fellowship.

In this paper, we give a new iterative algorithm to decode any turbo-like code. When applied to a repeat-accumulate (RA) code, with an appropriate choice of step size, the algorithm is guaranteed to converge to a certain fixed point. Additionally, the error patterns for which this fixed point is the original transmitted code word are well characterized. This allows us to prove that the the algorithm has a WER that is bounded by an inverse-polynomial in the block length under the binary symmetric channel (BSC) or the additive white Gaussian noise (AWGN) channel. This bound holds for (i) *any* code length, (ii) a specific, deterministically constructible interleaver and (iii) a simple message-passing decoder. Additionally, the algorithm has the *ML certificate* property: whenever it outputs a code word, it is guaranteed to be the ML codeword.

**1.1 Techniques.** Our analysis extends recent work by Feldman and Karger [8], where we introduced a new approach to decoding turbo-like codes based on *linear programming* (LP) *relaxation*, a standard technique for finding approximate solutions to difficult optimization problems. This approach allowed us to isolate a cycle-like structure called a *promenade* in a graph $G$ modeling the code, where $G$ is related to the *factor graph* of the code. We proved that our LP decoder produces an error if and only if the total noise across some promenade in $G$ exceeds a certain level; we call this a *noisy promenade*. We then used this theorem to design an interleaver, and proved that the WER of the LP decoder is bounded by an inverse polynomial in the block length, as long as the noise in the channel is below a certain constant threshold.

A similar structural analysis has recently been used to produce results for other codes under the binary erasure channel (BEC): Di et al. [6] showed that under the BEC, iterative decoding of low-density parity-check (LDPC) codes produces an error if and only if the pattern of erased bits forms a "stopping set" in the underlying graph.

To obtain the new iterative decoder, we reformulate the linear program in its dual form, and invoke a standard optimization technique. Under an appropriate choice of step size, this gives an iterative algorithm that is guaranteed to produce the same output as the LP decoder. However, it may take a long time to converge in practice, since this algorithm is inherently more "careful" about its steps than the standard iterative methods. This motivates exploring other more aggressive iterative algorithms and drawing connections to them as well.

Wainwright, Jaakkola and Willsky [13] recently proposed an approach to computing optimal configurations in factor graphs based on tree-reweighted max-product (TRMP) message-passing updates. Here we prove that when the TRMP algorithm reaches a certain stopping condition, then it has found the ML code word. We prove this by relating the algorithm directly to the dual form of the LP.

We also shed some light on the mysterious good behavior of the more conventional iterative decoders used for decoding turbo-like codes. We show experimentally that the existence of a noisy promenade is strongly correlated to the success of the sum- and max-product algorithms. Specifically, under the BSC with a crossover probability of at most $p \leq 10^{-1.5}$, using rate-1/2 RA codes, the observed WER for both the max- and sum-product algorithms, regardless of the interleaver, is always less than $1.43 \times 10^{-5}$. In contrast, in the presence of a noisy promenade, the WER always exceeds $.6$. It is then no surprise that under max- and sum-product decoding, the interleaver we designed (to make noisy promenades rare) far outperforms a random one.

**1.2 Outline.** The remainder of the paper is organized as follows. Section 2 provides background on the problem of decoding RA codes. In Section 3, we review the LP-based approach of Feldman and Karger. In Section 4, we describe iterative methods for solving the linear program, and our results concerning the stopping condition of the TRMP algorithm of Wainwright et al. Section 5 contains a discussion of our experimental work. We conclude in Section 6 with a discussion of open problems and future work.

## 2 Background

The class of *turbo-like codes* (as defined by Divsalar and McEliece [7]) is defined as any combination of convolutional codes concatenated in serial and in parallel, with interleavers between each constituent code. Repeat-accumulate (RA) codes (introduced in the same paper [7]) are perhaps the simplest non-trivial example of turbo-like codes. Their simple structure and highly efficient encoding scheme make them both practical and simpler to analyze than other more complex turbo-like codes. In fact, Kahale and Urbanke [10] show that the asymptotic distance of a random interleaver for any parallel concatenated convolutional code depends only on the number of parallel branches, not on the distance of the code; this suggests that accumulators make as good a choice as any convolutional code for use as the constituent code.

Let RA($R$) denote a rate-$1/R$ RA code. In this paper, we use the RA(2) code as a running example of a turbo-like code because its simple structure allows clear presentation. The LP decoder has a natural analogue for any turbo-like code, as do the iterative algorithms we give in this paper. When appropriate, we give intuition on generalizations to RA($R$) codes. We leave the general form for turbo-like codes for a later version.

**RA(2) Encoding.** The encoder for an RA(2) code takes the information word, repeats every bit twice, then sends it through an *interleaver* (known permutation), and then through an *accumulator*. The accumulator maintains a partial sum (modulo two) of the input seen so far, and outputs the new sum at each step.[1]

More formally, let $(x_0, \ldots, x_{k-1})$ be a binary information word, and let $n = 2k$. Let $\pi$ be a permutation on $\{0, \ldots, n-1\}$. The vector $z \in \{0,1\}^n$ will represent the repeated and permuted information word; specifically, for all $i \in \{0, \ldots, k\}$, we let $z_{\pi(2i)} = z_{\pi(2i+1)} = x_i$. The RA(2) code using $\pi$, given $x$, outputs a code word $(y_0, \ldots, y_{n-1})$, where for all $t \in \{0, \ldots, n-1\}$, $y_t = \sum_{i=0}^{t} z_t \mod 2$. Furthermore, we define $X_i$ to be the set of 2 code word indices to which information bit $i$ was repeated and permuted. Formally, $X_i = \{\pi(2i), \pi(2i+1)\}$, $i \in \{0, \ldots, k-1\}$. Let $\mathcal{X} = \{X_i : 0 \le i < k\}$. In RA($R$) codes, each bit is repeated $R$ times before being sent through the permutation and the accumulator, and thus the sets $X_i \in \mathcal{X}$ each have $R$ members.

**The Accumulator Trellis.** Figure 1 shows the trellis $T$ for the accumulator. We will use $\{s_0^0, \ldots, s_n^0\}$ and $\{s_1^1, \ldots, s_{n-1}^1\}$ to refer to the sets of "0-parity" and "1-parity" nodes of the trellis, respectively.[2] Node $s_t^\alpha$ represents the encoder being in state $\alpha$ at time $t$. The trellis contains edges between nodes on consecutive segments of the trellis; i.e., for all $\alpha, \beta \in \{0,1\}$, $t \in \{0, \ldots, n-1\}$ where $s_t^\alpha, s_{t+1}^\beta \in T$, there is an edge $(s_t^\alpha, s_{t+1}^\beta)$. A "0-edge" is one that transitions between nodes of the same parity ($\alpha = \beta$), and a "1-edge" transitions between nodes of different parity ($\alpha \neq \beta$); these edge types, respectively, represent information bits `0` and `1` and have solid and dashed lines in figure 1.

The state transitions of an accumulator from time 0 to time $n$ can be represented by a *path* $P$ in the trellis from $s_0^0$ to $s_n^0$. The *label* $\ell(s_t^\alpha, s_{t+1}^\beta)$ of an edge $(s_t^\alpha, s_{t+1}^\beta)$ represents the output of an encoder using that transition edge of the trellis, so $\ell(s_t^\alpha, s_{t+1}^\beta) = \beta$, for all $\alpha, \beta \in \{0,1\}$. If we concatenate the labels $\ell(e)$ on edges $e$ along $P$, we get the code word output by the accumulator. The "types" of the edges on $P$ (either 0-edge or 1-edge) represent the repeated and permuted information word $z$ that was provided as input to the accumulator.

[1] So as to keep the paper simpler, we assume that the input contains an even number of `1`s. This can be achieved by padding the information sequence with an extra parity bit. Thus the rate of this code is actually $(k-1)/2k$, or $1/2 - o(1)$. The $o(1)$ can be avoided by a more technical argument.

[2] Since we know that the start state is 0, we do not need the node $s_0^1$. Since we know that there are an even number of `1`s, we do not need a node $s_n^1$.

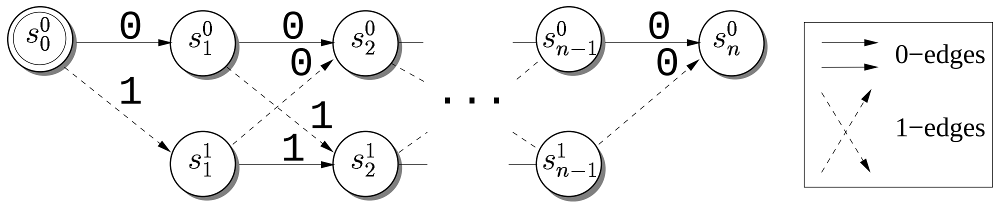

**Figure 1.** The trellis for an accumulator, used in RA codes. The edges are of two types, "0-edges" and "1-edges," corresponding to information bits `0` and `1`, respectively. The edges are labeled with their associated output (code) bit.

Let $(\hat{y}_0, \ldots, \hat{y}_{n-1})$ represent the noisy symbols received from the BSC or AWGN channel. The *cost* $c(s_t^\alpha, s_{t+1}^\beta)$ of an edge is a measure of the probability that the encoder entered state $\beta$ at time $t+1$, given that it was in state $\alpha$ at time $t$, and given the received symbol $\hat{y}_t$. For the BSC the cost is the Hamming distance between $\ell(s_t^\alpha, s_{t+1}^\beta)$ and the received symbol $\hat{y}_t$; for the AWGN channel, the cost is the Euclidean distance. The cost of a path is the sum of the costs of the edges on the path, and the minimum cost path is the path most likely taken by the encoder.

**Decoding.** Not every path in the trellis corresponds to a valid codeword, which is why the the Viterbi algorithm, which simply finds the most likely path from $s_0^0$ to $s_n^0$, does not work. Let $x_i$ be some arbitrary information bit, where $X_i = \{t, \hat{t}\}$; suppose for illustrative purposes that $x_i = 1$. Since $x_i$ constitutes the input bit to the accumulator at time $t$ *and* at time $\hat{t}$, any path through $T$ representing a valid encoding would take a 1-edge at time step $t$ *and* at time step $\hat{t}$. In general, any path representing a valid encoding would do the same thing (as in take a 1-edge or a 0-edge) at time steps $t$ and $\hat{t}$. We say a path is *agreeable* for $x_i$ if it has this property for $x_i$. An *agreeable path* is a path that is agreeable for all $x_i$. A path is agreeable if and only if it represents a valid encoding. The goal of the ML decoder, then, is to find the most likely agreeable path from $s_0^0$ to $s_n^0$, which we will refer to as the *ML agreeable path*.

## 3 RALP: Repeat-Accumulate Linear Program

The algorithm of Feldman and Karger [8] is based on the technique of *linear programming relaxation*. In this technique, an optimization problem is first expressed as an *integer program* (IP), which consists of a set of (typically binary) variables under linear constraints, with a linear objective function. Since IPs are NP-hard to solve, the variables are *relaxed* to take on real values in a fixed range, resulting in a solvable *linear program* (LP).

We illustrate the technique on the problem of finding the ML agreeable path in the trellis. Let $\mathcal{P}$ represent the set of all paths from $s_0^0$ to $s_n^0$ in the trellis. We have a variable $f_P$ for each path in $\mathcal{P}$, and a variable $\bar{x}_i$ for each information bit $x_i$. The IP enforces $f_P \in \{0,1\}$. Let $Z_t$ be the set of paths that use a 1-edge at trellis segment $t$. Let $c[P] = \sum_{e \in P} c[e]$ denote the cost of a path. We also define linear constraints and a linear objective function:

$$\text{RALP:} \quad \min \sum_{P \in \mathcal{P}} c[P] f_P \quad \text{s.t.}$$

$$\sum_{P \in \mathcal{P}} f_P = 1$$

$$\sum_{P \in Z_t} f_P = \sum_{P \in Z_{\hat{t}}} f_P = \bar{x}_i \quad \forall \{t, \hat{t}\} \in \mathcal{X} \quad \text{(agreeability)}$$

Note that we have defined an an exponential number of variables, which typically makes even the LP relaxation inefficient to solve. However, we can use the concept of a *network flow* (see the book [1] for background) to express the LP with only a linear number of variables.

For details on this representation of RALP, we refer the reader to the paper of Feldman and Karger [8].

A *feasible solution* to an integer or linear program is a setting of the variables such that all the constraints are satisfied. An *optimal solution* is a feasible solution that minimizes the objective function over all feasible solutions. If we enforce the constraints $f_P \in \{0,1\}$, then all feasible solutions to the resulting IP will have $f_P = 1$ for some agreeable path $P$ and $f_{P'} = 0$ for all $P' \neq P$. Furthermore, an *optimal* solution would be an ML agreeable path. Unfortunately, we cannot solve IPs efficiently, so we relax the constraint $f_P \in \{0,1\}$ to $0 \leq f_P \leq 1$ to make it a solvable LP.

We say a solution to an LP is *integral* if all its values are integers. If the optimal solution to an LP relaxation is integral, then it must be the optimal solution to the associated IP, since all feasible solutions to the IP are feasible for the LP. Therefore, if our LP optimal solution has $f_P = 1$ for some path $P$, $P$ must be the ML agreeable path. In this case we know the ML code word, and can output the associated information word by simply examining the variables $\bar{x}_i$.

If the solution is not integral, we get get a values $f^*_P$ for each path $P$, constituting an *agreeable distribution*; this is a distribution on paths where for all information bits $x_i$, where $X_i = \{t, \hat{t}\}$, the total mass on paths that take a 1-edge at time $t$ is equal to the mass on paths that take a 1-edge at time $\hat{t}$. In this case, we concede a decoding error. Alternatively, we could request a retransmission, produce an erasure, or try to guess at a solution; this depends on the application.

To actually solve the LP, the simplex algorithm [1] is the usual choice, but it can be slow sometimes in practice. However, there are efficient alternatives to using a general purpose LP solver for the case of RA(2) codes, where RALP can be solved by finding a (standard) min-cost flow in a certain auxiliary graph whose size is the same as the trellis [9]. This fact allows us to use various efficient methods for solving min-cost flow problems [1]. In Section 4 of this paper, we prove that the RALP can also be solved for arbitrary turbo-like codes by running iterative message-passing algorithms.

Finally, we note that RALP has a general form for any turbo-like code. For example, for RA($R$) codes, the sets $X_i \in \mathcal{X}$ have size $R$, and we enforce agreeability among all $t \in X_i$. Formally, we enforce the agreeability constraint $\sum_{P \in Z_t} f_P = \bar{x}_i$ for all $t \in X_i$.

**3.1 Error pattern conditions and provable error bounds for RA codes.** An advantage of casting the decoding problem as the solution to a linear program is that the noise patterns that cause decoding error can be well characterized. We have identified a particular graph $G$ modeling the code and a kind of subgraph of $G$ called a *promenade*; we proved that our LP decoder produces an error if and only if the total noise across some promenade in $G$ exceeds a certain level [8]. We call this a *noisy promenade*.

The structure of the graph $G$ is dictated by the interleaver $\pi$, and the "noise" on a promenade depends on the symbols received from the channel. Our theorem suggests a design criterion for the interleaver: it should be chosen so as to induce a graph in which the probability of a noisy promenade is low. The theorem holds for any turbo-like code [9], but for the case of RA(2) codes, we provide [8] a deterministic interleaver construction that obeys this design criterion. We also prove that by using this construction, the probability of the graph having a noisy promenade is bounded by an inverse-polynomial in the block length $n$.

In the remainder of this section, we formally define a promenade for RA(2) codes, and the conditions under which it is considered "noisy" for the BSC (the conditions for the AWGN channel are similar). We also describe how this definition can be generalized to any RA($R$) code. We then state our bounds [8] for RA(2) codes, under both the BSC and AWGN, on the probability of a noisy promenade.

**Promenades.** Let $G$ be an undirected line graph with $n$ nodes $(g_0, \ldots, g_{n-1})$, and an edge between every pair of consecutive nodes. Let the *cost* $c(g_t, g_{t+1})$ of edge $(g_t, g_{t+1})$ be $-1$ if the $t^{th}$ bit of the transmitted codeword is flipped by the channel, and $+1$ otherwise.

Let $P(j, j')$ denote the subpath $(g_j, \ldots, g_{j'})$. Let $F$ be an arbitrary set of subpaths of this form. For some node $g_t$ in $G$, let $\deg_t(F)$ be the number of subpaths in $F$ that start or end at $g_t$. Formally, $\deg_t(F) = |\{P(j, j') \in F : j = t \text{ or } j' = t\}|$. We say $F$ is a *promenade* if for all information bits $x_i$, where $X_i = \{t, \hat{t}\}$, $\deg_t(F) = \deg_{\hat{t}}(F)$. The cost $c(F)$ of a promenade $F$ is the sum of the costs of its subpaths. Formally, $c(F) = \sum_{P(j,j') \in F} \sum_{j}^{j'-1} c(g_j, g_{j+1})$. A promenade is *noisy* if its cost is less than or equal to zero.

**Theorem 1. [8]** *RALP decodes correctly iff $G$ has no noisy promenades.*

For a proof of this theorem, we refer the reader to the paper of Feldman and Karger [8]. Every edge of $G$ has negative cost $(-1)$ with the crossover probability $p < \frac{1}{2}$, and positive cost (+1) otherwise. Therefore, promenades with more edges are less likely to have a total negative cost. For the case of RA(2) codes, the promenade takes a simple form. If extra (zero-cost) edges are added to $G$ between $g_t$ and $g_{\hat{t}}$ for every $\{t, \hat{t}\} \in X_i$, then a promenade can be described as a type of *tour* through $G$ (non-simple cycle). The *girth* of a graph is the length of its shortest simple cycle. Since every promenade contains at least one simple cycle, graphs with high girth will have promenades with many edges.

Therefore, from the perspective of the RALP decoder, a good interleaver is one that produces a graph $G$ with high girth. An $O(n^3)$-time construction of Erdös [5] yields a graph $G$ with girth $\Theta(\log n)$, and it is straightforward to derive the *Erdös interleaver* from $G$. Theorem 1, along with this interleaver construction and a union bound over the promenades in $G$, gives an analytical bound on the probability of error of RALP decoding:

**Theorem 2. [8, 9]** *For any $\epsilon > 0$, the rate $1/2$ RA code with block length $n$ using the Erdös interleaver decoded with a RALP decoder, under the binary symmetric channel (BSC) with crossover probability $p < 2^{-4(\epsilon + (\log 24)/2)}$, or under the AWGN channel with noise variance $\sigma^2 \leq \frac{\log e}{4 + 2\log 3 + 4\epsilon}$, has a word error probability $P_w$ of at most $n^{-\epsilon}$.*

As $\epsilon \to 0$, the noise threshold for the BSC approaches $p \approx 2^{-9.17}$. For the AWGN, the noise threshold approaches $\sigma^2 \approx .201$, which corresponds to an SNR of $E_b/N_0 \approx 6.96$. We refer to this bound as the *path bound* because it is based on an analysis of the paths that make up the promenade. Figure 3 shows how the path bound compares to our experiments, where we measured the probability of a noisy promenade under the BSC. The figure also includes a tighter *tree bound* based on a different analysis [9]; the drawback to this bound is that we do not yet have a closed-form expression for it. We note that Theorem 2 also implies the same bound on ML decoding.

For RA($R$) codes, the degree constraint for $x_i$ on subpaths is enforced across all $R$ segments in $X_i$. For example, for RA(3), we say that $F$ is a promenade if, for all $x_i$ where $X_i = \{t, t', t''\}$, $\deg_t(F) = \deg_{t'}(F) = \deg_{t''}(F)$. In this case, if hyperedges $\{g_t : t \in X_i\}$ are added to $G$, promenades can be described as a "hyper-tour."

## 4 Iterative algorithms

In this section, we provide iterative algorithms for solving RALP. We begin by developing a particular Lagrangian dual formulation, the optimal value of which is equivalent to the optimal value of the RALP relaxation. As we describe, this dual formulation suggests the use of an iterative subgradient optimization method [4]. Such methods, while guaranteed to converge with appropriate choice of step size, may be slow in practice. We then consider the variant of the max-product algorithm, known as tree-reweighted max-product (TRMP), recently proposed by Wainwright, Jaakkola and Willsky [13] for maximum likelihood calculations on factor graphs. Like the standard max-product and sum-product algorithms, the algorithm is based on simple message-passing updates, so it has the same complexity per iteration as those algorithms. Here we show that TRMP is also attempting to solve a dual formulation of RALP. In addition, we prove that when TRMP reaches a certain stopping condition, it has found an optimal integral point of RALP, and thus has found the ML code word.

**4.1 Lagrangian Dual.** Without the agreeability constraints, the RALP problem is a standard shortest path problem. This observation motivates a partial Lagrangian dualization procedure, wherein we deal with the troublesome agreeability constraints using Lagrange multipliers. More specifically, for a particular path $P$, the "agreeability" of that path with respect to some $X_i \in \mathcal{X}, X_i = \{t, \hat{t}\}$ can be expressed by the following function $A$:

$$A_i(P) = \mathcal{I}[P \in Z_t] \; - \; \mathcal{I}[P \in Z_{\hat{t}}] \tag{1}$$

Here $\mathcal{I}[P \in Z_t]$ is an indicator function that takes the value one if path $P$ is in $Z_t$ (uses a 1-edge on segment $t$), and zero otherwise. Note that $A_i(P) = 0$ for all $X_i = \{t, \hat{t}\}$ if and only if $P$ is agreeable. We then consider the Lagrangian obtained by assigning a real-valued Lagrange multiplier $\lambda_i$ to each agreeability constraint:

$$\mathcal{L}(P; \lambda) \;\; = \;\; c[P] \; + \sum_{X_i \in \mathcal{X}} \lambda_i \, A_i(P) \tag{2}$$

For a fixed vector $\lambda \in \mathbb{R}^{|\mathcal{X}|}$, the corresponding value of the dual function $Q(\lambda)$ is obtained by minimizing the Lagrangian over all paths — that is, $Q(\lambda) = \min_{P \in \mathcal{P}} \mathcal{L}(P; \lambda)$ where $\mathcal{P}$ denotes the set of all paths through the trellis. Since the dual is the minimum of a collection of functions that are linear in $\lambda$, it is concave. Moreover, for a fixed $\lambda$, calculating the dual value $Q(\lambda)$ corresponds to solving a shortest path problem on the trellis, where the 1-edges in each paired set of segments $\{t, \hat{t}\} = X_i$, have been reweighted by $\lambda_i$ and $-\lambda_i$ respectively. Since there may not be a unique shortest path, we consider the set $SP(\lambda) = \{P : \mathcal{L}(P; \lambda) = Q(\lambda)\}$ of shortest paths under the weighting $\lambda$.

Note that the value of an agreeable path is unchanged by the reweighting $\lambda$, whereas any non-agreeable path may be affected by $\lambda$. Thus, if a decoder could find a setting of $\lambda$ such that all non-agreeable paths have higher cost under $\lambda$ than the ML agreeable path, then the ML agreeable path would "exposed" by the Viterbi algorithm, since it would be the shortest path under $\lambda$.

From linear programming duality [4], it follows that the optimal value $Q^* = \max_{\lambda \in \mathbb{R}^{|\mathcal{X}|}} Q(\lambda)$ of this Lagrangian dual problem is equal to the optimal value of the RALP relaxation, so that solving this problem is equivalent to solving RALP.

**4.2 Iterative Subgradient Decoding.** An iterative technique to compute $Q^*$ is the subgradient optimization algorithm [4]. For a concave function $Q$, a *subgradient* at $\lambda$ is a vector $d$ such that $Q(\mu) \leq Q(\lambda) + d^T(\mu - \lambda)$ for all $\mu$. For the particular form of $Q$ at hand, it can be shown [4] that the collection of all subgradients is given by the following convex hull:

$$\partial Q(\lambda) \;\; = \;\; \mathrm{CONV}\{\, A(P) \mid P \in SP(\lambda) \,\} \tag{3}$$

The subgradient method is an iterative method that generates a sequence $\{\lambda^m\}$ of Lagrange multipliers. As previously described, for any pair $X_i = \{t, \hat{t}\}$, segment $t$ is reweighted by $\lambda_i^m$, whereas segment $\hat{t}$ is reweighted by $-\lambda_i^m$. At each iteration $m$, the algorithm entails choosing a subgradient $d(\lambda^m) \in \partial Q(\lambda^m)$, and updating the multipliers via

$$\lambda^{m+1} \quad = \quad \lambda^m + \alpha^m d(\lambda^m), \tag{4}$$

where $\alpha^m$ is a step size parameter. Note that a subgradient $d(\lambda^m)$ can be calculated by finding a shortest path in the reweighted trellis $\mathcal{L}(P, \lambda^m)$. With appropriate choice of step size, it can be shown [4] that the sequence $\{\lambda^m\}$ will converge to a dual optimal solution. In practice, however, the rate of convergence may be slow.

**4.3 Iterative TRMP Decoding.** In lieu of the subgradient updates, we now consider the TRMP updates [13], which take a simple form for a turbo code with two constituent codes. The TRMP updates are similar to but distinct from standard max-product (min-sum) updates. Like the subgradient updates of equation (4), the TRMP algorithm generates a sequence of Lagrange multipliers $\{\lambda^m\}$. At each iteration, it uses the reweighted trellis problem $\mathcal{L}(P, \lambda^m)$ to compute two shortest paths for each segment $t$: the shortest path that uses a 1-edge in segment $t$, as well as a competing shortest path that uses a 0-edge. As with standard decoding, it then uses this information to form a min-log likelihood ratio for each trellis segment:

$$\mathrm{LLR}(\lambda^m; t) \quad = \quad \min_{P \in Z_t} \mathcal{L}(P, \lambda^m) - \min_{P \notin Z_t} \mathcal{L}(P, \lambda^m) \tag{5}$$

The value $\mathrm{LLR}(\lambda^m; t)$ corresponds to the difference between the costs of the most likely paths using a 1-edge or a 0-edge respectively on segment $t$. At iteration $m$, the log likelihood ratios can be efficiently computed using a single forward-backward pass of the Viterbi algorithm on the trellis, where $\lambda^m$ is used to reweight the edges appropriately.

The goal of the iterative decoder is to come to a decision for each information bit, i.e., to make the sign of $\mathrm{LLR}(\lambda^m; t)$ agree with the sign of $\mathrm{LLR}(\lambda^m; \hat{t})$ for all $X_i = \{t, \hat{t}\}$. With this goal in mind, the Lagrange multipliers are updated via the recursion

$$\lambda_i^{m+1} = \lambda_i^m + \alpha^m \big( \mathrm{LLR}(\lambda^m; \hat{t}) - \mathrm{LLR}(\lambda^m; t) \big), \tag{6}$$

where $\alpha^m \in (0, 1]$ is a step size parameter. Note that, as with the subgradient updates (4), the sum of the reweighting factors on segments $t$ and $\hat{t}$ is zero at all iterations.

An often used heuristic for standard iterative decoding algorithms is to terminate once thresholding the LLRs yields a valid codeword. With TRMP, we can prove that a heuristic of this form is optimal; specifically, if we terminate when for each segment $\{t, \hat{t}\}$, both of the LLRs have the same sign, then we have found the ML codeword. Formally, call a setting of $\lambda$ an *agreement* if for all $\{t, \hat{t}\} \in \mathcal{X}$, $\mathrm{LLR}(\lambda; t) \cdot \mathrm{LLR}(\lambda; \hat{t}) > 0$. In other words, $\mathrm{LLR}(\lambda; t)$ and $\mathrm{LLR}(\lambda; \hat{t})$ have the same sign, and neither is equal to zero.

**Theorem 3.** *If $\lambda^*$ is an agreement, then $SP(\lambda^*)$ contains only one path $P$ that corresponds to the ML code word.*

*Proof.* Since $\mathrm{LLR}(\lambda^*; t) \neq 0$ for all segments $t$, the type of edge (either 1-edge or 0-edge) used at each segment by any path in $SP(\lambda^*)$ is determined. It follows that $SP(\lambda^*)$ contains only one path $P$. Since $\lambda^*$ is an agreement, we know that for every $X_i \in \mathcal{X}$, where $X_i = \{t, \hat{t}\}$, the type of edge used by $P$ at segment $t$ matches the type used at segment $\hat{t}$, i.e., $A_i(P) = 0$. It follows that $P$ is agreeable, and that $\mathcal{L}(P; \lambda^*) = c[P]$. Since $P \in SP(\lambda)$, $Q(\lambda^*) = \mathcal{L}(P; \lambda^*) = c[P]$.

Now consider the primal solution to RALP that sets $f_P = 1$, and $f_{P'} = 0$ for all $P' \neq P$. The value of this primal solution is $c[P]$. Thus we have exhibited a primal solution $P$ and a dual solution $\lambda^*$ with the same value $c[P]$. By strong duality, they must both be optimal. Thus $P$ is the ML code word. □

**Corollary 4.** *If TRMP finds an agreement, then it has found the ML code word.*

We have not yet shown that TRMP always finds an agreement whenever RALP has an integral solution. Consequently, unlike the subgradient algorithm, we cannot assert that the TRMP result is always equivalent to RALP. However, we have observed identical behavior in experiments on RA codes, and further investigation should deepen the connection between these algorithms.

## 5 Experimental Results

Our experimental work focuses on testing the standard iterative sum-product (iterative MAP) and max-product (min-sum, SOVA) algorithms against the existence of a noisy promenade. Our experiments show that the decoding error of both the sum-product and max-product algorithms are strongly correlated to the existence of a noisy promenade under the BSC (see Figure 2). Specifically, at a crossover probability of at most $p \leq 10^{-1.5}$, when there is no noisy promenade, the observed WER of both algorithms, regardless of the interleaver, is always less than $1.43 \times 10^{-5}$. In contrast, in the presence of a noisy promenade, the WER always exceeds $.6$. Since the Erdös interleaver was designed to make noisy promenades rare, it is no surprise that the performance of the iterative decoders improves markedly (see Figure 3) when the Erdös interleaver is used.

Based on this evidence, we conjecture that for lower rate RA codes, and for other turbo-like codes, the decoding success of sum- and max-product both have a strong correlation to the existence of a noisy promenade analogue. If this were the case, then interleavers designed for LP-based decoding (based on the structure of promenades) would also be useful for the typical (and more efficient) iterative methods.

Our data also show the relationship between the upper bounds from Section 3.1 and the observed probability of the existence of a noisy promenade (see Figure 3) for the BSC. The gap between the bounds and the observed probabilities is most likely due to a union bound used in the analysis. The closed-form path bound of Theorem 1 and the recursively defined tree bound [9] could potentially be improved by a deeper understanding of the distribution of noisy promenades, but the slope of the bounds seems quite accurate.

## 6 Conclusions and Future Work

We have described an iterative algorithm for decoding any turbo-like code whose decoding success is precisely characterized by a combinatorial condition on the noise pattern. For the case of RA codes, this implies an inverse polynomial bound on the error rate of the decoder. Additionally, the decoder has the desirable property that whenever it outputs a code word, it is guaranteed to be the ML code word. We have extended the ML certificate property of the linear programming (LP)-based decoder to the tree-reweighted max-product (TRMP) algorithm, showing that when the TRMP algorithm finds a code word, it is the ML code word.

There are several interesting open question about the relationship between LP decoding and iterative methods. We have shown that when the TRMP algorithm finds a code word, it is the ML code word. To complete this result, we would like to see a proof that when the LP decoder finds the ML code word, the TRMP finds it as well. Experimental evidence suggests that this

|  | $p = 10^{-1}$ |  | $p = 10^{-1.5}$ |  | $p = 10^{-2}$ |  | $p = 10^{-2.5}$ |  |
|---|---|---|---|---|---|---|---|---|
|  | $err$ / $tot$ | $wer$ | $err$ / $tot$ | $wer$ | $err$ / $tot$ | $wer$ | $err$ / $tot$ | $wer$ |
| NP? | Max-Product, Erdös interleaver |  |  |  |  |  |  |  |
| Y | 787506 / 796648 | .989 | 55817 / 59519 | .938 | 1914 / 2099 | .912 | 101 / 112 | .902 |
| N | 4 / 203352 | 2e-5 | 0 / 940481 | 0 | 0 / 997901 | 0 | 0 / 999888 | 0 |
|  | Sum-Product, Erdös interleaver |  |  |  |  |  |  |  |
| Y | 792176 / 796648 | .994 | 55461 / 59519 | .932 | 1918 / 2099 | .914 | 98/ 112 | .875 |
| N | 19258 / 203352 | .0947 | 2 / 940481 | 2e-6 | 0 / 997901 | 0 | 0 / 999888 | 0 |
|  | Max-Product, random interleaver |  |  |  |  |  |  |  |
| Y | 859856 / 886050 | .970 | 176905 / 229738 | .770 | 36153 / 55146 | .656 | 9912 / 16234 | .611 |
| N | 3 / 113950 | 3e-5 | 1 / 770251 | 1e-6 | 0 / 944854 | 0 | 0 / 983776 | 0 |
|  | Sum-Product, random interleaver |  |  |  |  |  |  |  |
| Y | 864673 / 886050 | .976 | 170668 / 229738 | .743 | 35225 / 55146 | .639 | 9935 / 16234 | .612 |
| N | 12042 / 113950 | .106 | 11 / 770262 | 1e-5 | 0 / 944854 | 0 | 0 / 983766 | 0 |

$err$= number of errors  $tot$= total trials  $wer$= word error rate  NP = noisy promenade

**Figure 2.** Word error data for the standard max-product and sum-product iterative decoding algorithms, using the Erdös interleaver, and using a random interleaver (a new random interleaver is picked for each trial). For each of the four combinations, the data are separated into the case where there is a noisy promenade (Y), and when there is not (N). An RA(2) code with block length 128 is used, under the BSC with varying crossover probability $p$. The data show one million trials for each interleaver type.

is the case. Preliminary experiments also show that the more aggressive sum- and max-product algorithms converge faster than subgradient or TRMP decoding, and perform just as well. It would be interesting to see how the insights provided by LP-based decoding carry over to the performance of the sum and max-product algorithms.

Finally, the error bounds we gave [8] for LP decoding, and its iterative extension, were geared specifically for RA(2) codes. Although they trivially hold for higher rate RA codes, we suspect that significantly better bounds can be achieved by a deeper understanding of the combinatorics of "generalized promenades." Achieving this sort of bound requires the construction of an analogue to the Erdös interleaver, to make all promenades "large." There is evidence for a better bound in the work of Kahale and Urbanke [10], where they show that when at least three parallel branches of accumulators are used, the distance of a random set of interleavers approaches $n^{\epsilon}$ (for some constant $\epsilon$ that depends on the number of parallel branches), whereas for two parallel branches, the distance is at most logarithmic in $n$. Thus, since we achieved an inverse polynomial error rate for rate 1/2 (which is exponential in the distance), we would expect to achieve an error rate on the order of $exp(-\Omega(n^{\epsilon}))$ for codes of rate at most 1/3.

## References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, 1993.

[2] Benedetto, Divsalar, Montorsi, and Pollara. Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding. *IEEETIT: IEEE Transactions on Information Theory*, 44, 1998.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo-codes. *Proc. IEEE International Conference on Communication (ICC), Geneva, Switzerland*, pages 1064–1070, May 1993.

[4] D. Bertsimas and J. Tsitsikilis. *Introduction to linear optimization*. Athena Scientific, Belmont, MA, 1997.

[5] N. Biggs. Constructions for cubic graphs with large girth. *Electronic Journal of Combinatorics*, 5(A1), 1998.

[6] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke. Finite length analysis of low-density parity check codes. *IEEE Trans. Info Theory.*, 48(6), 2002.

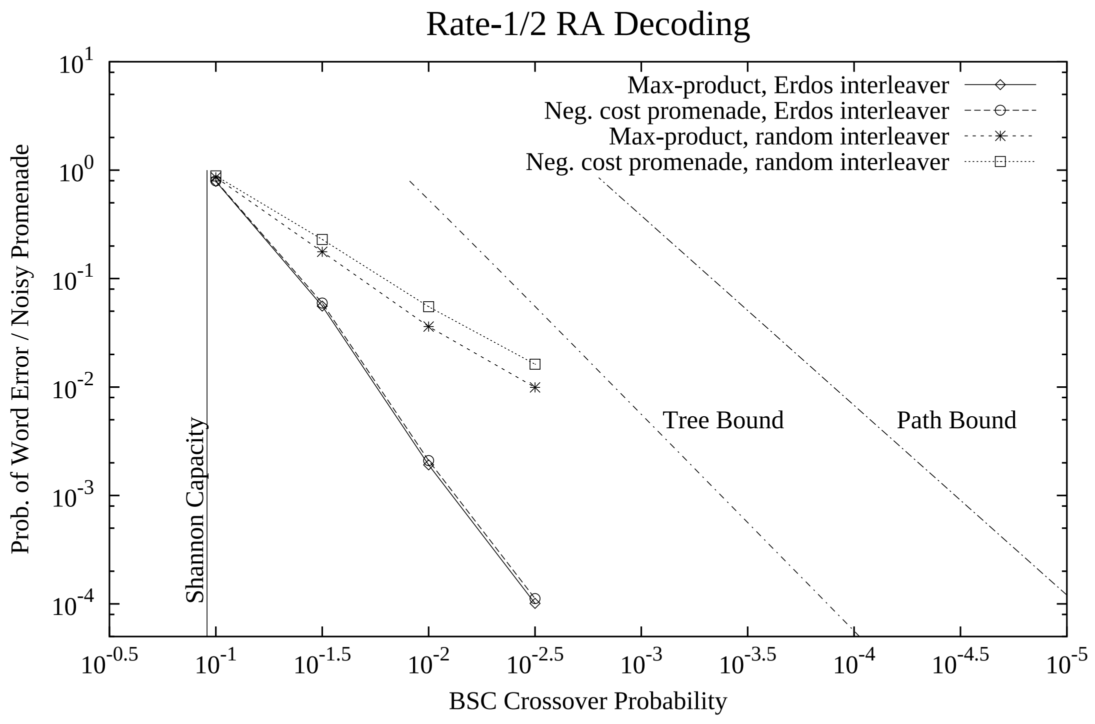

**Figure 3.** Comparison of the WER of the standard max-product algorithm to the frequency of a noisy promenade for both the Erdös interleaver and a random interleaver, taken from the experiments in Figure 2. The sum-product algorithm is not shown, since its rate would not be distinguishable from that of the max-product algorithm at this scale. Also included for reference are two theoretical bounds on WER [8] for the Erdös interleaver. The *path bound* is the closed-form bound of Theorem 1, and the *tree bound* [9] is a recursively defined bound computed specifically for this case.

[7] D. Divsalar, H. Jin, and R. McEliece. Coding theorems for 'turbo-like' codes. *Proc. 36th Annual Allerton Conference on Comm., Control, and Computing*, pages 201–210, Sept 1998.

[8] J. Feldman and D. R. Karger. Decoding turbo-like codes via linear programming. *Proc. 43rd annual IEEE Symposium on Foundations of Computer Science (FOCS)*, November 2002.

[9] J. Feldman and D. R. Karger. Notes on decoding turbo-like codes via linear programming. Manuscript, June 2002.

[10] N. Kahale and R. Urbanke. On the minimum distance of parallel and serially concatenated codes. *IEEE International Symposium on Information Theory*, 1998.

[11] T. Richardson and R. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2), February 2001.

[12] B. Vucetic and J. Yuan. *Turbo Codes*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[13] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. MAP estimation via agreement on (hyper)trees: Message-passing and linear programming approaches. In *Proc. Allerton Conf. Communication, Control, and Computing, to appear.*, October 2002.