Constructing Large Families of Pairwise Far Permutations: Good Permutation Codes Based on the Shuffle-Exchange Network

Oded Goldreich* Avi Wigderson†

December 27, 2020

Abstract

We consider the problem of efficiently constructing an as large as possible family of permutations such that each pair of permutations are far part (i.e., disagree on a constant fraction of their inputs). Specifically, for every $n \in \mathbb{N}$, we present a collection of $N = N(n) = (n!)^{\Omega(1)}$ pairwise far apart permutations $\{\pi_i : [n] \to [n]\}_{i \in [N]}$ and a polynomial-time algorithm that on input $i \in [N]$ outputs an explicit description of π_i.

From a coding theoretic perspective, we construct permutation codes of constant relative distance and constant rate along with efficient encoding (and decoding) algorithms. This construction is easily extended to produce codes on smaller alphabets in which every codeword is balanced; namely, each symbol appears the same number of times.

Our construction combines routing on the Shuffle-Exchange network with any good binary error correcting code. Specifically, we uses codewords of a good binary code in order to determine the switching instructions in the Shuffle-Exchange network.

Contents

*Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. E-mail: oded.goldreich@weizmann.ac.il. Partially supported by the Israel Science Foundation (grant No. 1041/18).

†School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540, USA. E-mail: avi@ias.edu. Research partially supported by NSF grant CCF-1900460.

1 Introduction

The notion of a good code $C : \Sigma^k \to \Sigma^n$ is well known. Such a code has constant relative distance (i.e., images of C differ in $\Omega(n)$ locations), and constant rate (i.e., $k = \Omega(n)$). In other words, the image of C, called its set of codewords, are pairwise *far apart* (in Hamming distance), and their number is polynomially related to the domain in which they reside (i.e., $|C(\Sigma^k)| = |\Sigma^n|^{\Omega(1)}$).

In this paper, we construct such codes in which all codewords are permutations. Namely, $\Sigma = [n]$, and all symbols in any codeword are distinct. In short, a good *permutation code* is a collection of permutations of maximal (up to polynomial) size such that the permutations in the collection are pairwise far apart. The coding theoretic perspective is spelled-out in Section 5. We mention that permutation codes (suggested in [7]) have been extensively studied before, but it seems that the focus of these studies was on different issues (see, e.g., [2, 3, 6]).

The concrete motivation for this work arose in our prior work [5], where we used a *large* collection of permutations of $[n]$ that are pairwise far-apart. Indeed, we say that $\pi, \pi' : [n] \to [n]$ are far apart if $|\{j \in [n] : \pi(j) \neq \pi'(j)\}| = \Omega(n)$ and "large" means either of size $(n!)^{\Omega(1)}$ or of size $\exp(\Omega(n))$, depending on the application in [5]. Specifically, in our prior work [5], we obtained the following two results:

1. A collection of $N = (n!)^{\Omega(1)}$ pairwise far-apart permutations over $[n]$ can be constructed in $\mathrm{poly}(N)$-time.

   This is done by a greedy algorithm that selects the permutations one by one, while relying on the existence of a permutation that augments the current collection (while preserving the distance condition).

2. A collection of $N = \exp(\Omega(n))$ pairwise far-apart permutations over $[n]$ can be *locally* constructed such that each permutation is constructed in $\mathrm{poly}(n)$-time (i.e., on input $i \in [N]$, the polynomial-time algorithm returns a description of the i^{th} permutation).

   This is done by using sequences of disjoint transpositions determined via a good binary error correcting code. (Our current construction will build on this idea.)

The foregoing results beg the challenge of locally constructing a collection of $N = (n!)^{\Omega(1)}$ pairwise far-apart permutations over $[n]$; that is, a collection that comes along with a polynomial-time algorithm that, on input $i \in [N]$, returns a description of the i^{th} permutation (i.e., the algorithm should run in $\mathrm{poly}(\log N)$-time). In this work, we meet this challenge by proving –

Theorem 1 (efficient construction of a large collection of pairwise far-apart permutations): *For every $n \in \mathbb{N}$, there exists a collection of $N = N(n) = (n!)^{\Omega(1)}$ pairwise far apart permutations $\{\pi_i : [n] \to [n]\}_{i \in [N]}$ and a polynomial-time algorithm that on input $i \in [N]$ outputs an explicit description of π_i.*

Theorem 1 follows directly from the next theorem, which presents the construction in a coding-theoretic language. It allows a very general conversion of binary codes to permutation codes that essentially preserves all the main parameters and efficiency.

Theorem 2 (good permutation codes): *For $\ell, k \in \mathbb{N}$ and $n = 2^\ell$, let $C : \{0,1\}^k \to \{0,1\}^{n/2}$ be any binary code, and $S_n \subset [n]^n$ denote the set of all permutations over $[n]$. Then, there exists a permutation code $C' : (\{0,1\}^k)^\ell \to S_n$ that uses C as a black-box such that evaluating C'* (i.e., encoding) *reduces to ℓ evaluations of C and the following properties hold:*

- *The rate of C' equals half the rate of C.*
- *The relative distance of C' is at least that of C.*
- *If C has a unique decoding from $\epsilon \cdot n$ errors, then so has C'. Furthermore, decoding C' reduces to ℓ invocations of decoding algorithm for C.*

Furthermore, individual symbols in codewords of C' depend on ℓ symbols in codewords of C, and can be computed in $\mathrm{poly}(\ell)$*-time.*

Actually, an even more general result holds, yielding so-called *constant composition codes* (cf., [4]) of constant rate and constant relative distance. These are codes of block-length $n = 2^\ell$ over $[2^{\ell'}]$ such that every symbol appears $2^{\ell-\ell'}$ times in each codeword, where the case of $\ell' = \ell$ coincide with permutation codes (and $\ell' = 1$ correspond to the special case of binary codes). For details see Section 5.

Theorems 1 and 2 are proved by considering a routing network for n elements. Such a network consists of layers such that in each layer specific pairs of elements can be switched, where each switch is governed by a corresponding instruction bit that determines whether or not to switch the pair. The basic idea is to use codewords of a good binary code (as postulated in Theorem 2) in order to determine the instruction bits for each layer. The point is showing that this works; that is, that this does yield permutations that are pairwise far apart. We prove this fact while referring to ℓ layers of the Shuffle-Exchange network for $n = 2^\ell$ elements.[1]

The simple nature of our construction immediately implies a simple encoding algorithm for our permutation code (Theorem 2). Furthermore, our distance analysis also leads to an efficient decoding algorithm from errors, by a simple reduction to any decoding algorithm of the original binary code used in out construction (as postulated in the last item of Theorem 2).

Perspective: Routing on the Shuffle-Exchange network. Starting with Benes [1], much research was devote to routing *all* possible permutations on the Shuffle-Exchange network (and related ones) in ways that optimize various parameters such as parallel time and congestion (see, e.g., Valiant's seminal work [8, 9]). These routing schemes use several repetitions of the network, whereas we shall use the known fact that a single repetition efficiently routes many permutations. Specifically, we show that if these routes are chosen according to codewords of a good error correcting code, then the resulting permutations are pairwise far apart.

Perspective: Permutation codes. As mentioned above, permutation codes (suggested by Slepian [7]) have been extensively studied before. It seems that the focus of most of these studies is on using permutation groups as codes (see, e.g., [2, 6]), and mostly on the study of different parameters. These results are inferior to ours in standard coding parameters and efficiency (mainly because we are not restricted by any algebraic structure). Alternative approaches have been suggested as well, including distance preserving maps from binary and other codes (see, e.g., [3]). However, the focus seems to be on fixed sizes and simulations (see [3]), and result in much weaker parameters and efficiency. *Constant composition codes*, briefly mentioned after Theorem 2, have

[1] The general case (i.e., n that is not necessarily a power of 2) is easily reducible to the special case of $2^{\lfloor \log_2 n \rfloor}$ elements. Specifically, we extend $\pi : [2^\ell] \to [2^\ell]$ to $\pi' : [n] \to [n]$, by letting $\pi'(x) = \pi(x)$ if $x \in [2^\ell]$ and $\pi'(x) = x$ otherwise.

been studied as well (see e.g., [4]). However, as in the case of permutation codes, the known results are inferior to ours in standard coding parameters and efficiency.

Organization. We start with some preliminaries regarding the Shuffle-Exchange network (Section 2), which is followed by our construction and its analysis (Sections 3 and 4, respectively). This establishes Theorem 1. In Section 5, we offer a coding theoretic perspective, which leads us to present an error-correcting procedure for the permutation code that we constructed. This establishes Theorem 2. The extension to good constant composition codes appears in Section 5.2.

2 Preliminaries

Our family of permutations will be described by an injective map from $\{0,1\}^{\ell \cdot 2^{\ell-1}}$ to the set of permutations over the set $\{0,1\}^\ell$. Each permutation in the image of this map may be viewed as generated by a process, initiated on the identity permutation, such that each step is governed by an input bit that specifies whether or not to perform a transposition between two predetermined locations. The specific process we have in mind is defined by the Shuffle-Exchange network and is described next.

We view each permutation (over $\{0,1\}^\ell$) as a sequence of 2^ℓ different ℓ-bit long strings that reside in 2^ℓ cells, which are named by ℓ-bit long strings. Initially, the string x resides in cell x. In each step of the Shuffle-Exchange network strings that reside in paired cells may be switched, as detailed next.

The Shuffle-Exchange network is viewed as consisting of ℓ layers of $2^{\ell-1}$ switches between pairs of cells, where the cells are paired according to their name (viewed as an ℓ-bit string). Specifically, in the i^{th} step, we pair cells whose locations (i.e., names) differ only in their i^{th} bit. This defines an ℓ-step process of permuting $\{0,1\}^\ell$, where initially the string $x \in \{0,1\}^\ell$ resides in cell x.

For $i = 1, ..., \ell$, in the i^{th} step, pairs of strings that are at locations that differ only in the i^{th} bit are possibly switched. The decision of whether or not to switch the contents of cells $\alpha 0 \beta$ and $\alpha 1 \beta$, where $(\alpha, \beta) \in \{0,1\}^{i-1} \times \{0,1\}^{\ell-i}$, is determined by an instruction bit, denoted $\eta_{\alpha,\beta}$; that is, if $\eta_{\alpha,\beta} = 1$, then we switch the contents, and otherwise we don't.

The $2^{\ell-1}$-bit long sequence of instruction bits for a step is called an instruction sequence, and the instruction sequence for the i^{th} step is indexed by pairs of strings of length $i-1$ and $\ell - i$ respectively. The permutation defined by the instruction sequence $\overline{\eta} = (\eta_{0^{i-1},0^{\ell-i}}, ..., \eta_{1^{i-1},1^{\ell-i}})$ of the i^{th} step is denoted $\pi^{(i)}_{\overline{\eta}} : \{0,1\}^\ell \to \{0,1\}^\ell$, and for every $(\alpha, \sigma, \beta) \in \{0,1\}^{i-1} \times \{0,1\} \times \{0,1\}^{\ell-i}$ it holds that

$$\pi^{(i)}_{\overline{\eta}}(\alpha\sigma\beta) \stackrel{\text{def}}{=} \begin{cases} \alpha\sigma\beta & \text{if } \eta_{\alpha,\beta} = 0, \\ \alpha\overline{\sigma}\beta & \text{otherwise} \quad (\text{where } \overline{\sigma} = 1 - \sigma). \end{cases} \tag{1}$$

For every sequence of ℓ instruction sequences, denoted $w_1, ..., w_\ell$, we define the permutation $\pi_{w_1,...,w_\ell} : \{0,1\}^\ell \to \{0,1\}^\ell$ such that

$$\pi_{w_1,...,w_\ell}(x) \stackrel{\text{def}}{=} \pi^{(\ell)}_{w_\ell} \circ \cdots \circ \pi^{(1)}_{w_1}(x). \tag{2}$$

Indeed, this is the permutation generated by the ℓ-step process when using w_i as an instruction sequence for Step i.

We stress that the foregoing ℓ-step process cannot generate all permutations of $\{0,1\}^\ell$. One

way of verifying this fact is observing that the number of generated permutation is upper-bounded[2] by the number of possible ℓ-step instruction sequences, whereas the latter number is $(2^{2^{\ell-1}})^\ell = 2^{2^\ell \cdot \ell/2} \ll 2^\ell!$.

Another point worthy of spelling out is that the permutation $\pi_{w_\ell,....,w_1}$ is the inverse of $\pi_{w_1,....,w_\ell}$; this holds since for every $i \in [\ell]$ and $w \in \{0,1\}^{\ell-1}$ it holds that $\pi_w^{(i)}$ is an involution.

3 The Construction

The basic idea it to use instruction sequences that are $2^{\ell-1}$-bit long codewords of some good binary error correcting code (rather than all possible instruction sequences). That is, given an ℓ-tuple of codewords $w_1, ..., w_\ell \in \{0,1\}^{2^{\ell-1}}$, we consider the permutation (over $\{0,1\}^\ell$) that is defined by using w_i as the instruction sequence for step i.

The key observation is that any difference between two ℓ-tuples of $2^{\ell-1}$-bit long strings that are used as instruction sequences yields a difference between the two generated permutations. Furthermore, a difference caused in Step i cannot be undone by a subsequent step, although differences in the instruction sequences used in different steps may "reinforce" an existing difference. (See rigorous analysis in Section 4.) Hence, using two different ℓ-tuples of codewords, even if these ℓ-tuples differ only in one codeword, yields $\Omega(2^\ell)$ differences between the generated permutations, since different codewords disagree on $\Omega(2^{\ell-1})$ bits.

The idea that codewords yield far apart permutations can be traced back to our prior work [5], where the codewords were used in order to define transpositions; that is, in terms of the current presentation, only a single step was performed. Hence, the collection of permutations constructed in [5] has size $2^{\Theta(2^\ell)}$. In contrast, the collection constructed here has size $(2^{\Theta(2^\ell)})^\ell = (2^\ell!)^{\Theta(1)}$.

The actual construction. Let $C : \{0,1\}^k \to \{0,1\}^{2^{\ell-1}}$ be an efficiently computable binary code of relative constant distance and constant rate (i.e., every two codewords disagree on $\Omega(2^{\ell-1})$ bits and $k = \Omega(2^{\ell-1})$). Then, the family of permutations (over $\{0,1\}^\ell$) that we consider includes all permutations $\pi_{w_1,....,w_\ell} : \{0,1\}^\ell \to \{0,1\}^\ell$ such that w_i is a codeword of C, for every $i \in [\ell]$. That is, we consider the family of permutations

$$\left\{\pi_{C(r_1),....,C(r_\ell)} : \{0,1\}^\ell \to \{0,1\}^\ell\right\}_{r_1,...,r_\ell \in \{0,1\}^k} \tag{3}$$

Recall that $\pi_{w_1,....,w_\ell}$ is defined in Eq. (2), and that it is the composition of the $\pi_{w_i}^{(i)}$'s. Specifically, defining $x^{(0)} \stackrel{\text{def}}{=} x$, for every $i \in [\ell]$ we have $x^{(i)} = \pi_{w_i}^{(i)}(x^{(i-1)})$, and $\pi_{w_1,....,w_\ell}(x) = x^{(\ell)}$.

4 The Analysis

The key observation is that the value of $\pi_{w_1,....,w_\ell}(x)$ is the exclusive-or of x and (some specific) ℓ bits in the ℓ instruction sequences (i.e., one bit in each of the w_i's). Furthermore, the location of the i^{th} bit (i.e., the bit that gets xor-ed with x) in w_i is determined by x and $w_1, ..., w_{i-1}$, obliviously of the sequences $w_i, ..., w_\ell$.

[2] Actually, we will prove that this upper bound is tight, by showing that each sequence results in different permutation. (See comment following Theorem 4.)

Observation 3 (key observation (folklore)): *For any $x \in \{0,1\}^\ell$ and $w_1, ..., w_\ell \in \{0,1\}^{2^{\ell-1}}$, it holds that $\pi_{w_1,...,w_\ell}(x) = x \oplus b_1 \cdots b_\ell$ where for every $i \in [\ell]$ the bit b_i is a bit in w_i whose location is determined by x and $w_1, ..., w_{i-1}$. Specifically, letting $x_1 \cdots x_\ell = x$ and $w_{i,0^{\ell-1}} \cdots w_{i,1^{\ell-1}} = w_i$, it holds that $b_i = w_{i,y'x''}$, where $y' = x_1 \cdots x_{i-1} \oplus b_1 \cdots b_{i-1}$, and $x'' = x_{i+1} \cdots x_\ell$; that is, b_i is the bit that resides in location $y'x''$ in w_i, where* locations in w_i *are indexed as $(\ell-1)$-bit long strings.*

This observation is well-known and was used explicitly and implicitly in many works that study routing on the Shuffle-Exchange network. What we believe is new is that this observation imply the distance preservation feature stated in Theorem 4.

Proof: We first observe that, *for any $z = z_1 \cdots z_\ell \in \{0,1\}^\ell$, $i \in [\ell]$ and $w \in \{0,1\}^{2^{\ell-1}}$, it holds that $\pi^{(i)}_w(z) = z \oplus 0^{i-1}b0^{\ell-i}$ where b is the bit that resides in location $z_1 \cdots z_{i-1}z_{i+1} \cdots z_\ell$ in w_i*; that is, letting $(\eta_{0^{i-1},0^{\ell-i}}, ..., \eta_{1^{i-1},1^{\ell-i}}) = w_i$, it holds that $b = \eta_{z_1 \cdots z_{i-1}, z_{i+1} \cdots z_\ell}$.

The main claim follows by induction on i. Specifically, defining $x^{(0)} \stackrel{\text{def}}{=} x$ and $x^{(i)} = x^{(i)}_1 \cdots x^{(i)}_\ell \stackrel{\text{def}}{=} \pi^{(i)}_{w_i}(x^{(i-1)})$, it holds that $x^{(i)} = x^{(i-1)} \oplus 0^{i-1}b_i0^{\ell-i}$, where b_i is the bit that resides in location $x^{(i-1)}_1 \cdots x^{(i-1)}_{i-1}x_{i+1} \cdots x_\ell$ in w_i. Using the induction hypothesis, it follows that $x^{(i)} = x \oplus b_1 \cdots b_i 0^{\ell-i}$ and that $x^{(i-1)}_1 \cdots x^{(i-1)}_{i-1}x_{i+1} \cdots x_\ell = y'x''$ such that $y' = x_1 \cdots x_{i-1} \oplus b_1 \cdots b_{i-1}$ and $x'' = x_{i+1} \cdots x_\ell$. ■

We now prove that the fact that the code C is good (i.e., has relative constant distance and constant rate) implies that the collection of permutations based on it (per Eq. (3)) constitutes a good code. Hence, *good binary codes imply good permutation codes.* The claim regarding rate is evident (since $k \cdot \ell = \Omega(\log(2^\ell!))$ and C is injective) and so we focus on the distance between different permutations.

Theorem 4 (distance between permutations in the family defined by Eq. (3)): *For any $\overline{r} = (r_1, ..., r_\ell) \in (\{0,1\}^k)^\ell$ and $\overline{s} = (s_1, ..., s_\ell) \in (\{0,1\}^k)^\ell$ such that $\overline{r} \neq \overline{s}$, it holds that*

$$\left|\left\{x \in \{0,1\}^\ell : \pi_{C(r_1),...,C(r_\ell)}(x) \neq \pi_{C(s_1),...,C(s_\ell)}(x)\right\}\right| \geq \delta_C \cdot 2^\ell,$$

where δ_C is the relative distance of C.

As a special case (when using the identity map in the role of C), it follows that that different ℓ-step instruction sequences generate different permutations; that is, for every two distinct $\overline{u} = (u_1, ..., u_\ell) \in (\{0,1\}^{2^{\ell-1}})^\ell$ and $\overline{v} = (v_1, ..., v_\ell) \in (\{0,1\}^{2^{\ell-1}})^\ell$, it holds that $\pi_{\overline{u}} \neq \pi_{\overline{v}}$ (i.e., there exists $x \in \{0,1\}^\ell$ such that $\pi_{\overline{u}}(x) \neq \pi_{\overline{v}}(x)$).[3]

Proof: For simplicity of notation, suppose that $i \in [\ell]$ is smallest such that $r_i \neq s_i$. Let $(\eta_{0^{i-1},0^{\ell-i}}, ..., \eta_{1^{i-1},1^{\ell-i}}) = C(r_i)$ and $(\zeta_{0^{i-1},0^{\ell-i}}, ..., \zeta_{1^{i-1},1^{\ell-i}}) = C(s_i)$. Using Observation 3, it follows that if $\eta_{\alpha,\beta} \neq \zeta_{\alpha,\beta}$, then, for both $\sigma \in \{0,1\}$, the ℓ-bit long strings $\pi_{C(r_1),...,C(r_\ell)}(\alpha\sigma\beta)$ and $\pi_{C(s_1),...,C(s_\ell)}(\alpha\sigma\beta)$ differ on their i^{th} bit (whereas they agree on their $i-1$ first bits). By the hypothesis there are at least $\delta_C \cdot 2^{\ell-1}$ locations on which $C(r_i)$ and $C(s_i)$ differ, and each such location contributes two units (to the difference between $\pi_{C(r_1),...,C(r_\ell)}$ and $\pi_{C(s_1),...,C(s_\ell)}$). The claim follows. ■

[3] Specifically, we consider use the identity map as a code $C : \{0,1\}^{2^{\ell-1}} \to \{0,1\}^{2^{\ell-1}}$ (i.e., $k = 2^{\ell-1}$), while noting that $\delta_C = 2^{-(\ell-1)} > 0$. Hence, each tuple of instruction sequences yields a different permutation.

Digest. Note that the difference between $\pi_{C(r_1),...,C(r_\ell)}(x)$ and $\pi_{C(s_1),...,C(s_\ell)}(x)$ was charged to the first index i such that $r_i \neq s_i$. In particular, in that case $\pi_{C(r_1),...,C(r_\ell)}(x)$ and $\pi_{C(s_1),...,C(s_\ell)}(x)$ differ on the i^{th} bit. Indeed, these two strings may differ also on subsequent bits (due to $i' > i$ such that $r_{i'} \neq s_{i'}$), but we avoided double-counting by focusing on a single index i. We also note that working with the first possible index i (such that $r_i \neq s_i$) merely simplifies the notation and clarifies the argument.

5 The Coding Theoretic Perspective

In this section we prove Theorem 2 as well as its extension to constant composition codes, which is stated below as Theorem 6.

5.1 Proof of Theorem 2

The algorithm guaranteed by Theorem 1 asserts a mapping from $[N]$ to a collection of $N = N(n) = (n!)^{\Omega(1)}$ pairwise far apart permutations $\{\pi_i : [n] \to [n]\}_{i\in[N]}$. Specifically, using any good binary code $C : \{0,1\}^k \to \{0,1\}^{2^{\ell-1}}$, the proof identifies $[n]$ with $\Sigma \stackrel{\text{def}}{=} \{0,1\}^\ell$ (and so $n = 2^\ell$ and $N(n) = (2^\ell)^k$), and specifies a mapping from $\{0,1\}^{k\cdot\ell} \equiv \Sigma^k$ to Σ^n. This mapping, denoted M, maps $\overline{r} = (r_1, ..., r_\ell) \in \{0,1\}^{\ell\cdot k}$ to $(\pi_{\overline{C}(\overline{r})}(0^\ell), ..., \pi_{\overline{C}(\overline{r})}(1^\ell)) \in \Sigma^{2^\ell}$, where $\overline{C}(\overline{r}) = (C(r_1), ..., C(r_\ell))$. Needless to say, we can present the $\ell \cdot k$-bit long sequence of $\overline{r}$ (equiv., ℓ-long sequence of r_i's) as a k-long sequence over $\Sigma \equiv \{0,1\}^\ell$.[4] Recalling that the locations in the codewords of M are associated with ℓ-bit strings, the symbol in location x of the codeword $M(\overline{r})$ is $\pi_{\overline{C}(\overline{r})}(x)$; that is, $M(\overline{r})_x = \pi_{\overline{C}(\overline{r})}(x)$.

The relative distance and rate of M. Theorem 4 asserts that the relative distance of M is lower-bounded by the relative distance of C. Let us spell out the fact that the rate of M equals $\frac{k}{n} = \frac{1}{2} \cdot \frac{k}{2^{\ell-1}}$, where $\frac{k}{2^{\ell-1}}$ is the rate of C.

Inverting M (i.e., error-less decoding). By definition, each valid codeword $(Y_{0^\ell}, .., Y_{1^\ell})$ is associated with a unique permutation $\pi_{C(r_1),...,C(r_\ell)}$. When given the codeword, the permutation is given explicitly, but inverting M means that we should find the corresponding sequence $(r_1, ..., r_\ell) \in (\{0,1\}^k)^\ell$. This sequence is easy to find by using Observation 3. Specifically, the fact that, for every $x \in \{0,1\}^\ell$, the equality $\pi_{C(r_1),...,C(r_\ell)}(x) = Y_x$ imposes ℓ conditions on the bits of the $C(r_i)$'s, and combining all these conditions determines all the $C(r_i)$'s, which in turn determines all r_i's. (See details in the proof of Theorem 5.)

Error correction for M. The error-less regime provides a good warm-up for the following.

Theorem 5 (efficient error correction for the code M): *Suppose that there exists a polynomial-time algorithm for decoding C at an error rate of at most ϵ_C. Then, there exists a polynomial-time algorithm for decoding M at an error rate of at most ϵ_C.*

[4] One appealing way of doing this is by encoding the j^{th} bit of each of the r_i's in the j^{th} symbol of the Σ-sequence.

Proof: We consider the same conditions as in the warm-up (error-less) case. Intuitively, although the corrupted codeword (of M) may violate some of the conditions, it still holds that we get enough correct conditions for the values of the bits in each $C(r_i)$, and can overcome the incorrect ones by relying on the error-correction algorithm for C. Specifically, we will show that if the corrupted M-codeword is ϵ_C-close to the code M, then we can obtain the correct values of at least a $1-\epsilon_C$ fraction of the bits of $C(r_i)$, which allows us to recover $C(r_i)$ by error-correction. Hence, we provide a simple reduction from error correction of this permutation code to error correction of the associated binary code, without any loss in parameters. Details follow.

Rephrasing Observation 3, we observe that if $\pi_{w_1,...,w_\ell}(x) = y$, then, for every $i \in [\ell]$, the i^{th} bit of y equals the exclusive-or of the i^{th} bit of x and the bit in location $y'x''$ in w_i, where y' is the $(i-1)$-bit long prefix of y and x'' is the $(\ell-i)$-bit long suffix of x. Hence, for each $i \in [\ell]$, we get two equations for the value of each bit in w_i.

Now, suppose that the sequence $(Y_{0^\ell}, .., Y_{1^\ell})$ is ϵ_C-close to the M-codeword $\pi_{C(r_1),...,C(r_\ell)}$; that is, for at least a $1-\epsilon_C$ fraction of the x's it holds that $Y_x = \pi_{C(r_1),...,C(r_\ell)}(x)$. Then, given $(Y_{0^\ell}, .., Y_{1^\ell})$, we recover the r_i's in ℓ iterations. Specifically, in the i^{th} iteration we recover r_i and also correct the i^{th} bit in all Y_x's as follows.

1. For each $(x', x'') \in \{0,1\}^{i-1} \times \{0,1\}^{\ell-i}$, we have two equations for the value of location $y'x''$ in $C(r_i)$, where y' is the $(i-1)$-bit long prefix of $Y_{x'0x''}$ (which equals the $(i-1)$-bit long prefix of $Y_{x'1x''}$). One equation suggests that the value of this bit (of $C(r_i)$) equals the i^{th} bit of $x'0x'' \oplus Y_{x'0x''}$, and the other equation suggests that the value of this very bit equals the i^{th} bit of $x'1x'' \oplus Y_{x'1x''}$.

   We record both suggestions, and call them the 0-vote and the 1-vote for this bit of $C(r_i)$. Note that the b-vote is correct if the bit i^{th} bit of $Y_{x'bx''}$ is correct (i.e., equals the i^{th} bit of $\pi_{C(r_1),...,C(r_\ell)}(x'bx'')$). Since at most $\epsilon_C \cdot 2^\ell$ of the Y_x's are incorrect, there exists a $b \in \{0,1\}$ such that at most $\epsilon_C \cdot 2^{\ell-1}$ of the b-votes are incorrect.

2. For each $b \in \{0,1\}$, we try to decode $C(r_i)$ according to the b-votes for its bits. We stress that the correctness of the decoding can be verified; that is, the decoding is accepted if the corresponding C-codeword disagrees with at most an ϵ_C fraction of the b-votes. Hence, for at least one of the b's, we recover $C(r_i)$, which also yields r_i. At this point we correct the i^{th} bit of all Y_x's accordingly (i.e., set it to the i^{th} bit of $\pi_{C(r_1),...,C(r_\ell)}(x)$), and proceed to the next iteration.

Hence, if $(Y_{0^\ell}, .., Y_{1^\ell})$ is ϵ_C-close to the M-codeword $\pi_{C(r_1),...,C(r_\ell)}$, then we correctly recover all r_i's. The claim follows. ■

5.2 Constructing good constant composition codes

Recall that a constant composition code over an alphabet Σ is a code in which in each codeword each symbol of Σ appears the same number of times [4]; that is, if the codewords have length n, then each symbol appears $n/|\Sigma|$ times in each codeword. Permutation codes correspond to the special case in which $|\Sigma| = n$. Hence, the following result generalizes Theorem 2.

Theorem 6 (good constant composition codes): *For $\ell, \ell', k \in \mathbb{N}$ and $n = 2^\ell \geq 2^{\ell'}$, let $C : \{0,1\}^k \to \{0,1\}^{n/2}$ be any binary code, and $S'_n \subset [2^{\ell'}]^n$ denote the set of all n-long sequences in which each*

element appears $2^{\ell-\ell'}$ times. Then, there exists a constant composition code $C' : (\{0,1\}^k)^{\ell'} \to S'_n$ that uses C as a black-box such that evaluating C' (i.e., encoding) *reduces to ℓ' evaluations of C and the following properties hold:*

- *The rate of C' equals half the rate of C.*
- *The relative distance of C' is at least that of C.*
- *If C has a unique decoding from $\epsilon \cdot n$ errors, then so has C'. Furthermore, decoding C' reduces to ℓ' invocations of decoding algorithm for C.*

Furthermore, individual symbols in codewords of C' depend on ℓ' symbols in codewords of C, and can be computed in $\mathrm{poly}(\ell)$*-time.*

Proof Sketch: We modify the construction that underlies the proof of Theorem 2 by replacing permutation over $\{0,1\}^\ell$ that are indexed by ℓ-long sequences of codewords of C with functions from $\{0,1\}^\ell$ to $\{0,1\}^{\ell'}$ that are indexed by ℓ'-long sequences of codewords of C. Specifically, the generic permutation $\pi_{w_1,....,w_\ell} : \{0,1\}^\ell \to \{0,1\}^\ell$ defined in Eq. (2) is replace by the generic function $f_{w_1,....,w_{\ell'}} : \{0,1\}^\ell \to \{0,1\}^{\ell'}$ defined by

$$f_{w_1,....,w_{\ell'}}(x) \stackrel{\text{def}}{=} \mathtt{pref}_{\ell'}(\pi^{(\ell')}_{w_{\ell'}} \circ \cdots \circ \pi^{(1)}_{w_1}(x)), \tag{4}$$

where $\mathtt{pref}_{\ell'}(z)$ is the ℓ'-bit long prefix of z, and the $\pi^{(i)}_w$'s are as in Eq. (1). Indeed, $f_{w_1,....,w_{\ell'}}$ returns the ℓ'-bit long prefices of values of the permutation that is generated by the (partial) ℓ'-step walk on the (first ℓ' layers of the) Shuffle-Exchange network that is determined by the ℓ' instruction sequences $w_1, ..., w_{\ell'}$. Note that for every $y \in \{0,1\}^{\ell'}$ (and every $w_1, ..., w_{\ell'} \in \{0,1\}^{2^{\ell-1}}$) it holds that $|\{x \in \{0,1\}^\ell : f_{w_1,....,w_{\ell'}}(x) = y\}| = 2^{\ell-\ell'}$.

Letting $\Sigma = \{0,1\}^{\ell'}$, the code $C' : (\{0,1\}^k)^{\ell'} \to S'_n \subset \Sigma^{2^\ell}$ is defined such that it maps $\overline{r} = (r_1, ..., r_{\ell'}) \in \{0,1\}^{\ell' \cdot k}$ to $(f_{\overline{C}(\overline{r})}(0^\ell), ..., \pi_{\overline{C}(\overline{r})}(1^\ell)) \in \Sigma^{2^\ell}$, where $\overline{C}(\overline{r}) = (C(r_1), ..., C(r_{\ell'}))$. Recalling that the locations in the codewords of C' are associated with ℓ-bit strings, the symbol in location x of the codeword $C'(\overline{r})$ is $f_{\overline{C}(\overline{r})}(x)$; that is, $C'(\overline{r})_x = f_{\overline{C}(\overline{r})}(x)$.

We first note that C' is a constant composition code (since for every $y \in \{0,1\}^{\ell'}$ and $\overline{r} \in \{0,1\}^{\ell' \cdot k}$, it holds that $|\{x \in \{0,1\}^\ell : f_{\overline{C}(\overline{r})}(x) = y\}| = 2^{\ell-\ell'}$). The rate of C' is $\frac{\ell' \cdot k}{2^\ell \cdot \ell'}$, which is half the rate of C, and the distance of C' is established exactly as in the proof of Theorem 4, where the crucial observation that a difference created by the i^{th} step appears in the i^{th} bit.[5] Analogous considerations apply to the proof of Theorem 5, and this establishes the error correction feature of C'. ■

Acknowledgements

We are grateful to Venkatesan Guruswami for extremely useful discussions regarding Section 5. In particular, Theorem 6 answers a question that he suggested.

[5] For distinct $(r_1, ..., r_{\ell'}), (s_1, ..., s_{\ell'}) \in \{0,1\}^{\ell' \cdot k}$, let $i \in [\ell']$ be the smallest index such that $r_i \neq s_i$. Then, the permutations $\pi^{(i)}_{C(r_i)} \circ \cdots \circ \pi^{(1)}_{C(r_1)}$ and $\pi^{(i)}_{C(s_i)} \circ \cdots \circ \pi^{(1)}_{C(s_1)}$ differ on $\delta_C \cdot 2^\ell$ values, where δ_C is the relative distance of C, and these values differ on their i^{th} bits. Furthermore, subsequent steps do not affect these bits.

References

[1] V. Benes. Optimal Rearrangeable Multistage Connecting Networks. *Bell System Technical Journal*, Vol. 43 (4), pages 1646–1656, 1964.

[2] P.J. Cameron. Permutation Codes. *European Journal of Combinatorics*, Vol. 31 (2), pages 482–490, 2010.

[3] Y.M. Chee and P. Purkayastha. Efficient decoding of permutation codes obtained from distance preserving maps. In *2012 IEEE International Symposium on Information Theory*, pages 636-640, 2012.

[4] W. Chu, C.J. Colbourn, P. Dukes. On Constant Composition Codes. *Discrete Applied Mathematics*, Vol. 154 (6), pages 912–929, 2006.

[5] O. Goldreich and A. Wigderson. Robustly Self-Ordered Graphs: Constructions and Applications to Property Testing. *ECCC*, TR20-149, September 2020.

[6] F.H. Hunt, S. Perkins, and D.S. Smith. Decoding Mixed Errors and Erasures in Permutation Codes. *Designs, Codes and Cryptography*, Vol. 74, pages 481–493, 2015.

[7] D. Slepian. Permutation Modulation. *Proceeding of the IEEE*, Vol. 53, pages 228–236, 1965.

[8] L.G. Valiant. A Scheme for Fast Parallel Communication. *SIAM Journal on Computing*, Vol. 11 (2), pages 350–361, 1982.

[9] L.G. Valiant and G.J. Brebner. Universal Schemes for Parallel Communication. In *13th ACM Symposium on the Theory of Computing*, pages 263–277, 1981.