Electronic Colloquium on Computational Complexity, Report No. 64 (2011)

# Towards deterministic tree code constructions

Mark Braverman*

**Abstract**

We present a deterministic operator on tree codes – we call tree code product – that allows one to deterministically combine two tree codes into a larger tree code. Moreover, if the original tree codes are efficiently encodable and decodable, then so is their product. This allows us to give the first deterministic subexponential-time construction of explicit tree codes: we are able to construct a tree code $\mathcal{T}$ of size $n$ in time $2^{n^\varepsilon}$. Moreover, $\mathcal{T}$ is also encodable and decodable in time $2^{n^\varepsilon}$.

We then apply our new construction to obtain a deterministic constant-rate error-correcting scheme for interactive computation over a noisy channel. If the length of the interactive computation is $n$, the amount of computation required is deterministically bounded by $n^{1+o(1)}$, and the probability of failure is $n^{-\omega(1)}$.

*University of Toronto. Email: mbraverm@cs.toronto.edu. Supported by an NSERC Discovery Grant.



ISSN 1433-8092

# 1 Introduction

In this paper we study ways to implement communication protocols over a noisy channel, and related fundamental coding-theory questions. Classical coding theory addresses the following practical problem. Suppose Alice wants to send Bob a message $m$ over a channel that is affected by a certain amount of noise. How should Alice encode $m$ so that if the amount of noise is not excessive Bob would be able to recover the message $m$? For an encoding/decoding scheme to be useful, the following three properties need to hold: (1) Bob should be able to recover $m$ from a corrupted version $C(m)'$ of the encoded message $C(m)$; (2) the ratio $\frac{|C(m)|}{|m|}$, called *rate* of the code, is small, i.e. $C(m)$ is not much longer than $m$; and (3) the encoding/decoding algorithms are efficient. In the past 60 years, scientist and engineers have been extremely successful in addressing this coding challenge. In particular, "good codes", satisfying the properties above, have been successfully constructed [MS77, Sud01]. For example, there are good efficient codes that are able to deliver the message even if the adversary corrupts, say, 20% of the symbols in the message $C(m)$.

Suppose that instead of the simple task of sending a message, Alice and Bob are trying to implement a more complex protocol $\pi$ over the channel. Is there a similarly good way to encode $\pi$ using a code $C(\pi)$ so that even if some of the messages in the encoded protocol $C(\pi)$ are corrupted, the parties would still be able to simulate the functionality of $\pi$? When this problem arises in practice, such as in TCP/IP protocols, this problem is reduced to treating each message of $\pi$ separately through a round-by-round encoding. Note that, however, even if the noise on the channel is random and not adversarial, this approach requires an $\Omega(\log n)$ bit overhead per round to keep the probability of failure at each round below $1/n$. Thus if the messages of the original protocol $\pi$ are short, this approach introduces a blowup in the amount of communication required. Another problem is that if the noise is adversarial, this approach is doomed to fail, since the adversary may choose to corrupt entirely the encoding of a single message in $\pi$, thus completely derailing the execution of the protocol.

In fact, it is far from obvious that a constant-rate encoding scheme that protects *the entire protocol* from an adversary that is allowed to corrupt a constant fraction of the messages exists. It is not even clear that such a scheme exists when the noise is random. In a breakthrough result, Schulman [Sch96] introduced a new coding-theory primitive, called the *tree code* (see Section 2.2 below for the definition of tree codes). Schulman showed how tree codes can be used to obtain a constant-rate encoding of $\pi$ that protects it against an adversary that is allowed to corrupt a constant fraction of the communication symbols. Schulman's original construction dealt with an adversary that corrupts at most a $1/240$ fraction of the input. In a subsequent work, [BR10] devised a different protocol that deals with adversarial error rates of up to $1/4 - \varepsilon$. The main drawback of all these schemes is the fact that while tree codes can be shown to exist via the probabilistic method, no explicit constant-rate constructions with efficient encoding and decoding are known.

In an unpublished memo [Sch03] Schulman outlines a construction (joint with Will Evans and Michael Klugerman) of a $\frac{1}{\log n}$-rate deterministic tree code, where $n$ is the length of $\pi$. Combined with the results in [BR10], one can obtain a fairly efficient scheme for error-proofing interactive communication. Unfortunately, the scheme falls short of being "good", because it has a subconstant $\frac{1}{\log n}$ rate. A very promising recent approach by Gelles and Sahai [GS11] is to show that objects weaker than tree codes, called potent tree codes, suffice for the main application. While finding potent tree codes is provably easier, there are no known constructions of efficiently encodable/decodable potent tree codes either. Finally, one can only look at *randomized* errors, rather

than adversarial ones. It turns out that in this case one only needs a much weaker object called a local tree code. As this object generally can be represented using $O(\log n)$ bits, and thus exhaustive search techniques can be applied to it. This was first noted informally by Schulman in [Sch03]. A different construction has been formally presented by Moitra in [Moi11]. Both constructions yield a constant-rate encoding scheme for a channel affected by random noise. The scheme fails with probability $1/n^c$ for some constant $c$. The scheme requires time polynomial in $n$ to set up, and the encoding/decoding time is also polynomial in $n$, where the exponent depends on the security parameter $c$. For the random noise model the construction of Gelles and Sahai delivers a much better performance, requiring *expected* time $O(n)$ to encode/decode, and having an exponentially small error probability.

## Contributions

Our main technical contribution is Theorem 6, which gives a new way to combine two tree codes to obtain a larger tree code. If a tree code $\mathcal{T}_1$ is of size $d_1$ (i.e. it encodes strings of length $d_1$), and a tree code $\mathcal{T}_2$ is of size $d_2$, then the newly obtained product tree code $\mathcal{T} = \mathcal{T}_1 \otimes \mathcal{T}_2$ will be of size $d = \Omega(d_1 \times d_2)$. This gives a way of constructing larger tree codes from smaller ones. In particular, one can construct a tree code of size $n$ from a constant number of copies of a tree code of size $n^\varepsilon$ using the $\otimes$ operator. Since a tree code of size $n^\varepsilon$ can be found by brute force in time $2^{O(n^\varepsilon)}$, we obtain a subexponential $2^{O(n^\varepsilon)}$-time deterministic algorithm for constructing a tree code of size $n$. Moreover, the resulting code is also encodable and decodable in time $2^{O(n^\varepsilon)}$. A formal statement of the result on tree code product requires us to define tree codes first, and thus is deferred until Section 3.

Using the $\otimes$ operators we can construct a code of size $\log n$ from codes of size $\log^\varepsilon n$. Codes of size $\log^\varepsilon n$ can be found in time $2^{\log^\varepsilon n} = n^{o(1)}$. Thus we obtain a way of constructing a tree code of size $\log n$ (and, in fact, of size $\log^C n$ for any $C > 1$) in subpolynomial time. By combining this new construction with ideas from [Sch03, Moi11], we show (see Corollary 9 below):

**Theorem A.** *For every noise rate $\beta < 1/2$, and any parameter $c > 1$, there is a deterministic transformation of communication protocols $C_{\beta,c}(\pi)$, such that for every binary protocol $\pi$, $C_{\beta,c}$ is a* binary protocol, *$|C_{\beta,c}(\pi)| = O_{\beta,c}(|\pi|)$, and for all inputs $x, y$, both parties can determine $\pi(x, y)$ correctly with probability $> 1-2^{-\Omega((\log n)^c)}$, by running $C_{\beta,c}(\pi)$ if the channel is affected by a random noise of rate $< \beta$. Moreover, the encoding/decoding of each symbol takes time $2^{O((\log n)^{1/c})}$, and thus the computational resources required to execute $C_{\beta,c}(\pi)$ are bounded by $n \cdot 2^{O((\log n)^{1/c})} = n^{1+o(1)}$.*

The theorem has two advantages over prior deterministic constructions. Firstly, it is very efficient: the computational overhead is a subpolynomial multiplicative factor. Secondly, it achieves a subpolynomial failure probability. The theorem is incomparable to the construction of [GS11]. The running time in Theorem A is higher by a factor of $n^{o(1)}$, and the error probability in Theorem A is $2^{-\Omega((\log n)^c)}$ rather than exponential in $n$. On the other hand, the construction in Theorem A is completely deterministic, and the running time is $n^{1+o(1)}$ *in the worst case*, rather than in expectation. In addition, the error-correction scheme in Theorem A also works for adversarial noise models, as long as the adversary is not allowed to place a disproportionately high fraction of errors in any single stretch of length $(\log n)^c$. It is very possible that the two constructions can be fused to obtain parameters that dominate both.

In addition to the immediate benefits in terms of better error-correcting schemes for interactive communication, we believe that our construction can potentially be a building block in constructing

efficient tree codes. The reason why we cannot apply the $\otimes$ operator repeatedly to obtain a highly efficient tree code of arbitrary size is that the encoding rate of $\mathcal{T}_1 \otimes \mathcal{T}_2$ is worse than the rate of $\mathcal{T}_1$ and $\mathcal{T}_2$. Thus we are only allowed a constant number of tree code product applications if we are to keep the rate from falling below a constant. One possible avenue of attack if one were to apply $\otimes$ more than a constant number of times is to come up with a variant of the $\otimes$ operator that does not cause such a significant rate drop. A different context in Theoretical Computer Science where such an approach has recently been successful is the zig-zag product of graphs [RVW02] which is a variant of graph squaring that manages to keep the degree from being squared while preserving some desirable properties of graph squaring.

The rest of the paper is organized as follows. In Section 2 we state a lemma about (non-tree) codes that we will use in this paper; we then define tree codes, local tree codes, as well as what it means for these objects to be efficiently encodable and decodable. In Section 3 we give our main tree code product construction. In Section 4 we show how to use the tree code product construction to obtain better explicit tree codes and local tree codes, as well as to obtain better deterministic error-correcting schemes for interactive communication over a noisy channel.

# 2 Preliminaries

## 2.1 Good (non-tree) codes

An error-correcting code is a function $C : \Sigma^n \to \Sigma^m$ such that the message $m$ can be recovered from $C(m)$ even if some of the symbols in $C(m)$ are corrupted. There are several parameters involved: the alphabet size $\sigma = |\Sigma|$, the amount (and type) of error one can recover from, and the rate $\frac{m}{n}$ of the encoding. The problem generally gets harder when $\sigma$ becomes smaller, when the errors are large (constant-rate) and adversarial, and when we would like to keep the rate constant. A code with a constant alphabet size, a constant rate and which can withstand constant-rate errors is said to be good. It is a well known fact from coding theory [Jus72, MS77, Sud01] that good error-correcting codes exist, and moreover can be encoded and decoded efficiently. We will use the following reformulation of this fact. We omit the proof here, since it is a standard corollary of the existence of good codes, combined with an application of an expander code (cf. [SS96]):

**Lemma 1.** *For each $\Sigma_1$ of size $\sigma_1 = |\Sigma_1|$, and for each pair of parameters $(n, \varepsilon)$ there is an alphabet $\Sigma_2$ of size $\sigma_2 = \sigma_1^{O(1)}$ and a code $C : \Sigma_1^n \to \Sigma_2^n$ with the following properties:*

1. **Distance:** *for $w_1 \neq w_2$, $H(C(w_1), C(w_2)) > (1 - \varepsilon)n$.*
2. **Efficient encoding:** *$C(w)$ can be computed in time $poly(n, \log \sigma_2)$.*
3. **Efficient decoding:** *there is a partial function $D : \Sigma_2^n \to \Sigma_1^n$ computable in time $poly(n, \log \sigma_2)$, such that for each $w \in \Sigma_2^n$, whenever there is an $w' \in \Sigma_1^n$ with $H(C(w'), w) < \frac{1}{2}(1 - \varepsilon)n$, $D(w) = w'$.*

## 2.2 Tree codes and local tree codes

Tree codes were introduced by Schulman [Sch96], and are currently the only known tool that allows good error-correcting coding for interactive computation. For a string $s$, we denote by $s[i..j]$ the substring of $s$ between locations $i$ and $j$, inclusively.

**Definition 2.** *A tree code $\mathcal{T}$ with parameters $(d, \alpha, \sigma_i, \sigma_o)$ corresponding to depth, distance, input alphabet size and output alphabet size is a $\sigma_i$-regular tree of depth $d$ with edges labeled with labels from the alphabet $\Sigma_o = \{1, \ldots, \sigma_o\}$. Denote $\Sigma_i = \{1, \ldots, \sigma_i\}$. The tree defines a natural mapping $\mathcal{T} : \Sigma_i^{\leq d} \to \Sigma_o^{\leq d}$. We require that for each three words $w, w_1, w_2$ over $\Sigma_i$, such that $|w_1| = |w_2|$, $|w| + |w_1| \leq d$, and $w_1[1] \neq w_2[1]$, we have*

$$H(\mathcal{T}(w \circ w_1), \mathcal{T}(w \circ w_2)) \geq \alpha \cdot |w_1|,$$

*where $\circ$ is the concatenation operator.*

In other words, the tree code is an encoding scheme where the $k$-th symbol of the encoding depends only on the first $k$ symbols of the source, and where the encodings of two different words diverge at a linear rate from the first location of disagreement. Mimicking coding theory terminology, a tree code is said to be *good* when the distance parameter $\alpha$ and the rate are both constant, i.e. $\alpha = \Omega(1)$ and $\sigma_o = \sigma_i^{O(1)}$. Good tree codes have been shown to exist in [Sch96] via a probabilistic argument. Unfortunately, to date, no explicit constructions of good tree codes are known.

When one deals with uniformly random noise, rather than with adversarial noise, coding for interactive communication does not require full tree codes, but rather only *local tree codes*. Local tree codes were introduced by Moitra [Moi11], but were also mentioned in an earlier unpublished memo by Schulman [Sch03] (under the name "weak tree codes"). In a local tree code one only requires divergent paths to diverge at a linear rate for a limited number $\ell$ of steps:

**Definition 3.** *A <u>local</u> tree code $\mathcal{T}$ with parameters $(d, \alpha, \ell, \sigma_i, \sigma_o)$ corresponding to depth, distance, locality, input alphabet size and output alphabet size is a $\sigma_i$-regular tree of depth $d$ with edges labeled with labels from the alphabet $\Sigma_o = \{1, \ldots, \sigma_o\}$. Denote $\Sigma_i = \{1, \ldots, \sigma_i\}$. The tree defines a natural mapping $\mathcal{T} : \Sigma_i^{\leq d} \to \Sigma_o^{\leq d}$. We require that for each three words $w, w_1, w_2$ over $\Sigma_i$, such that <u>$|w_1| = |w_2| \leq \ell$</u>, $|w| + |w_1| \leq d$, and $w_1[1] \neq w_2[1]$, we have*

$$H(\mathcal{T}(w \circ w_1), \mathcal{T}(w \circ w_2)) \geq \alpha \cdot |w_1|,$$

*where $\circ$ is the concatenation operator.*

Local tree codes are much easier to construct efficiently. As noted by Schulman in his note [Sch03], knowing a tree code of depth $\ell$ is enough to construct a local tree code. We will repeat this argument below for completeness. Moitra [Moi11] gives another way of constructing local tree codes with locality $\ell = \Theta(\log n)$ in polynomial time.

### 2.3 Efficient encoding and decoding of tree codes

For the main application of the codes, namely for coding for interactive communication, it is not enough to just construct the code $\mathcal{T}$. One also needs to be able to perform the encoding and decoding operations explicitly. While derandomization tools may allow us one day to get rid of the randomness in the tree code construction, it is not clear that they would yield an efficient encoding, and especially an efficient decoding algorithm. Looking for a "best match" path in a tree of depth $n$ may generally take time exponential in $n$. Our goal in this paper is to produce efficient encoding and decoding algorithms, thus we need to define efficient decoding that is "good enough" for the applications.

**Definition 4.**

- **Efficient encoding:** *A tree code (either local or general) $\mathcal{T}$ of depth $d$ and output alphabet size $\sigma_o$ is encodable in time $t(d, \sigma_o)$ if each symbol of the output can be computed in $t(d, \sigma_o)$ time.*

- **Efficient decoding – general tree codes:** *A tree code $\mathcal{T}$ with parameters $(d, \alpha, \sigma_i, \sigma_o)$ is decodable in time $t(d, \sigma_o)$ if there is an algorithm $D$ that runs in time $t(d, \sigma_o)$ that for each $k \leq d$ and for each string $s \in \Sigma_o^k$, outputs a string $D(s) \in \Sigma_i^k$ such that for each $s' \in \Sigma_i^k$, if $j$ is the first location of disagreement between $s'$ and $D(s)$, then*

$$H(s[j..k], \mathcal{T}(s')[j..k]) \geq \frac{\alpha}{2} \cdot (k - j + 1).$$

- **Efficient decoding – local tree codes:** *A local tree code $\mathcal{T}$ with parameters $(d, \alpha, \ell, \sigma_i, \sigma_o)$ is decodable in time $t(d, \ell, \sigma_o)$ if there is an algorithm $D$ that runs in time $t(d, \ell, \sigma_o)$ that for each $r \leq \ell$ and $k \leq d - r$ and for each pair of strings $s_1 \in \Sigma_i^k$, $s_2 \in \Sigma_o^r$ outputs a string $D(s_1, s_2) \in \Sigma_i^r$ such that for each $s' \in \Sigma_i^r$, if $j$ is the first location of disagreement between $s_2$ and $D(s)$, then*

$$H(s_2[j..r], \mathcal{T}(s_1 \circ s')[k + j..k + r]) \geq \frac{\alpha}{2} \cdot (r - j + 1).$$

The first two parts of Definition 4 are fairly straightforward and closely follow standard definitions of encoding and decoding. The tree code guarantee is that "a mistake causes the encoded words to diverge in Hamming distance". Thus the decoding guarantee is that "if we failed to decode $s'$ correctly from some location $j$, it is because there is a big Hamming distance starting from location $j$ between the encoding of $s'$ and the received word." The third part of the definition states that we can decode the last $r \leq \ell$ symbols with the same guarantees as with ordinary tree codes, *provided* that we are given all preceding symbols correctly.

Definition 4 is sufficient for applications in error-correction of interactive communications. Ideally, we would like to be able to decode tree codes in time polynomial in $d$ (assuming $\sigma_o$ is constant). A brute force decoding would take time exponential in $d$ for general tree codes, and exponential in $\ell$ for local tree codes. In the application of local tree codes to error-correcting for interactive communication that is affected by random noise one needs $\ell$ to be $\approx \log d$. Thus even the brute-force decoding scheme is sufficient if one is interested in an algorithm that runs in time polynomial in $d$. We would like our algorithm to run in subpolynomial time $d^{o(1)}$ per symbol. To achieve this we need a local tree code that can be decoded faster than brute force, i.e. in subexponential time. Note that a local tree code that is decodable in time $2^{O(\sqrt{\ell})}$ already suffices, as it would translate into a communication scheme that requires $2^{\sqrt{\log d}} = d^{o(1)}$ computation per round.

# 3 Main results: combining multiple tree codes

In this section we will introduce an operation we call *tree code product* that uses two tree codes of depths $d_1$ and $d_2$ to produce a tree code of depth roughly $d_1 \times d_2$ without decreasing the rate too much. The newly obtained tree code will be decodable in time similar to the two original tree codes. We will then use this operation to obtain good tree codes that can be constructed and decoded in subexponential time. We will use these, in turn, to obtain local tree codes that can be deterministically constructed and decoded in subpolynomial time.

### 3.1 From a small tree code to a local tree code

As a first step, we will need a generic tool for converting a tree code of depth $d$ into a local tree code of depth $D \gg d$ with $\ell \approx d$. In addition to the obvious benefit in the construction of local tree codes, we will also need this tool to perform the tree code product later in the section. To our knowledge, this construction was first outlined in an unpublished note by Schulman [Sch03]. We repeat essentially the same proof here both for completeness and to get all the parameters set up for later use.

**Lemma 5.** *Let $\mathcal{T}$ be a tree code with parameters $(d, \alpha, \sigma_i, \sigma_o)$. Then for any depth parameter $D$ and for any $\varepsilon$, we can construct a depth $D$ local tree code $\mathcal{T}'$ with parameters $(D, \alpha, d\cdot(1-\varepsilon), \sigma_i, \sigma'_o)$, where $\sigma'_o = \sigma_o^{O_\varepsilon(1)}$. Moreover, up to a factor of* polylog $D$*, the encoding and decoding times for $\mathcal{T}'$ are the same as for $\mathcal{T}$.*

*Proof.* The proof proceeds by overlapping $k = 1 + \lceil 1/\varepsilon \rceil = O_\varepsilon(1)$ copies of $\mathcal{T}$. Pick $k$ numbers $t_1 = 1 < t_2 < \ldots < t_k \leq d$ such that for each $i$, $t_{i+1} - t_i \leq 1 + \varepsilon d$, and $d + 1 - t_k \leq 1 + \varepsilon d$. Let $\mathcal{T}'_i$ be a (periodic) tree code that uses $\mathcal{T}$ to encode each block of $d$ inputs starting from locations that are $t_i \mod d$. Thus each $\mathcal{T}'_i$ outputs a label in $\Sigma_o$. Let $\mathcal{T}'$ be the concatenation of $\mathcal{T}'_1, \ldots, \mathcal{T}'_k$. Then $\mathcal{T}'$ is a tree code with alphabet $\Sigma_o^k$ of size $\sigma_o^k$.

To see that $\mathcal{T}'$ is a local tree code observe that if $w, w_1, w_2$ are three strings over $\Sigma_i$ such that $|w_1| = |w_2| \leq d \cdot (1 - \varepsilon)$, and let $j := |w| + 1$. Then there is an $i$ and a location $q \leq j$ with $q = t_i \mod d$ such that $j - q \leq d\varepsilon$. Then by the tree code property of $\mathcal{T}$, the encodings of $w \circ w_1$ and $w \circ w_2$ will disagree in the $i$-th coordinate in at least an $\alpha$-fraction of the locations from $j$ to $j + |w_1| - 1$.

Finally, it is evident from the construction that encoding and decoding of $\mathcal{T}'$ reduces to at most $k$ operations on the tree $\mathcal{T}$ and simple arithmetic with numbers of magnitude $\leq D$.

Let us also note that the construction would work and produce a $(D, \alpha, d/2, \sigma_i, \sigma'_o)$-local codes by overlapping just two copies of $\mathcal{T}$, thus yielding $\sigma'_o = \sigma_o^2$. □

### 3.2 Tree code product

We can now formulate our main theorem on combining two tree codes to obtain a third, larger, tree code. The operation resembles a product operation.

**Theorem 6.** *Let $\mathcal{T}_I$ and $\mathcal{T}_O$ be two tree codes of depth, distance and alphabet size parameters $(d_1, \alpha_1, \sigma_i, \sigma_{o1})$ and $(d_2, \alpha_2, \sigma_i, \sigma_{o2})$, respectively. Then we can construct a "product tree code" $\mathcal{T}_P := \mathcal{T}_I \otimes \mathcal{T}_O$ with parameters $(d, \alpha, \sigma_i, \sigma_o)$ where $\alpha = \min(\alpha_1, \alpha_2/10)$, $d = d_1 \times d_2/4$, and $\sigma_o = (\sigma_{o1}\sigma_{o2})^{O(1)}$.*

*Moreover, the encoding and decoding times for $\mathcal{T}_P$ are the same as for $\mathcal{T}_I$ and $\mathcal{T}_O$, except there is a multiplicative overhead polynomial in $d_1 + d_2$.*

*Proof Idea:* We use a tiling by the "internal" tree code $\mathcal{T}_I$ to obtain a local tree code with $\ell = d_1/2$. This takes care of divergent paths that diverge for at most $d_1/2$ positions. To take care of longer paths, we use the outer tree code $\mathcal{T}_O$. Consider a code where we use $T = d_1/4$ interleaved copies of $\mathcal{T}_O$. The first copy encodes positions $1, 1+T, \ldots, 1+(d_2-1)\cdot T$, the second copy encodes positions $2, 2+T, \ldots, 2+(d_2-1)\cdot T$, etc. This is, of course, not a good tree code, since one disagreement in the source will only lead to a $\Theta\left(\frac{1}{d_1}\right)$-fraction of disagreements in the encoding. However, these disagreements will be well-spread among the blocks of length $d_1/4$: a constant fraction of the blocks

will have at least on location of disagreement. To amplify this to a constant distance we wrap each block with an ordinary good error-correcting code. Divergent paths of length $> d_1/2$ will contain one or more blocks of length $d_1/4$. A constant fraction of these blocks will have a constant rate Hamming distance from each other, yielding the tree code property for longer divergent paths. □

*Proof.* Assume without loss of generality that $d_1/4$ is an integer. We will make use of a good code $C : \Sigma_{o2}^{d_1/4} \to \Sigma_3^{d_1/4}$ from Lemma 1 with $\varepsilon = 1/3$, so that the distance between any two codewords is at least $d_1/6$. Furthermore, we know that $\sigma_3 := |\Sigma_3| = \sigma_{o2}^{O(1)}$.

By Lemma 5 we can use $\mathcal{T}_I$ to construct a local tree code $\mathcal{T}'_I$ of depth $d = d_1 \times d_2/4$ with parameters $(d, \alpha_1, d_1/2, \sigma_i, \sigma'_{o1})$, where $\sigma'_{o1} = \sigma_{o1}^{O(1)}$. The tree code $\mathcal{T}_P$ will be the concatenation of $\mathcal{T}'_I$ with another code $\mathcal{T}'_O$. The local tree code $\mathcal{T}'_I$ "takes care" of divergent paths that are of length $d_1/2$ or shorter. The code $\mathcal{T}'_O$, which will also be a tree code, "takes care" of divergent paths that are of length $\Omega(d_1)$. We note that the code $\mathcal{T}'_O$ will not have distance properties of a good tree code on scales below $d_1$.

Before constructing the code $\mathcal{T}'_O : \Sigma_i^{\leq d} \to \Sigma_3^{\leq d}$, we construct an intermediate code $\mathcal{T}''_O : \Sigma_i^{\leq d} \to \Sigma_{o2}^{\leq d}$. For a string $w \in \Sigma_i^*$, $w = s_1, s_2, \ldots, s_n$ and an integer $k$, define

$$w \bmod k := s_{n-\lfloor \frac{n-1}{k} \rfloor \cdot k}, s_{n-\lfloor \frac{n-1}{k} \rfloor \cdot k+k}, \ldots, s_{n-k}, s_n.$$

In other words, $w \bmod k$ is obtained from $w$ by starting from the last element and going backwards, taking every $k$-th element. We define

$$\mathcal{T}''_O(w) := \mathcal{T}_O(w \bmod d_1/4).$$

We now define $\mathcal{T}'_O$ from $\mathcal{T}''_O$. $\mathcal{T}'_O$ is defined in blocks of size $d_1/4$. Let the output of $\mathcal{T}''_O$ be given by blocks $B_1, B_2, \ldots, B_{d_2}$ of length $d_1/4$ each. Then the output of $\mathcal{T}'_O$ is

$$B_0, C(B_1), C_B(2), \ldots, C(B_{d_2-1}).$$

Here $B_0$ is an arbitrary string in $\Sigma_3^{d/4}$, and $C$ is the good code from Lemma 1 that we have assumed. Note that $\mathcal{T}'_O$ thus defined is a tree code in that the $i$-th symbol in the output only depends on the first $i$ symbols of the input.

As noted earlier, we define $\mathcal{T}_P$ to be the concatenation of $\mathcal{T}'_I$ and $\mathcal{T}'_O$. The alphabet size $\sigma_o$ of the resulting code is $\sigma_o = \sigma'_{o1} \cdot \sigma_3 = (\sigma_{o1}\sigma_{o2})^{O(1)}$. The depth of the code is $d = d_1 \times d_2/4$. It is also easy to see that the encoding process is efficient. It remains to see that the distance of the code is indeed $\alpha = \min(\alpha_1, \alpha_2/10)$, and that decoding can be performed efficiently.

**The distance of $\mathcal{T}_P$.** Let $w, w_1, w_2$ be three strings over $\Sigma_i$ such that $|w_1| = |w_2|$, $w_1[1] \neq w_2[1]$, and $|w| + |w_1| \leq d$. We need to show that

$$H(\mathcal{T}_P(w \circ w_1), \mathcal{T}_P(w \circ w_2)) \geq \alpha \cdot |w_1|.$$

There are two cases to consider.
*Case 1:* $|w_1| \leq d_1/2$. In this case,

$$H(\mathcal{T}_P(w \circ w_1), \mathcal{T}_P(w \circ w_2)) \geq H(\mathcal{T}'_I(w \circ w_1), \mathcal{T}'_I(w \circ w_2)) \geq \alpha_1 \cdot |w_1| \geq \alpha \cdot |w_1|.$$

*Case 2:* $|w_1| > d_1/2$. In this case let $B_i$ be the length-$(d/4)$ block that contains the first location of $w_1$. Let $B_{i+1}, \ldots, B_{i+j}$ be the blocks that are entirely contained within the locations covered by

$w_1$. Note that since $|w_1| > d_1/2$, $j \cdot d_1/4 \geq |w_1|/3$, i.e. $j \geq \frac{4}{3} \cdot \frac{|w_1|}{d_1}$. Let $k$ be the location of the first symbol of $w_1$ within $B_i$. Since the $k$-th symbol in each block of the encoding of $w \circ w_1$ by $\mathcal{T}''_O$ is encoded using the tree code $\mathcal{T}_O$, at least $\alpha_2 \cdot j$ of the blocks $B_i, \ldots, B_{i+j-1}$ contain a disagreement location under the encoding $\mathcal{T}''_O$. Finally, since $C$ is an error-correcting code of distance $d_1/6$ on each block, the distance between the blocks $B_{i+1}, \ldots, B_i$ under the encoding $\mathcal{T}'_O$ is at least

$$\alpha_2 \cdot j \cdot \frac{d_1}{6} \geq \alpha_2 \cdot \frac{4}{3} \cdot \frac{|w_1|}{6} = \frac{2\alpha_2}{9} \cdot |w_1| > \alpha \cdot |w_1|.$$

**Efficient decoding of $\mathcal{T}_P$.** The decoding algorithm also works on two different scales. Given an input string $s \in \Sigma_o^k$ with $k \leq d$ that needs to be decoded, it uses the decoder for $\mathcal{T}'_I$ to decode the last $d_1/2$ symbols. Note that the local tree code guarantee is not sufficient to be able to decode $\mathcal{T}'_I$ since we may not have a decoded prefix. However, the structure of the construction of $\mathcal{T}'_I$ implies that there is a copy of $\mathcal{T}_I$ that covers the last $d_1/2$ symbols of $s$, and we use this copy to decode them. We thus obtain a decoding $D(s)[(k - d_1/4 + 1)..k]$.

To decode symbols in locations $[1..k - d_1/2]$, we use $\mathcal{T}'_O$. Let $B_1, \ldots, B_j$ be the length-$d_1/4$ blocks that are contained entirely within the locations of $s$. Thus $j \cdot (d_1/4) \leq k < (j+1) \cdot (d_1/4)$. Let $s_1, \ldots, s_{j-1} \in \Sigma_{o2}^{d_1/4}$ be strings obtained by decoding the blocks $B_2, B_3, \ldots, B_j$ in $s$ using the decoder for the code $C$. Finally, if we are interested in the $q$-th location in block $B_b$, where $1 \leq b \leq j-1$, we first construct the string $B_1[q], B_2[q], \ldots, B_{j-1}[q]$, and then use the decoder for $\mathcal{T}_O$ to decode it and obtain the decoding for $D(s)[(b-1) \cdot d_1/4 + q] \in \Sigma_i$. This allows us to decode $D(s)[1..(j-1) \cdot (d_1/4)]$.

We have $(j-1) \cdot (d_1/4) > k - d_1/2$, and thus at least one of the two methods allows us to decode all symbols in $D(s) \in \Sigma_i^k$. If a location $i$ happens to be decoded by both, we use the first method, i.e. the local decoding using $\mathcal{T}_I$ to determine $D(s)[i]$. It is easy to see that the overhead is polynomial in depth. The only operations performed are decoding using $\mathcal{T}_I$ and $\mathcal{T}_O$ and decoding with respect to the good code $C$.

The performance analysis of the decoding algorithm is very similar to the distance analysis. Let $s' \in \Sigma_i^k$, and let $t$ be the first location of disagreement between $s'$ and $D(s)$. We need to show that

$$H(s[t..k], \mathcal{T}_P(s')[t..k]) \geq \frac{\alpha}{2}(k - t + 1).$$

Once again, there are two cases to consider.
*Case 1:* $t > k - d_1/2$. In this case, by the decoding properties of $\mathcal{T}_I$, we must have

$$H(s[t..k], \mathcal{T}_P(s')[t..k]) \geq \frac{\alpha_1}{2}(k - t + 1) \geq \frac{\alpha}{2}(k - t + 1).$$

*Case 2:* $t \leq k - d_1/2$. Let $B_i$ be the block containing the location $t$. By the tree code decoding property of $\mathcal{T}_O$, at least $\frac{\alpha_2}{2} \cdot (j - i)$ of the strings $s_i, \ldots, s_{j-1}$ have disagreements with the corresponding strings in the encoding of $s'$. By the properties of the code $C$, if there is a disagreement between an $s_q$ and the corresponding string in the encoding of $s'$, it means that the number of disagreements between $s$ and $\mathcal{T}_P(s')[t..k]$ in the locations corresponding to the next block $B_{q+1}$ is at least $\frac{d_1}{12}$. Thus we have

$$H(s[t..k], \mathcal{T}_P(s')[t..k]) \geq \frac{\alpha_2}{2} \cdot (j - i) \cdot \frac{d_1}{12} \geq \frac{\alpha_2}{2} \cdot \frac{k - t + 1}{3d_1/4} \cdot \frac{d_1}{12} = \frac{\alpha_2}{18} \cdot (k - t + 1) > \frac{\alpha}{2} \cdot (k - t + 1).$$

□

Note that the construction is not associative, i.e. $(\mathcal{T}_1 \otimes \mathcal{T}_2) \otimes \mathcal{T}_3$ and $\mathcal{T}_1 \otimes (\mathcal{T}_2 \otimes \mathcal{T}_3)$ are different codes. Another interesting feature is that the the distance parameter $\alpha$ only decreases with respect to the second coordinate. This means that we can use a single code $\mathcal{T}$ with parameters $(d, \alpha, \sigma_i, \sigma_o)$, apply its product with itself on the right $k$ times to obtain a code $\mathcal{T}_k = ((\ldots(\mathcal{T} \otimes \mathcal{T}) \otimes \mathcal{T}) \otimes \ldots) \otimes \mathcal{T}$ of depth $\sim d^k$ and distance $\geq \alpha/10$.

# 4 Applications: interactive computation with random noise

## 4.1 Subexponential-time tree codes

Theorem 6 allows us to give the first deterministic construction of good tree codes in time subexponential in their depth. Moreover, the encoding/decoding operations will also be subexponential.

**Theorem 7.** *Let $d$ and $\varepsilon > 0$ be any parameters, and let $\sigma_i$ be the constant size of the input alphabet. Then there is a constant $\alpha$ independent of $d$ and $\varepsilon$ and a constant $\sigma_o = \sigma_o(\varepsilon, \sigma_i)$ such that we can construct a tree code $\mathcal{T} = \mathcal{T}(d, \varepsilon)$ with parameters $(d, \alpha, \sigma_i, \sigma_o)$. Moreover, the construction, as well as the encoding and decoding operations will run in time polynomial in $2^{d^\varepsilon}$.*

*Proof.* We know [Sch96] that there exists a tree code with parameters $(4d^\varepsilon, \alpha', \sigma_i, \sigma_o')$. Where $\sigma_o'$ is a constant, and $\alpha'$ is a constant we can make arbitrarily close to 1. Using brute force we can find such a code $\mathcal{T}_1$ in time $2^{O(d^\varepsilon)}$. Moreover, $\mathcal{T}_1$ will be encodable and decodable in time $2^{O(d^\varepsilon)}$.

Set $k := \lceil 1/\varepsilon \rceil$, and let $\mathcal{T}_k = ((\ldots(\mathcal{T}_1 \otimes \mathcal{T}_1) \otimes \mathcal{T}_1) \otimes \ldots) \otimes \mathcal{T}_1$. Then by $k$ repeated applications of Theorem 6, we see that $\mathcal{T}_k$ is a $(d', \alpha'/10, \sigma_i, \sigma_o)$-tree code, where $d' > d$ and $\sigma_o$ is a constant. Moreover, $\mathcal{T}_k$ is encodable and decodable in time $2^{O(d^\varepsilon)}$. □

## 4.2 Interactive computation with random noise

We can now use Theorem 7 together with Lemma 5 to obtain a local tree code of depth $n$ with $\ell \gg \log n$ and subpolynomial construction, encoding and decoding times.

**Theorem 8.** *For any parameters $c > 1$, any depth $n$ and any input alphabet size $\sigma_i$, we can construct a local tree code with parameters $(n, \alpha, \ell, \sigma_i, \sigma_o)$, where $\alpha$ is a constant independent of $c$, $n$, and $\sigma_i$, $\sigma_o = O_{c,\sigma_i}(1)$, and $\ell = \Omega((\log n)^c)$. Moreover, the construction, as well as each encoding/decoding operation, can be performed in subpolynomial time of $t(n, \ell, \sigma_o) = 2^{O((\log n)^{1/c})}$.*

*Proof.* We apply Theorem 7 with $d = (\log n)^c$, $\varepsilon = 1/c^2$, and $\sigma_i = \sigma_i$ to obtain a tree code $\mathcal{T}'$ with parameters $((\log n)^c, \alpha, \sigma_i, \sigma_o')$, where $\alpha$ is a universal constant, and $\sigma_o'$ is a constant that depends on $\sigma_i$ and $c$. The code $\mathcal{T}'$ is constructible, encodable, and decodable in time

$$t(n) = 2^{O(d^\varepsilon)} = 2^{O((\log n)^{1/c})}.$$

Next we apply Lemma 5 to construct the local tree code $\mathcal{T}$ from $\mathcal{T}'$. The local tree code $\mathcal{T}$ will have the desired parameters. □

If we are using a tree code to send messages down a channel that is affected by i.i.d. random noise of a constant rate, as opposed to an adversarial noise, then the probability that a stretch of $\ell$ positions contains a disproportionately high number of errors decays exponentially in $\ell$. This

**The transmitting player:** encodes her messages using $\mathcal{T}$.
**The receiving player:** maintains the currently decoded messages.
After step $i$, maintain a string $w_i$ of length $l_i := \max(i - \ell + 1, 0)$.
On step $i + 1$:
• The receiver decodes the symbols in locations $[l_i + 1..i + 1]$ using the local tree code decoder and the string $w_i$, thus obtaining a string $s[l_i + 1..i + 1]$.
• If $l_{i+1} > 0$, the receiver updates $w_{i+1} := w_i \circ w[l_i + 1]$.

Figure 1: The protocol for communication over a noisy channel using a local tree code

means that if $\ell \gg \log n$, the probability of having a stretch of errors that is longer than $\ell$ anywhere during the communication becomes small.

More formally, consider the following algorithm for simulating a tree code with a local tree code $\mathcal{T}$ where the noise is random. Suppose that $\mathcal{T}$ has parameters $(n, \alpha, \ell, \sigma_i, \sigma_o)$, and assume that the noise rate of the channel is $< \alpha/4$. The protocol is outlines on Figure 1. At each step the receiver maintains a string $w_i$. These are the "verified" decoded symbols – and they include all the symbols but the last $\ell$. The receiver uses the local tree code decoder to recover the remaining symbols. He then adds the earliest (and thus the most certain) recovered symbol to $w_i$ before proceeding to the next iteration. The computational cost per round is the encoding/decoding cost of $\mathcal{T}$.

In all the applications of tree codes, particularly in [BR10], the encoding/decoding scheme on Figure 1 is a good replacement for the actual tree code, *provided* that all the decodings $w_i$ are perfectly correct. By the local tree code property, this is indeed the case as long as there is no continuous stretch of $\ell$ locations which contains at least $\alpha/2$ transmission errors. Since we assume that the error rate is $< \alpha/4$, the probability of having a continuous stretch of $\ell$ locations with at least $\alpha/2$ errors is bounded by

$$n \cdot \left( \left( \frac{3}{4} \right)^{\alpha/4} \right)^{\ell} = n \cdot 2^{-\Omega(\ell)},$$

since $\alpha$ is a constant. The scheme on Figure 1 will also work when the errors on the channel are adversarial, as long as no stretch of $\ell$ adjacent locations has an error rate exceeding $\alpha/4$.

If our channel is affected by random errors, we can reduce the error rate to any constant, in particular below $\alpha/4$, while reducing the code rate by a constant. This can be achieved, for example, by retransmitting each symbol a constant number of times. By applying the parameters from Theorem 8, and combining the result with the protocol from [BR10] we obtain:

**Corollary 9.** *For every noise rate $\beta < 1/2$, and any parameter $c > 1$, there is a deterministic transformation of communication protocols $C_{\beta,c}(\pi)$ and a constant $\sigma_o$, such that for every binary protocol $\pi$, $C_{\beta,c}$ is a protocol over $\Sigma_o$, $|C_{\beta,c}(\pi)| = O_{\beta,c}(|\pi|)$, and for all inputs $x, y$, both parties can determine $\pi(x, y)$ correctly with probability $> 1 - 2^{\Omega((\log n)^c)}$, by running $C_{\beta,c}(\pi)$ if the channel is affected by a random noise of rate $\beta$. Moreover, as the encoding/decoding of each symbol takes time $2^{O((\log n)^{1/c})}$, the computational resources required to execute $C_{\beta,c}(\pi)$ are bounded by $n \cdot 2^{O((\log n)^{1/c})}$.*

It is also easy to see that the alphabet $\Sigma_o$ can be replaced with the binary alphabet, with only a constant overhead in the rate. This can be done by replacing each symbol with its bi-

nary representation, and repeating this representation a constant number of times, thus implying Theorem A.

### Acknowledgments

I would like to thank Swastik Kopparty, Anup Rao, and David Zuckerman for inspiring discussions and advice.

## References

[BR10] M. Braveman and A. Rao. Towards coding for maximum errors in interactive communication. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2010.

[GS11] R. Gelles and A. Sahai. Potent Tree Codes and their applications: Coding for Interactive Communication, revisited. *Arxiv preprint arXiv:1104.0739*, 2011.

[Jus72] J. Justesen. Class of constructive asymptotically good algebraic codes. *Information Theory, IEEE Transactions on*, 18(5):652–656, 1972.

[Moi11] A. Moitra. Efficiently coding for interactive communication. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2011.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes*. North–Holland, New York, 1977.

[RVW02] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *The Annals of Mathematics*, 155(1):157–187, 2002.

[Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996.

[Sch03] L. Schulman. A postcript to "Coding for Interactive Communication", 2003. http://www.cs.caltech.edu/~schulman/Papers/intercodingpostscript.txt.

[SS96] M. Sipser and D.A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[Sud01] M. Sudan. Algorithmic introduction to coding theory – course notes, 2001. http://people.csail.mit.edu/madhu/FT01/course.html.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de