Electronic Colloquium on Computational Complexity, Report No. 167 (2025)

CHS-alike $1/O(\log\log n)$-rate tree codes from elementary binary shifts

Tal Yankovitz*

Abstract

In a breakthrough in the long-going attempt to construct good explicit tree codes, Cohen, Haeupler and Schulman (CHS) (STOC 2018) constructed constant-distance tree codes with rate $1/O(\log\log n)$. In their construction a large-alphabet tree code is used as a core element - and they were able to utilize polynomials over the Newton-basis to construct one, relying on a sparsity lemma for such polynomials (or alternatively, by using a construction of Pudlák). Here we simplify the proof of their result by replacing this element with an alternative construction, which consists of shifting input binary vectors and taking their binary sum. As an artifact of that, we note that the obtained rate-$1/O(\log\log n)$ tree code is linear (a property which to our knowledge was not previously known).

1 Introduction

Definition 1.1. *A tree code of length n, rate $1/r$ and distance δ is an online function*

$$\mathcal{TC} : \{0,1\}^n \to (\{0,1\}^r)^n$$

such that for every distinct $(x_1,\ldots,x_n),(x'_1,\ldots,x'_n) \in \{0,1\}^n$ the relative-Hamming distance, of every continuous same-length two substrings of $\mathcal{TC}(x_1,\ldots,x_n),\mathcal{TC}(x'_1,\ldots,x'_n)$ which start from the first index where x and x' differ, is at least δ. By online *we mean that for every $t \in [n]$ the function $\mathcal{TC}(x_1,\ldots,x_n)_t$, denoted $\mathcal{TC}^t$, depends only on $x_1,\ldots,x_t$. We say that the tree code is* explicit *if it can be computed in time* $\mathrm{poly}(n)$*. We say that $\mathcal{TC}$ is* linear *if for every $t \in [n]$ $\mathcal{TC}^t$ is an ($\mathbb{F}_2$-)linear map.*

*UT Austin. talyanko@utexas.edu. Supported by NSF Grant CCF-2312573.

ISSN 1433-8092

Tree codes were introduced in the seminal work of Schulman [Sch93] who showed that they yield interactive-schemes for achieving error-resilient two-party protocols, and the problem of constructing them has attracted significant attention [Sch94, Bra12, MS14, Pud16b, GHK+16, CHS18, NW20, BH20, BYCN21, Pud22, BYCY22, MRR25, CSS25]. Schulman [Sch93, Sch96] raised the question of whether explicit constructions of tree codes having constant distance and constant rate are possible, which has been open since. In an important breakthrough, Cohen, Haeupler and Schulman were able to show the following.

Theorem 1.2 ([CHS18])**.** *For every $n \in \mathbb{N}$ there is an explicit tree code*

$$\mathcal{TC} : \{0,1\}^n \to (\{0,1\}^r)^n$$

with $r = O(\log \log n)$ and distance $\delta = \Omega(1)$.[1]

That is, they obtained explicit constant-distance tree codes with rate $1/O(\log \log n)$. This consisted an exponential improvement (in the rate) over what was previously known [Sch94].

1.1 [CHS18] construction details and our contribution

As no constant-distance $1/o(\log \log n)$-rate explicit tree code construction has been discovered following [CHS18] we see importance in simplifying or finding alternatives to the elements upon which the present state-of-the-art can be established. Let us briefly overview of the CHS construction before describing our suggested alternative.

1.1.1 On the CHS construction

In their proof, [CHS18] rely on a construction of a tree code over a large input-alphabet, which they show can be constructed by encoding a given message as a polynomial $p \in \mathbb{R}[x]$ over the Newton-basis, which is $\{\binom{x}{i}\}_{i \in \mathbb{N} \cup \{0\}}$ rather than the standard basis $\{x^i\}_{i \in \mathbb{N} \cup \{0\}}$ usually used in polynomial-based error correcting codes. [CHS18] show that in the case of polynomials expressed in the Newton-basis *over the reals*, the Gessel-Viennot Lemma can be employed in order to bound the number of integral roots in terms of the number of non-zero coefficients (i.e., the sparsity). This establishes that their mentioned large input-alphabet tree code has constant distance.

The second part of their proof consists of showing that such a large-alphabet tree code can be used in order to obtain tree codes over a binary-input alphabet (as in Definition 1.1). Besides the large-alphabet tree code, the building blocks[2] used in this part of

[1] In fact [CHS18] showed that this holds for any constant $\delta < 1$.

[2] See Section 6 in [CHS18].

their proof are (plain) error codes and tree codes for a very small message length which can be found by brute-force. The encoding then boils down to outputting *in parallel* (bits of) the large-alphabet tree codes concatenated with an error correcting code, applied to different length substrings of the message. Their idea behind this reduction is that is that each of the different parallel tree codes assures distance after a different *lag*, and conjointly they imply tree code distance overall.

It is important to note that [CHS18] also observe that by using a construction of Pudlák [Pud16a] one can obtain a large-alphabet tree code which can be used instead of the one based on the Newton-basis. The [Pud16a] construction goes by devising[3] a lower triangular matrix such that every square submatrix, with a diagonal entirely in the lower triangle, is nonsingular - from which a construction of tree code readily follows. [Pud16a]'s proof that his matrix is such relies on the underlying field being large enough and on an inductive argument which uses the Leibniz formula for determinant.

1.1.2 Our contribution

Our contribution is to replace the mentioned large input-alphabet with an alternative one - which we fully describe in the (short) Section 2. We briefly overview the idea of this simple alternative and then discuss two artifacts.

We suggest a construction which works as follows. The input is some $(x_1, \ldots, x_m) \in (\{0,1\}^n)^m$, that is, a sequence of m binary vectors of length n (thus the "large alphabet" tree code has input alphabet $\{0,1\}^n$), and we need to set an output y_t for each time $t \in [m]$. As [CHS18] note, it is enough to achieve $\{y_t\}_{t\in[m]}$ such that the number of zeroes among them (that is, $\bar{0}$) starting from the first index i where $x_i \neq \bar{0}$ – is bounded by the number non-zero x_i's. Then, the output at time t can be redefined to $y'_t = (x_t, y_t)$, and it would readily follow that this is a tree code with distance (larger than) $1/2$.

We need to define $y_1, \ldots, y_t$ as an online function, i.e., to have that y_t is dependent only on $x_1, \ldots, x_t$ – but for simplicity let us start by describing a construction which isn't online (and then say what should be adjusted).

For the (non-online) output y_1 at the first time $t = 1$, we arrange $x_1, \ldots, x_m$ as rows of a matrix, and output their sum (each entry summed modulo 2), which is a length-n binary vector. For the output y_2 at the second time $t = 2$, we leave the first row of the matrix as is, but shift the second row one place the left, the third row two places to the left, and so on – we add zeroes in places where there are now blank entries, and again sum up the all the rows and output the result, which is a binary vector of length $n + m$. At the next time, we again so shift each row (row i further shifts by $i - 1$ spots) and our

[3] See lemma 6.1 in [Pud16a].

output is of length $n+2m$. At the last point in time, y_m is of length $n+m^2$ (and we can see that if one sets $m=\sqrt{n}$, this is manageable in terms of rate).

Clearly, only shifting vectors and taking binary sums, each bit in the output consists a linear map of the bits of the message. We argue that the number of zeroes among $y_1,\ldots,y_m$ is bounded by the number of non-zeros among $x_1,\ldots,x_m$ (assuming they are not all zero). One way to see that is by induction on the number of non-zeroes among $x_1,\ldots,x_m$. Consider the first point t in time where the summation resulted in a zero vector. Further consider the left-most column of the shifted matrix at that time which has a non-zero at any row, and consider the smallest row index i on which it is non-zero. Since the result of the summation is the zero vector, there is at least one more row j which is non-zero on that column. Since $j>i$, at every time $t'>t$, row j would be shifted further to the left compared to row i (compared to what it was at time t), and thus row i can never again be relevant to making the sum of the left-most (shifted) non-zero column equal to 0. Thus, since x_i is "out of the game", there is one less non-zero row at play, and the induction hypothesis can be employed. A more exact formal account can be found in Section 2.

To make this construction online a simple modification is made: instead of summing all the rows at each time, at time $t=1$ we only set y_1 to be the sum the first 1 rows (that is, the first row), and the output y_2 is set to be the shifted sum of only the first 2 rows, and so on, the output y_t only sums up the shifted rows that correspond to $x_1,\ldots,x_t$. It is not hard to see that this does not change the correctness of the described argument (but we now only need to consider zeroes among $y_1,\ldots,y_m$ following the first time where x_t is non-zero[4]). From this, a length m, constant rate, distance $1/2$, tree code over input alphabet $\{0,1\}^n$ follows.

1.1.3 Remarks regarding linearity and decoding

To our knowledge an explicit linear tree code matching the parameters of [CHS18] has not been observed. This has been noted as a hurdle to some applications as noted, for example in [MRR25] (see there Remark 1.1). However by considering the binary-shifts large-alphabet tree code we can amend that.

Proposition 1.3. *For every $n\in\mathbb{N}$ there is an explicit* linear *tree code*

$$\mathcal{TC}:\{0,1\}^n\to(\{0,1\}^r)^n$$

with $r=O(\log\log n)$ and distance $\delta=\Omega(1)$.

[4] And the "left-most non-zero (shifted) column" should be defined only by rows of index at most t.

Indeed, in the [CHS18] framework the only non-linear operation in the encoding was that of the Newton-basis large-alphabet tree code[5]. Since in the binary-shifts case every bit of the output is a linear map of bits of the input, and [CHS18] proceeds by applying plain error correcting codes or very small tree codes – both of which can be assumed linear, the proposition follows.

Secondly, we address the question of efficient decoding. While no efficient algorithm is known that corrects a constant fraction of errors for the codes of Theorem 1.2, Narayanan and Weidner [NW19] gave a randomized Las Vegas algorithm running in expected polynomial time that corrects a $\tilde{\Omega}(\frac{1}{n^{1/4}})$-fraction of errors.

We do not know an efficient decoding algorithm for the binary-shifts large-alphabet tree code, however we note that one can (very simply) decode it from a polynomial numbers of bitwise errors. While this suffices in order to decode the final resulted binary tree code from some polynomial number of errors, as [NW19] observe that in the [CHS18] framework a decoding algorithm for the large-alphabet tree code implies a decoding algorithm for the final resulted code, there are other ways to get such a decoding guarantee.[6]

We shortly explain the very simple way to decode the binary-shifts tree code from a weak guarantee of polynomial amount of bitwise errors. Note that the outputs $y_1, \ldots, y_m \in \{0,1\}^{n+m^2}$ described in the previous part - in fact consist the parity bits of different *lines* in the grid in which $x_1, \ldots, x_m$ are arranged as rows of a matrix: each y_t contains the parity bits of the lines with slope t. Since the lines passing through each point are disjoint, if only a small fraction of bits in $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$ has been changed, for each bit within a certain x_i most of the lines passing through it will have both their parity bit, and the other bits on the line - both uncorrupted. Thus, each bit can be decoded by taking a majority vote between its different lines.

2 Reproof of Theorem 1.2 with a "binary shifts" alternative

We set a few preliminaries.

Preliminaries. $\mathbb{N} = \{1, 2, \ldots, \}$ is the set of natural numbers. For two vectors or strings $u, v \in \Sigma^n$ we use $\mathsf{Dist}(u, v)$ to denote their absolute Hamming distance, and $u \circ v$

[5] It is not linear because its output consists of truncated integers based on an input of smaller integers.

[6] One can construct a tree code with rate $1/O(\log \log n)$ that can decode from a higher number of errors for example by appending to the output of the constant-distance tree code, a low-distance tree code - say that of [GHK+16], which can be seen to admit efficient decoding within its lower-distance.

to denote their concatenation. Given $u \in \Sigma^n$ and $i \leq j$ we use $u_{i,\ldots,j}$ to denote the substring $(u_i, \ldots, u_j)$.

We start by noting that Theorem 1.2 is proven in [CHS18] by invoking their important following proposition.

Proposition 2.1 ([CHS18]). *Assume that for some constants $c > 0$ and $0 < \delta < 1$, for every $\ell \in \mathbb{N}$ there exists an explicit tree code*

$$\mathcal{TC}_\ell : (\{0,1\}^{\ell^c})^\ell \to (\{0,1\}^{O(\ell^c)})^\ell.$$

with distance δ. That is, that as in Definition 1.1, it is online, and the distance property described in Definition 1.1 is met for every distinct $(x_1, \ldots, x_\ell), (x'_1, \ldots, x'_\ell) \in (\{0,1\}^{\ell^c})^\ell$ considering $\mathcal{TC}_\ell(x_1, \ldots, x_\ell)$ and $\mathcal{TC}_\ell(x'_1, \ldots, x'_\ell)$[7]. Then, for every $n \in \mathbb{N}$ there exists an explicit tree code

$$\mathcal{TC} : \{0,1\}^n \to (\{0,1\}^{O(\log\log n)})^n$$

with distance $\Omega(1)$.

Different than [CHS18] who use a large-alphabet tree-code arising from a sparsity lemma for polynomials over the Newton basis, which they prove by making use of the Gessel-Viennot Lemma, we will instantiate the above proposition with an alternative simpler construction yielding the following.

Lemma 2.2. *For every $\ell \in \mathbb{N}$ there exists an explicit tree code*

$$\mathcal{TC}_\ell : (\{0,1\}^{\ell^2})^\ell \to (\{0,1\}^{2\ell^2+(\ell-1)^2})^\ell$$

with distance (larger than) $1/2$.

We note that Proposition 2.1 with Lemma 2.2 immediately implies Theorem 1.2. Before we present the construction yielding Lemma 2.2 we turn to define two operations which it relies upon, and following them, we state a related lemma to be used in the proof of distance.

Definition 2.3. *Given $v = (v_1, \ldots, v_n) \in \{0,1\}^n$, $i \geq 0$ and $r \geq n + i$ we define $shift_r(v, i) \in \{0,1\}^r$ to be $(\underbrace{0, \ldots, 0}_{r-n-i \text{ times}}, v_1, \ldots, v_n, \underbrace{0, \ldots, 0}_{i \text{ times}})$, that is, the shift of v i places to left, and adding zeroes on the left and right so that it is of length r.*

Secondly, for $m \in \mathbb{N}$, $t \in [m]$ and $r \geq n + (t-1)^2$, we define the function $Eval_t : (\{0,1\}^n)^m \to \{0,1\}^r$ by

$$Eval_t(x_1, \ldots, x_m) = \sum_{i=1}^{t} shift_r(x_i, (i-1)(t-1)),$$

[7] We are being explicit since Definition 1.1 was for binary inputs.

the summation identifying $\{0,1\}^n$ *with* $\mathbb{F}_2^n$ *(that is, done entry-wise modulo* 2*).*

The proof of distance of the to-be constructed tree code is based on the following lemma.

Lemma 2.4. *Let* $x_1,\ldots,x_m \in \{0,1\}^n$ *be such that* $|\{i \mid x_i \neq \bar{0}\}| = \ell > 0$*. Then*

$$|\{t \in [m] \mid t \geq \min\{i \mid x_i \neq \bar{0}\} \wedge \mathit{Eval}_t(x_1,\ldots,x_m) = \bar{0}\}| < \ell.$$

Before turning to prove Lemma 2.4, we conclude from it the construction which yields Lemma 2.2.

Proof for Lemma 2.2 (construction of $\mathcal{TC}_\ell$*).* Let $\ell \in \mathbb{N}$. Set $n = \ell^2$, $m = \ell$ and $r = \ell^2 + (\ell-1)^2$ and we will invoke the definition of Eval_t from Definition 2.3 with these n, m and r. On input $x_1,\ldots,x_m \in \{0,1\}^n$ we define for every $t \in [\ell]$

$$\mathcal{TC}_\ell(x_1,\ldots,x_m)_t = (x_t, \mathrm{Eval}_t(x_1,\ldots,x_m)).$$

Since Eval is online, so is $\mathcal{TC}_\ell$. At each time t the length of $\mathcal{TC}_\ell(x_1,\ldots,x_m)_t$ is $n + r = 2\ell^2 + (\ell-1)^2$. The fact that $\mathcal{TC}_\ell$ has distance more than $1/2$ follows from Lemma 2.4. Indeed, assume towards contradiction that there are $(x_1,\ldots,x_m)$ and $(x'_1,\ldots,x'_m)$ such that $i \in [m]$ is the smallest such that $x_i \neq x'_i$ and there is $h \in [m-i+1]$ such that $\mathcal{TC}_\ell(x_1,\ldots,x_m)$ and $\mathcal{TC}_\ell(x'_1,\ldots,x'_m)$ agree on at least $h/2$ locations in $[i, i+h-1]$. Then $\mathrm{Eval}_1(x'_1-x_1),\ldots,\mathrm{Eval}_{i+h-1}(x'_{i+h-1}-x_{i+h-1})$ contradict Lemma 2.4 since there are at most $h/2$ indices $j \in [i+h-1]$ such that $x'_j - x_j \neq \bar{0}$ while $(\mathrm{Eval}_i(x'_i-x_i),\ldots,\mathrm{Eval}_{i+h-1}(x'_{i+h-1}-x_{i+h-1}))$ has at least $h/2$ zeroes. Finally, clearly $\mathcal{TC}_\ell$ is explicit. □

It thus only remains to prove Lemma 2.4. We will give a simple proof by induction, of a strictly stronger lemma, Lemma 2.7. To describe it easily we make two definitions.

Definition 2.5. *For every* $x \in \{0,1\}^r$ *such that* $x \neq \bar{0}$ *we define Left-Most-Nonzero*$(x) = \min\{i \in [r] \mid x_i \neq 0\}$.

Note that Left-Most-Nonzero gets vectors of length r because in the next definition it is applied to shifted vectors.

Definition 2.6. *For every not-all-zero* $x_1,\ldots,x_m \in \{0,1\}^n$ *and* $t \geq \min\{i \in [m] \mid x_i \neq \bar{0}\}$ *we define*

$$\textit{Left-Most-Nonzero-Column}_t(x_1,\ldots,x_m) = \min_{i \leq t \mid x_i \neq \bar{0}} \textit{Left-Most-Nonzero}(\textit{shift}_r(x_i,(i-1)(t-1))).$$

With that we can state and prove the stronger lemma.

Lemma 2.7. *The following holds for every $\ell \in \mathbb{N}$. Let $x_1, \ldots, x_m \in \{0,1\}^n$ be such that the set $I(x_1, \ldots, x_m) = \{i \mid x_i \neq \bar{0}\}$ is of size ℓ. Then the set*

$$T(x_1, \ldots, x_m) = \{t \in [m] \mid t \geq \min I(x_1, \ldots, x_m) \wedge \mathit{Eval}_t(x_1, \ldots, x_m)_{\mathit{Left\text{-}Most\text{-}Nonzero\text{-}Column}_t(x_1,\ldots,x_m)} = 0\}$$

is of size less than ℓ.

Proof. The proof is by induction on ℓ. In the base case $\ell = 1$, there is only one x_{i^*} which is non-zero and it trivially cannot be that for some $t \geq i^*$,

$$\mathrm{Eval}_t(x_1, \ldots, x_m)_{\text{Left-Most-Nonzero-Column}_t(x_1,\ldots,x_m)} = 0.$$

For the induction step, let $\ell > 1$, and assume towards contradiction that there is $x_1, \ldots, x_m$ such that $I(x_1, \ldots, x_m) = \ell$, while $T(x_1, \ldots, x_m)$ is of size at least ℓ. Let $t^* = \min T(x_1, \ldots, x_m)$ and

$$S = \{i \in [m] \mid (\mathrm{shift}_r(x_i, (i-1)(t^*-1)))_{\text{Left-Most-Nonzero-Column}_{t^*}(x_1,\ldots,x_m)} \neq 0\}.$$

Notice that Left-Most-Nonzero-Column$_{t^*}(x_1, \ldots, x_m)$ is well defined as $t^* \in T(x_1, \ldots, x_m)$ and that by its definition S is non-empty. Further let $i^* = \min S$ and notice that $t^* \geq i^*$ since $i^* \in I(x_1, \ldots, x_m)$ and $t^* \in T(x_1, \ldots, x_m)$. Further, as $t^* \in T(x_1, \ldots, x_m)$ we have the following equalizes which hold modulo 2

$$\begin{aligned}
0 &= \mathrm{Eval}_{t^*}(x_1, \ldots, x_m)_{\text{Left-Most-Nonzero-Column}_{t^*}(x_1,\ldots,x_m)} \\
&= \sum_{i \in S \mid i \leq t^*} \mathrm{shift}_r(x_i, (i-1)(t^*-1))_{\text{Left-Most-Nonzero-Column}_{t^*}(x_1,\ldots,x_m)} \\
&= 1 + \sum_{i \in S \setminus \{i^*\} \mid i \leq t^*} \mathrm{shift}_r(x_i, (i-1)(t^*-1))_{\text{Left-Most-Nonzero-Column}_{t^*}(x_1,\ldots,x_m)}
\end{aligned}$$

and thus $S \setminus \{i^*\}$ is also non-empty, and $i^{**} := \min S \setminus \{i^*\} \leq t^*$. By the definition of Left-Most-Nonzero-Column$_{t^*}(x_1, \ldots, x_m)$ and as

$$\begin{aligned}
&\mathrm{shift}_r(x_{i^*}, (i^*-1)(t^*-1))_{\text{Left-Most-Nonzero-Column}_{t^*}(x_1,\ldots,x_m)} = 1, \\
&\mathrm{shift}_r(x_{i^{**}}, (i^{**}-1)(t^*-1))_{\text{Left-Most-Nonzero-Column}_{t^*}(x_1,\ldots,x_m)} = 1,
\end{aligned}$$

and it must be that

$$\begin{aligned}
\text{Left-Most-Nonzero}(\mathrm{shift}_r(x_{i^*}, (i^*-1)(t^*-1))) &= \text{Left-Most-Nonzero}(\mathrm{shift}_r(x_{i^{**}}, (i^{**}-1)(t^*-1))) \\
&= \text{Left-Most-Nonzero-Column}_{t^*}(x_1, \ldots, x_m)
\end{aligned}$$

since $i^* < i^{**} \leq t^*$. It follows that for every $t > t^*$,

$$\begin{aligned}
\text{Left-Most-Nonzero}(\mathrm{shift}_r(x_{i^*}, (i^*-1)(t-1))) &> \text{Left-Most-Nonzero}(\mathrm{shift}_r(x_{i^{**}}, (i^{**}-1)(t-1))) \\
&\geq \text{Left-Most-Nonzero-Column}_t(x_1, \ldots, x_m)
\end{aligned}$$

since $(i^*-1)(t-1)-(i^*-1)(t^*-1) < (i^{**}-1)(t-1)-(i^{**}-1)(t^*-1)$ and by the definition of shift. In particular for every $t > t^*$

$$\text{Left-Most-Nonzero-Column}_t(x_1,\ldots,x_m) \neq \text{Left-Most-Nonzero}(\text{shift}_r(x_{i^*},(i^*-1)(t-1))). \tag{2.1}$$

Now, consider $x'_1, x'_2, \ldots, x'_m$ where $x'_i = x_i$ if $i \neq i^*$ and $x'_{i^*} = \bar{0}$. The set $I(x'_1,\ldots,x'_m) = \{i \mid x'_i \neq \bar{0}\}$ is of size $\ell - 1 > 0$, and $i^{**} \in I'(x'_1,\ldots,x'_m)$. Furthermore, for every $t \in T(x_1,\ldots,x_m) \setminus \{t^*\}$ we have that

$$\begin{aligned}
&\text{Eval}_t(x'_1,\ldots,x'_m)_{\text{Left-Most-Nonzero-Column}_t(x'_1,\ldots,x'_m)} \\
&= \sum_{i\le t} \text{shift}_r(x'_i,(i-1)(t-1))_{\text{Left-Most-Nonzero-Column}_t(x'_1,\ldots,x'_m)} \\
&= \sum_{i\le t} \text{shift}_r(x_i,(i-1)(t-1))_{\text{Left-Most-Nonzero-Column}_t(x_1,\ldots,x_m)} = 0
\end{aligned}$$

where the last equality is by $t \in T(x_1,\ldots,x_m)$, and the penultimate equality is as for any $t > t^*$, since $x_i \neq x'_i$ only for $i = i^*$, by Equation (2.1)

$$\text{Left-Most-Nonzero-Column}_t(x'_1,\ldots,x'_m) = \text{Left-Most-Nonzero-Column}_t(x_1,\ldots,x_m).$$

It follows that $T(x_1,\ldots,x_m)\setminus\{t^*\} \subseteq T(x'_1,\ldots,x'_m)$ (note that $\min I(x'_1,\ldots,x'_m) \le i^{**} \le t^*$ and for every $t \in T(x_1,\ldots,x_m)\setminus\{t^*\}$, $t > t^*$, by the definition of t^*). Thus $x'_1,\ldots,x'_m$, since $|T(x'_1,\ldots,x'_m)| \ge \ell - 1$, contradict the induction hypothesis, as desired.

□

2.1 Decoding from a weak guarantee of polynomial-fraction of errors

We start by saying what we mean be decoding a tree code $\mathcal{TC} : \{0,1\}^n \to (\{0,1\}^r)^n$ from a γ-fraction of errors. Note that as a tree code is not a code per se[8], a different than usual notion of decoding is the right one.

Definition 2.8 (See also Definition 3.2 in [NW19])**.** *We say that a tree code $\mathcal{TC} : \{0,1\}^n \to (\{0,1\}^r)^n$ can be efficiently decoded from a γ-fraction of errors if there exists an algorithm which runs in time* poly(n) *and for every $t \in [n]$, given input $y_1,\ldots,y_t \in \{0,1\}^r$ such that:*

- *There is $(x_1,\ldots,x_t) \in \{0,1\}^t$ such that* for every suffix $(y_i,\ldots,y_t)$,

$$\mathsf{Dist}(\mathcal{TC}(x_1,\ldots,x_t)_{i,\ldots,t}, y_i,\ldots,y_t) \le \gamma(t-i+1),$$

[8]Indeed it suffices to change only the last output of the tree code and obtain a valid code word.

outputs $x_1, \ldots, x_t$.

We claim that the binary shifts large-alphabet tree code can be efficiently decoded from a polynomial number of errors. We need however to consider a slightly different notion of decoding, one that measures errors in *bits* of every suffix, rather than blocks. We remark that in [NW19] (Theorem 3.1) it is proven that the problem of decoding the [CHS18]-framework reduces to the problem of decoding the large-alphabet tree code.

Claim 2.9. *Let* $\mathcal{TC}_\ell : (\{0,1\}^{\ell^2})^\ell \to (\{0,1\}^{2\ell^2+(\ell-1)^2})^\ell$ *be the large-alphabet tree code defined in Section 2. Set* $\gamma = \frac{1}{16\ell^2}$. *Assume that* $\tilde{y}'_1, \ldots, \tilde{y}'_t \in \{0,1\}^{2\ell^2+(\ell-1)^2}$ *are such that:*

- *There is* $(x_1, \ldots, x_t) \in (\{0,1\}^{\ell^2})^t$ *such that* for every suffix $(y'_i, \ldots, y'_t)$,

  $$\mathsf{Dist}\left((\mathcal{TC}(x_1, \ldots, x_t)_i \circ \cdots \circ \mathcal{TC}(x_1, \ldots, x_t)_t), (\tilde{y}'_i \circ \cdots \circ \tilde{y}'_t)\right) \leq \gamma \cdot (t-i+1) \cdot (2\ell^2 + (\ell-1)^2),$$

  that is, the relative distance of the bits *of every suffix is at most* γ.

Then, there is a procedure which gets as input $\tilde{y}'_1, \ldots, \tilde{y}'_t$ *and outputs* $(x_1, \ldots, x_t)$, *which runs in time* $\mathrm{poly}(\ell)$.

Proof. Write $\tilde{y}'_i = (\tilde{x}_i, \tilde{y}_i)$ for every $i \in [t]$. That is, we split each possibly corrupted output to the alleged systematic part $\tilde{x}_i$, and the alleged shifted-vectors sum part, $\tilde{y}_i$.

We not first that we can immediately decode x_t, since it must by that $\tilde{x}_t = x_t$. Indeed, if we consider just the length-1 suffix $\tilde{x}_t$, then $\mathsf{Dist}(\tilde{x}_t, x_t) \leq \gamma \cdot 3\ell^2 < 1$.

We proceed to show, by induction on $i \in \{0, \ldots, t-1\}$, that we can decode the sequence of rows $x_1, \ldots, x_i$ (in the case $i = 0$, this sequence is empty, and thus the base case of the induction is trivial). Let, therefore $i \geq 1$, and we assume that the hypothesis holds for $i-1$; thus we can assume that we have decoded $x_1, \ldots, x_{i-1}$, and we need to show that we can decode x_i. Towards that, we show that we can decode every bit $j \in [\ell^2]$ of x_i, that is $(x_i)_j$. Fix $j \in [\ell^2]$. We will use only the suffix $(\tilde{x}_{i+1}, \tilde{y}_{i+1}), \ldots, (\tilde{x}_t, \tilde{y}_t)$ part of the input in order to decode $(x_i)_j$.

As said in the introduction, for every $h \in \{i+1, \ldots, t\}$, y_h contains (at some column of y_h), the parity bit of a sum that involves $(x_i)_j$ – the sum of the coordinates which fall on a line of the $t \times \ell^2$ grid which passes through (i, j). The slopes of these line are growing with h, and therefore all these slopes are distinct, and thus the lines are disjoint except for the point (i, j). For every $h \in \{i+1, \ldots, t\}$ let j_h denote the index of the column of y_h which contains the parity bit of the line which passes through (i, j).

First, we argue that for more than $\frac{t-i}{4}$ of these lines $h \in \{i+1, \ldots, t\}$, $(\widetilde{y_h})_{j_h} = (y_h)_{j_h}$. Indeed, per the hypothesis of the claim and considering the $i+1, \ldots, t$ suffix of the input

$$|\{h \in \{i+1, \ldots, t\} \mid (\widetilde{y_h})_{j_h} \neq (y_h)_{j_h}\}| \leq \gamma \cdot (t-i) \cdot 3\ell^2 < \frac{t-i}{4},$$

as $\gamma = \frac{1}{16\ell^2}$.

Second, among the lines $h \in \{i+1, \ldots, t\}$, call a line "bad" if for there is a point (i', j') which lies in a row $i+1, \ldots, t$ of the $t \times \ell^2$ grid, such that $(\widetilde{x}_{i'})_{j'} \neq (x_{i'})_{j'}$. Since the lines all disjoint on rows $i+1, \ldots, t$, the number of bad lines is at most the number of corruptions of $\widetilde{x}_{i+1}, \ldots, \widetilde{x}_t$. Thus,

$$|\{h \in \{i+1, \ldots, t\} \mid h \text{ is bad}\}| \leq \gamma \cdot (t-i) \cdot 3\ell^2 < \frac{t-i}{4}.$$

Combining the two bounds, less than $2 \cdot \frac{t-i}{4} = \frac{t-i}{2}$ of the lines either have their parity bit incorrect, or have any corruption on them on $\widetilde{x}_{i+1}, \ldots, \widetilde{x}_t$. Thus, a majority of the $t-i$ lines have both their parity bit correct and all of the points of $\widetilde{x}_{i+1}, \ldots, \widetilde{x}_t$. For any such good line, the sum of the parity bit, its points on $\widetilde{x}_{i+1}, \ldots, \widetilde{x}_t$, and all of the points of the line which fall on $x_1, \ldots, x_{i-1}$ (which, recall, we have) – is equal to $(x_i)_j$. Thus, doing this computation for each line, and taking the majority, decodes $(x_i)_j$. We conclude that the induction step holds. □

References

[BH20] Siddharth Bhandari and Prahladh Harsha. A note on the explicit constructions of tree codes over polylogarithmic-sized alphabet. *arXiv preprint arXiv:2002.08231*, 2020.

[Bra12] Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 161–167. ACM, New York, 2012.

[BYCN21] Inbar Ben Yaacov, Gil Cohen, and Anand Kumar Narayanan. Candidate Tree Codes via Pascal Determinant Cubes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207, pages 54:1–54:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[BYCY22] Inbar Ben Yaacov, Gil Cohen, and Tal Yankovitz. Explicit binary tree codes with sub-logarithmic size alphabet. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 595–608, 2022.

[CHS18] Gil Cohen, Bernhard Haeupler, and Leonard J. Schulman. Explicit binary tree codes with polylogarithmic size alphabet. In *STOC'18—Proceedings of*

the 50th Annual ACM SIGACT Symposium on Theory of Computing, pages 535–544. ACM, New York, 2018.

[CSS25] Gil Cohen, Leonard J Schulman, and Piyush Srivastava. The rate-immediacy barrier in explicit tree code constructions. *arXiv preprint arXiv:2504.09388*, 2025.

[GHK+16] R. Gelles, B. Haeupler, G. Kol, N. Ron-Zewi, and A. Wigderson. Towards optimal deterministic coding for interactive communication. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1922–1936, 2016.

[MRR25] Tamer Mour, Alon Rosen, and Ron Rothblum. Locally testable tree codes. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5523–5559. SIAM, 2025.

[MS14] Cristopher Moore and Leonard J. Schulman. Tree codes and a conjecture on exponential sums. In *ITCS'14—Proceedings of the 2014 Conference on Innovations in Theoretical Computer Science*, pages 145–153. ACM, New York, 2014.

[NW19] A. K. Narayanan and M. Weidner. On decoding cohen-haeupler-schulman tree codes. *arXiv preprint arXiv:1909.07413*, 2019.

[NW20] Anand Kumar Narayanan and Matthew Weidner. On decoding Cohen-Haeupler-Schulman tree codes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1337–1356. SIAM, 2020.

[Pud16a] P. Pudlák. Linear tree codes and the problem of explicit constructions. *Linear Algebra and its Applications*, 490:124–144, 2016.

[Pud16b] Pavel Pudlák. Linear tree codes and the problem of explicit constructions. *Linear Algebra Appl.*, 490:124–144, 2016.

[Pud22] Pavel Pudlák. On matrices potentially useful for tree codes. *Information Processing Letters*, 174:106180, 2022.

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the 25th annual ACM Symposium on Theory of Computing*, pages 747–756, 1993.

[Sch94] Leonard J. Schulman. Postscript of 21 september 2003 to coding for interactive communication, 1994.

[Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Trans. Inform. Theory*, 42(6, part 1):1745–1756, 1996. Codes and complexity.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il