

MIT Open Access Articles

Coding for locality in reconstructing permutations

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

Citation: Raviv, Netanel, et al. "Coding for Locality in Reconstructing Permutations." Designs, Codes and Cryptography, vol. 86, no. 2, Feb. 2018, pp. 387–418.

As Published: <http://dx.doi.org/10.1007/s10623-017-0378-9>

Publisher: Springer US

Persistent URL: <http://hdl.handle.net/1721.1/114517>

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike



Coding for Locality in Reconstructing Permutations

Netanel Raviv · Eitan Yaakobi ·
Muriel Médard

Received: date / Accepted: date

Abstract The problem of storing permutations in a distributed manner arises in several common scenarios, such as efficient updates of a large, encrypted, or compressed data set. This problem may be addressed in either a combinatorial or a coding approach. The former approach boils down to presenting large sets of permutations with *locality*, that is, any symbol of the permutation can be computed from a small set of other symbols. In the latter approach, a permutation may be coded in order to achieve locality. Both approaches must present low *query complexity* to allow the user to find an element efficiently. We discuss both approaches, and give a particular focus to the combinatorial one.

In the combinatorial approach, we provide upper and lower bounds for the maximal size of a set of permutations with locality, and provide several simple constructions which attain the upper bound. In cases where the upper bound is not attained, we provide alternative constructions using a variety of tools, such as Reed-Solomon codes, permutation polynomials, and multi-permutations. In addition, several low-rate constructions of particular interest are discussed.

In the coding approach we discuss an alternative representation of permutations, present a paradigm for supporting arbitrary powers of the stored permutation, and conclude with a proof of concept that permutations may be stored more efficiently than ordinary strings over the same alphabet.

Keywords Permutations · Locally Recoverable Codes · Distributed Storage

PACS 05A05 · 68R05

N. Raviv, E. Yaakobi
Department of Computer Science, Technion, Haifa 3200003, Israel, E-mail:
{netanel.raviv,yaakobi}@gmail.com.

M. Médard
Research Lab. of Electronics, Massachusetts Institute of Technology, Cambridge, MA,
E-mail: medard@mit.edu.

Note to Reviewers:

Parts of this work appeared in the International Symposium on Information Theory in Barcelona, Spain, July 2016 [23]. An online version of the conference paper is available at

<http://ieeexplore.ieee.org/document/7541339/>.

In the conference version, the upper bounds of Section 3.1 are given without proofs, some of the lower bounds from Section 3.2 are given, together with most of the high-rate constructions of Section 4.

The remaining parts of this paper, including additional motivation, full proofs of all bounds, additional lower bound, full proofs and additional examples of all high rate constructions, are new. In addition, Section 5 and Section 6 are completely new, and do not appear in the conference version.

1 Introduction

For an integer n , let S_n be the group of all permutations on n elements. Given a permutation $\pi \in S_n$ we consider the problem of storing a representation of π in a distributed system of storage nodes. This problem arises when considering efficient *permutation updates* to a distributed storage system. For example, in a system which stores large entries whose order commonly changes, one might prefer to store the permutation of the entries, rather than constantly shift them around. Alternatively, the stored file may be *signed* or *hashed* (using cryptographic primitives), and storing the permutation alongside the file allows to update the file without altering its signature. The given file may also be compressed using a *source code*, and storing the permutation enables to perform permutation updates without the need to decompress the file. Perhaps the most natural example for such a scenario is the common operation of *cut and paste*, which may be modelled as a permutation update, and briefly discussed in Subsection 1.1.

Above all questions of efficiency, in very simple storage schemes, a permutation update might require to decode the file (see Subsection 1.1). Storing the permutation alongside x allows a permutation update to be made without decoding the file, at the price of storage overhead.

The crux of enabling efficient storage lies in the notion of *locality*, that is, any failed storage node may be reconstructed by accessing a small number of its neighbors. The corresponding coding problem is often referred to as *symbol locality*, in which every symbol of a codeword is a function of a small set of other symbols [26]. Another approach towards efficient storage stems from *array codes*, in which each storage node stores a large set of symbols from the codeword (e.g., [28] and references therein). For simplicity, in this paper we consider symbol locality. Furthermore, since our underlying motivation is allowing *small* updates to be done efficiently, we disregard the notion

of minimum distance between the stored permutations, and focus solely on locality. Occasionally, we will discuss local correction of simultaneous erasures, as in [22].

As mentioned earlier, locality in permutations may be considered in either a combinatorial or a coding approach. Under the combinatorial one, the underlying motivation is set aside, and the problem boils down to finding (or bounding the maximum size of) sets of permutations which present locality; i.e., such that any symbol in any permutation in this set may be computed from small number of other symbols. This approach is the main one in this paper. Under the coding approach, the given permutation may be coded in order to achieve locality, e.g., by using a *locally recoverable code* (LRC). The combinatorial approach clearly outperforms the use of LRCs in terms of redundancy (see Section 2), at the price of not being able to store any permutation. Furthermore, it is also shown in Section 2 that storing a subset of S_n using an LRC while maintaining the same overhead as in the combinatorial approach does not enable an instant access to the elements of the permutation, as discussed further in this section.

The combinatorial approach may also be applied in rank modulation coding for flash memories [16], in which each flash cell contains an electric charge, and a block of cells contains the permutation which is induced by the charge levels. Flash memories are susceptible to various types of hardware failures, some of which result in a complete loss of the charge in a cell, i.e., an erasure [15]. A rank modulation code which enables local erasure correction allows quick recovery from a such loss of charge. Yet, this application requires some further adjustments of our techniques, since the charge levels usually represent *relative* values rather than *absolute* ones.

Several natural questions, which are irrelevant in ordinary storage, may arise when discussing storage of permutations. For example, a storage system which stores $\pi \in S_n$ may be required to answer either $\pi^{-1}(i) = ?$ or $\pi(i) = ?$ quickly. These questions are denoted by Q1 and Q2, respectively, and notice that without this additional requirement, storing permutations reduces to storing binary strings of length $\lceil \log(n!) \rceil$ by enumerative encoding. In the combinatorial approach, either one of Q1 or Q2 becomes trivial, depending if we consider the permutation at hand as $(\pi(1), \dots, \pi(n))$ or $(\pi^{-1}(1), \dots, \pi^{-1}(n))$. For example, when storing the latter, answering Q1 is straightforward, and answering Q2 is possible by inspecting $\pi^{-1}(i), \pi^{-1}(\pi^{-1}(i)), \dots$, etc., until i is found (see [12, ch. 1.3, p. 29]). Hence, the number of required queries for Q1 is 1 (or $\log n$ bits), and for Q2 it is at most the length of the longest cycle in π . Although it is not the general purpose of this research, we take initial steps towards efficient retrieval of $\pi(i)$ and $\pi^{-1}(i)$ simultaneously. Clearly, allowing the permutation to be encoded provides more freedom in devising storage techniques. However, maintaining a concise representation which enables Q1 and Q2 to be answered quickly is a rather involved question, which was studied in the past in a non-distributed setting (e.g. [20, 21], see further details in Section 2).

Since a variety of mathematical techniques are used throughout this paper, in each technique we consider the permutations in S_n as operating on a different sets of symbols. These sets may be either $[n] \triangleq \{1, \dots, n\}$ or $\{0, \dots, n-1\}$. Alternatively, we may assume that n is a power of prime, and $\{0, 1, \dots, n-1\}$ is an enumeration of the elements in \mathbb{F}_n , the finite field with n elements, where the additive identity element of \mathbb{F}_n is denoted by “0” and the multiplicative identity element is denoted by “1”. Unless otherwise stated, we consider permutations in the *one line representation* (one-liner, in short), that is, $\pi \triangleq (\pi_1, \dots, \pi_n) = (\pi^{-1}(1), \dots, \pi^{-1}(n))$. A well known tool in the analysis of permutations, that will be used several times in the sequel, in the *disjoint cycle representation* [7, Sec. 3.1]. In this regard, for distinct integers $i_0, \dots, i_{s-1} \in [n]$, a *cycle* $(i_0 \ i_1 \ \dots \ i_{s-1})$ represents the permutation π such that $\pi(i_j) = i_{j+1 \bmod s}$ for all $j \in \{0, \dots, s-1\}$, and $\pi(k) = k$ for all $k \in [n] \setminus \{i_0, \dots, i_{s-1}\}$. The *product* between cycles is defined naturally as the composition of the respective cycles. It is widely known that any permutation can be represented as a product of disjoint cycles, and this representation is unique up to the order of the cycles, and up to cyclic rotations of each cycle.

Given a set $S \subseteq S_n$, we say that S has locality d if for any $\pi \in S$, any symbol π_i may be computed from d other symbols of π . In order to formally define this notion, we follow an outline similar to [26, Sec. II]. For a set of permutations $S \subseteq S_n$, an integer $i \in [n]$, and a symbol $a \in [n]$, let $S(i, a) \triangleq \{\pi \in S \mid \pi_i = a\}$. For a set of integers $I \subseteq [n]$ let S_I be the restriction of the permutations in S to the index set I , i.e., the set which results from deleting all entries indexed by $[n] \setminus I$ for any $\pi \in S$.

Definition 1 For positive integer d and n such that $d < n$, a set of permutations $S \subseteq S_n$ is said to have locality d if for every $i \in [n]$, there exists a subset $I_i \subseteq [n] \setminus \{i\}$, $|I_i| \leq d$, such that $S(i, a)_{I_i} \cap S(i, a')_{I_i} = \emptyset$ for any distinct a and a' in $[n]$.

The *rate* of S is defined as $\log |S| / \log(n!)$, the identity permutation is denoted by Id , and \circ denotes the concatenation of sequences.

This paper is organized as follows. Additional motivation for studying storage of permutations is given in Subsection 1.1. Section 2 summarizes related previous work. Section 3 discusses upper and (existential) lower bounds on the maximal possible size of subsets of S_n which present locality. Section 4 provides several simple constructions, some of which attain the upper bound which is presented in Section 3. A construction which shows a connection to Reed-Solomon codes via permutation polynomials, and a construction via multi-permutations are also given in Section 4. In Section 5 two sets of permutations of particular interest are presented. These sets have *low* rate, and low locality. The set from Subsection 5.1 will be later shown to have a more efficient representation (Subsection 6.1). The coding approach is discussed in Section 6, in which the main result is a technique which allows to compute every *power* of the stored permutation very efficiently, and is strongly based on [20]. Subsection 6.3 shows a preliminary proof of concept that permutations may be stored using less redundancy bits than ordinary strings, and shows a

concrete technique of doing so, attaining a negligible advantage. Concluding remarks and problems for future research are given in Section 7.

1.1 Motivation

This subsection presents a general motivation for distributed storage of permutations through common file updates, and through applications in cryptography. We begin by showing that in a certain simple scenario, permuting the coordinates of the stored file without decoding is impossible.

Assume that a file x , which contains n entries, is divided into two halves x_1, x_2 and stored in three nodes using the simplest parity check code $x_1, x_2, x_1 + x_2$ (as in the RAID4 storage system). In addition, assume that the user would like to apply $\pi \in S_{n/2}$ on x_1 . Applying π only over the systematic part, that is, update the system to contain $\pi(x_1), x_2, x_1 + x_2$, clearly does not maintain the error correction capability. On the other hand, updating the parity node to contain $\pi(x_1) + x_2$ without knowing either x_1 or x_2 is information theoretically impossible. Hence, given π , the storage node which contains $x_1 + x_2$ cannot update its own content to $\pi(x_1) + x_2$. This fact is illustrated in the following lemma.

Lemma 1 *If x_1 and x_2 are strings of length $n/2$ over a field \mathbb{F}_q , then given a nontrivial $\pi \in S_{n/2}$ and $x_1 + x_2$, it is information theoretically impossible to compute $\pi(x_1) + x_2$.*

Proof It is widely known that given $x_1 + x_2$, one cannot infer any information on either of x_1 and x_2 ¹. We show that if $\pi(x_1) + x_2$ may be computed, then some information about certain symbols of x_1 can be inferred.

Knowing $x_1 + x_2$ and $\pi(x_1) + x_2$, we may calculate $\alpha \triangleq \pi(x_1) + x_2 - (x_1 + x_2) = \pi(x_1) - x_1$. Furthermore, since $\pi \in S_{n/2}$ is nontrivial, we may arbitrarily choose one cycle (i_1, i_2, \dots, i_t) from the disjoint cycle representation of π , for some $t > 1$. Hence, we may assemble the following linear system of equations, in the variables $x_1^{i_1}, x_1^{i_2}, \dots, x_1^{i_t}$.

$$\begin{pmatrix} -1 & & & 1 \\ 1 & -1 & & 0 \\ 0 & 1 & -1 & 0 \\ & & \ddots & \vdots \\ 0 & & & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1^{i_1} \\ x_1^{i_2} \\ \vdots \\ x_1^{i_t} \end{pmatrix} = \begin{pmatrix} \alpha_{i_1} \\ \alpha_{i_2} \\ \vdots \\ \alpha_{i_t} \end{pmatrix} \quad (1)$$

Since the matrix in (1) has rank $t - 1 > 0$, and since the system *has* a solution, we have that the affine space of solutions of this system is of dimension 1. Therefore, the sequence $x_1^{i_1}, \dots, x_1^{i_t}$ may have either one of q possible values, rather than q^t values. Since $t > 1$, the claim follows. \square

¹ In other words, if X_1 and X_2 are two uniform random variables that take values on $\mathbb{F}_q^{n/2}$, then the *mutual information* $I(X_1; X_1 + X_2)$ is zero.

Permutation updates arise in everyday scenarios. Two very common updates may be modeled as a special case of permutation updates. One is the well-known *cut-paste* operation, in which a portion of data is removed and placed elsewhere in the file. The other one is the *replacement* operation, in which two distinct portions of equal size switch places. The following definitions formalize these common notions.

Definition 2 A cut-paste update is a permutation update whose corresponding permutation is either

$$\pi = (1, \dots, t-1, t+\alpha, \dots, s, t, t+1, \dots, t+\alpha-1, s+1, \dots, n), \text{ or}$$

$$\pi = (1, \dots, t-1, s-\alpha+1, \dots, s-1, s, t, t+1, \dots, s-\alpha),$$

for some t and s in $[n]$ such that $t < s$, and for some $\alpha \leq s - t$.

It should be noted that for $\alpha = 1$ and $s \neq t$, Definition 2 describes a set of permutations called *translocations*. Translocations are the building blocks of the Ulam metric [14, Proposition 3], which is an essential tool for error-correction in rank modulation for flash memories [14] and DNA research [5, 11].

Definition 3 A replacement update is a permutation update whose corresponding permutation is

$$\pi = (1, \dots, t-1, s, s+1, \dots, s+\alpha-1, t+\alpha, \dots, s-1, t, t+1, \dots, t+\alpha-1, s+\alpha, \dots, n),$$

for some t and s in $[n]$ such that $t < s$, and for some α such that $\alpha < \min\{s-t, n-s+1\}$.

Transpositions can be seen as the building blocks of the transposition metric, considered in [17]. The number of cut-paste and replacement operations is rather small, as shown in the following lemma, which is easy to prove.

Lemma 2 *The number of replacement updates on a file of n elements is $O(n^3)$, and the number of cut-paste update on a file of n elements is $O(n^3)$.*

Clearly, storing one of $O(n^3)$ possible values requires $O(\log n)$ bits. Therefore, providing an efficient answer to Q1 and Q2 becomes trivial, since obtaining *all* information about one of these permutations is possible by reading $O(\log n)$ bits. Thus, these specific permutations, however common, will not be discussed further in this paper. Yet, other interesting types of permutations arise in daily scenarios, see Section 5.

Similar questions were studied from a cryptographic perspective. The works of [3, 4] initiated the research of cryptographic primitives, such as hash functions or signature schemes, that enable efficient updates. That is, small changes in the file may be incorporated into its hash (or signature) efficiently, without requiring to recompute it from scratch. The updates considered in [3, 4] are of different nature (replacement rather than permutations), and the file is not

stored in a distributed manner, yet the underlying motivation is highly similar, that is, how to perform efficient updates to a stored file, without the need to encode it anew.

Our work may be used in conjunction with any existing (distributed) cryptosystem, which was not necessarily meant for updates, while enabling permutation updates to be done efficiently. Furthermore, in coding-based cryptosystems (such as the McEliece cryptosystem [18]) any update of the un-encrypted file is inefficient. This is since any change of a symbol in the un-encrypted file requires changing at least as many symbols of its encrypted counterpart as the minimum distance of the underlying code.

2 Previous Work

Coding over S_n , endowed with either of several possible metrics [11], was extensively studied under many different motivations. For example, codes in S_n under the Kendall's τ metric [2] and the infinity metric [27] were shown to be useful for non-volatile memories, and codes under the Hamming metric (also known as permutation arrays) were shown to be useful for power-line communication [9]. In all of these works, the permutations are encodings of messages, and hence should maintain minimum distance constraints. In this work, however, the permutation itself is of interest, thus minimum distance is not considered, and certain sets of permutations with low rate are also of interest. In addition, when taking the coding approach, the added redundancy need not to comply with any combinatorial constraints, e.g., to result in a permutation of a larger base set.

As mentioned in the Introduction, we consider permutations in their one line representation (one-liner, in short). Our problem may be seen as allowing local *erasure* correction of permutations in the one-liner. Erasure and deletion correction of permutation codes was discussed in [15], in which it was shown that the most suitable metric for erasure correction (called “stable erasure” in [15]) is the Hamming metric, that measures the number of entries in which the one-liners differ. However, the work of [15] was motivated by the rank modulation scheme in flash memories and thus locality was not discussed.

Furthermore, it is obvious that a permutation array with minimum Hamming distance $n - d + 1$ allows local erasure correction of any symbol from any d other symbols. However, constructing permutation arrays with minimum Hamming distance is an infamously hard problem, let alone in the high distance region [6]. Moreover, construction of permutation arrays with minimum Hamming distance is *not* equivalent to finding sets of permutations with locality, since the inverse is clearly untrue, that is, a set with locality d does not imply a permutation array with minimum Hamming distance $n - d + 1$.

A similar motivation lies behind the work of [24], where the authors considered updates of a distributed storage system which involve *deletions* and *insertions* to a file which is stored in a distributed system. Clearly, a permutation update can be seen as a series of deletions and insertions. The so-called

“scheme P” [24, Section 4.2] provides a framework for maintaining a file $x \in \mathbb{F}_q^n$ in a distributed storage system under insertions and deletions. The entire file is assumed to be stored using an arbitrary array-code, and the deletions and insertions are taking place with respect to any specific block. Interestingly, a deletion is treated as a permutation, where the deleted symbol is replaced with the symbol 0 and pushed to the end of the block. A set of permutation matrices $\{A^{(i)}\}$, one for each block, is stored in the system to keep track of the permuted symbols. An insertion is treated similarly, keeping track of the location of insertion using the permutation matrices. The overhead of storing the matrices $\{A^{(i)}\}_{i \in [n]}$ is improved by using the fact that the entire matrices need not to be stored, and we may settle for the *locations* of the edits. Our work may be seen as an extension of scheme P from [24] to *permutation* updates, as we handle various types of larger sets of permutations.

Recall that we are interested in supporting the queries Q1 or Q2. A similar problem, which relates to general strings rather than to permutations, was addressed in the past as a problem about data structures. A representation technique called “the succincter”, which enables one-symbol recovery², was discussed in [21]. This representation requires only slightly more bits than the optimal one, but it seems to be superfluous when discussing large alphabets. Formally, according to [21, Theorem 1], for any t , a file x of length n over an alphabet Σ can be represented by $O(|\Sigma| \log n) + f(n, x, t)$ for some function f , while allowing one-symbol recovery in $O(t)$ time. In our scenario we have that $\Sigma = [n]$, and optimal one-symbol recovery is trivially possible by using an $n \log n$ bit representation.

When considering the coding approach, a standard technique may be the use of *Locally Recoverable Codes* (LRCs). An (m, k, d) LRC is a code that produces an m -symbol codeword from a k -symbol message, such that any symbol of the produced codeword may be recovered by contacting at most d other symbols. LRCs have been subject to extensive research in recent years [26], mainly due to their application in distributed storage systems. Consider any permutation $\pi \in S_n$ as a string over the alphabet³ $[n]$, and encode it to m symbols using an optimal *systematic* LRC (e.g., [26]). Singleton-optimal LRCs that encode $n = k$ symbols to m symbols and admit locality of d must satisfy $q \geq m$, and [26, Theorem 2.1]

$$\frac{n}{m} \leq \frac{d}{d+1}, \quad (2)$$

i.e., their *rate* is bounded from above by $d/(d+1)$. Thus, n/d redundant information symbols are required to achieve locality of d . Using the combinatorial approach (and some of the techniques in the coding approach, see Section 6.1) we achieve smaller storage overhead, in the price of not being able to store any permutation. Notice that by restricting our attention to a subset $S \subseteq S_n$

² One-symbol recovery is the term used in [21] for returning a given entry of the stored string, without requiring to decode it.

³ More precisely, the alphabet $[n]$ when seen as a subset of a large enough finite field \mathbb{F}_q , over whom the construction of the LRC is possible.

it is possible to obtain locality of d (assuming each storage node may contain $\log n$ bits) by using LRCs with $\log |S| \cdot \frac{d+1}{d}$ bits of storage. This amount of storage outperforms the combinatorial approach if $\log |S| \cdot \frac{d+1}{d} \leq n \log n$, that is, the rate of S is at most $\frac{d}{d+1}$. It will be shown in the sequel (Theorem 3 and Lemma 5) that sets of permutations of this rate and locality d do exist. Moreover, the use of LRC for storing the subset S with this much overhead seems to require enumerative decoding of S , a procedure that eradicates the ability for quick answer to Q1 and Q2. Therefore, there exist scenarios in which the combinatorial approach outperforms the coding one, both in terms of redundancy and in terms of instant access.

In addition, simple LRCs will be used for proof of existence of optimal sets of permutations with locality (i.e., having maximum possible rate). In particular, in Subsection 3.2 it will be shown that there exists a *coset* of an optimal locally recoverable code C , which contains a set S of words that can be considered as permutations. However, this claim is merely existential, and does not provide any significant insights on the structure of S .

3 Bounds

Let $A(n, d)$ be the maximum size of a subset of S_n with locality d . This section presents an upper bound and an existential lower bound on $A(n, d)$. The upper bound in Subsection 3.1 is an adaptation of a bound for LRCs ([26, Theorem 2.1], given in (2)). This upper bound is later improved for $d = 1$, and is attained by a certain construction in Section 4.2 to follow. To obtain an existential lower bound, in Subsection 3.2 it is shown that a certain LRC has a coset which contains a set of permutations with the same locality as the LRC itself. For certain small values of locality, an equivalent lower bound will also be derived in Subsection 3.2 by a connection to a classical problem in chess.

Notice that in this section the combinatorial approach is considered. This clearly does not fully reflect the entire spectrum of techniques that might be used to store permutations. Nevertheless, it presents the limitation of a certain approach towards storage of permutations, which is the main one in this paper.

3.1 Upper Bounds

The bound for LRCs (2) can be used as-is if n is a power of prime, and the set of permutations is considered as a non-linear code in \mathbb{F}_n^n . By a simple adaptation of [26, Theorem 2.1] to non-linear codes, we have that a non-linear code in \mathbb{F}_n^n with locality d contains at most $n^{\lfloor dn/(d+1) \rfloor}$ codewords. This bound may be improved by utilizing the combinatorial structure of permutations.

The following bound, as the one given in (2), assumes a *non-adaptive* decoder. That is, it is assumed that for a given erased entry i , the decoder accesses a set I_i of d entries, and receives all their content at once. A different

approach, which is partially implemented in Section 5.2, is to access the non-erased symbols in an adaptive manner, where the locations of the latter ones depend on the content of the former ones. The following lemma, due to [26], is a variant of a classic result by [1].

Lemma 3 [26, Theorem A.1] *If G is a directed graph on n vertices then there exists an induced directed acyclic subgraph of G on at least*

$$\frac{n}{1 + \frac{1}{n} \sum_i d_i^{\text{out}}}$$

vertices, where d_i^{out} is the outgoing degree of vertex i .

This lemma provides the following adaptation of [26, Theorem 2.1] to permutations.

Theorem 1 $A(n, d) \leq \frac{n!}{\left\lceil \frac{n}{d+1} \right\rceil!}$.

Proof Let $C \subseteq S_n$ be a set of permutations with locality d , and let G be a directed graph whose vertex set is $[n]$, and (i, j) is an edge if entry j in $\pi \in C$ is required for the local correction of entry i . Notice that since C has locality d , Lemma 3 implies that G has an induced directed acyclic subgraph on a set U of at least $\left\lceil \frac{n}{d+1} \right\rceil$ vertices. Since this subgraph is acyclic, it contains a vertex i with no outgoing edges. Hence, entry i is a function of entries in $[n] \setminus U$. Repeating this argument for the induced graph on $U \setminus \{i\}$, we have that there exists a vertex i' with no outgoing edges to $U \setminus \{i\}$. Hence, entry i' is a function of entries in $[n] \setminus (U \setminus \{i\})$. Since entry i is a function of entries in $[n] \setminus U$, we have that i' is also a function of entries in $[n] \setminus U$. Iterating over all vertices in U , we have that there are at least $\left\lceil \frac{n}{d+1} \right\rceil$ entries that depend on the other $\left\lfloor \frac{dn}{d+1} \right\rfloor$ entries.

Therefore, there exists a set of at most $\left\lfloor \frac{dn}{d+1} \right\rfloor$ entries, which determines the entire permutation. There are $\binom{n}{\left\lfloor \frac{dn}{d+1} \right\rfloor}$ different ways to choose the elements in these entries, and $\left\lfloor \frac{dn}{d+1} \right\rfloor!$ ways to permute them. Therefore, the size of C is at most

$$\begin{aligned} \binom{n}{\left\lfloor \frac{dn}{d+1} \right\rfloor} \cdot \left\lfloor \frac{dn}{d+1} \right\rfloor! &= \frac{n!}{\left\lfloor \frac{dn}{d+1} \right\rfloor! \cdot \left(n - \left\lfloor \frac{dn}{d+1} \right\rfloor\right)!} \cdot \left\lfloor \frac{dn}{d+1} \right\rfloor! \\ &= \frac{n!}{\left(n - \left\lfloor \frac{dn}{d+1} \right\rfloor\right)!} = \frac{n!}{\left\lceil \frac{n}{d+1} \right\rceil!} \end{aligned}$$

□

As a simple corollary of Theorem 1 we obtain an upper bound on the rate of a set of permutations with locality d . Notice that by the Stirling approximation

of the factorial function, we have that $\log(n!) \approx n \log n$, and hence the optimal rate implied by Theorem 1 is

$$\begin{aligned} \frac{\log\left(\left\lceil \frac{n!}{\left\lceil \frac{n}{d+1} \right\rceil!} \right\rceil\right)}{\log(n!)} &= 1 - \frac{\log\left(\left\lceil \frac{n}{d+1} \right\rceil!\right)}{\log(n!)} \\ &\xrightarrow{n \rightarrow \infty} 1 - \frac{\frac{n}{d+1} \cdot \log\left(\frac{n}{d+1}\right)}{n \log n} \\ &= \frac{d}{d+1} + \frac{\log(d+1)}{n \log n(d+1)} \xrightarrow{n \rightarrow \infty} \frac{d}{d+1}. \end{aligned} \quad (3)$$

Hence, in the sequel a set C of permutations with locality d is called *optimal* if its rate is $\frac{d}{d+1}$, and *asymptotically optimal* if its rate tends to $\frac{d}{d+1}$ as n tends to infinity.

The trivial subset $C = S_n$ admits locality of $d = n - 1$, and attains the upper bound. In addition, the *alternating group*, and its complement, have locality of $n - 2$ (see Subsection 4.1). According to these examples, it is clear that $A(n, n - 1) = n!$ and $A(n, n - 2) = n!/2$, and hence, any upper bound will coincide with the one given in Theorem 1 for $d \in \{n - 1, n - 2\}$.

For $d < n - 2$ there exists a large gap between this bound and the sizes of the sets presented in this paper. This gap may be resolved for $d = 1$ by using a graph theoretic argument on the dependency graph in the proof of Theorem 1. In what follows, we say that a directed graph G is connected if removing the directions from the edges of G yields an undirected connected graph T . If T is not connected, then every connected component of T is considered as a connected component of G .

Lemma 4 *If G is a directed graph of constant out-degree one, then any connected component of G contains precisely one cycle.*

Proof Let G' be a connected component of G . If G' contains no vertex with in-degree zero, then it is a cycle, and the claim is clear. Else, let v_1 be a node in G' with in-degree zero. Since G has constant out-degree one, it follows that there exists a unique path $v_1 \rightarrow \dots \rightarrow v_t$, such that all vertices are distinct, and t is maximal. Since v_t has out-degree one, it follows that G' contains a cycle.

If G' contains no other nodes besides v_1, \dots, v_t , then it contains precisely one cycle, and the claim follows. If G' contains no other node with in-degree zero, we have that G' contains two connected components, a contradiction. Hence, let v_{t+1} be another node in G of in-degree zero. Similarly, G also contains a path $v_{t+1} \rightarrow \dots \rightarrow v_s$ with distinct nodes and a maximal s such that $s \notin \{v_1, \dots, v_t\}$. The node v_s has out-degree one, and hence must be connected to another node u in G' . If $u \in \{v_{t+1}, \dots, v_s\}$, we have that G' contains two connected components, a contradiction. Therefore, the path v_{t+1}, \dots, v_s is connected to one of the nodes v_1, \dots, v_t , and thus no additional cycle is possible. By iterating this argument until all vertices of G' are traversed, we have that G' contains precisely one cycle. \square

As a result, we obtain the following bound on the maximal size of sets of permutations with locality one.

Theorem 2 $A(n, 1) \leq n!! \triangleq \prod_{i=0}^{\lceil n/2 \rceil - 1} (n - 2i)$.

Proof Let $C \subseteq S_n$ be a set with locality one, and let G be a directed graph whose vertex set is the set of entries $[n]$, and (i, j) is an edge if entry j in $\pi \in C$ is required for the local correction of entry i . Since G has locality one, it follows that any vertex in G has out-degree one. Furthermore, for any edge (i, j) , fixing value of π_j determines the value of π_i .

Let G_1, \dots, G_t be the connected components of G . According to Lemma 4, each G_i contains precisely one cycle. Clearly, fixing the value of any entry in such a cycle, determines the value of all other entries in its connected component. Hence, the entire permutation $\pi \in C$ is determined by fixing the value of t of its entries, one entry in the unique cycle in each connected component. Since each connected component contains at least two nodes, we have that the maximum size of C is

$$n \cdot (n - 2) \cdot (n - 4) \cdot \dots \cdot \left(n - 2 \left(\left\lceil \frac{n}{2} \right\rceil - 1 \right) \right).$$

□

Since the set constructed in Section 4.2 below attains the bound of Theorem 2 for $d = 1$, we have that $A(n, 1) = n!!$. Generalizing the techniques in the proof of Theorem 2 to any locality seems to be related to extinction problems in cellular automata on graphs [29], and might improve the bound given in Theorem 1 for many special cases.

3.2 Lower Bound

A locally recoverable code C of optimal rate, length n , and locality d , may easily be constructed over $\mathbb{Z}_n \triangleq \{0, 1, \dots, n - 1\}$, the set of integers modulo n with the respective modular addition operation. This is done by adding $n/(d + 1)$ “parity checks” to all disjoint sets of d consecutive symbols in $\mathbb{Z}_n^{n - n/(d + 1)}$.

Example 1 For $n = 6$ and $d = 2$ the set

$$\{(a, b, c, d, a + b, c + d) \mid a, b, c, d \in \mathbb{Z}_6\}$$

is an LRC of optimal rate $\frac{2}{3}$ over \mathbb{Z}_6 .

The rate of C attains the upper bound of $\frac{n - n/(d + 1)}{n} = \frac{d}{d + 1}$, given in (2). Also, it is readily verified that C is closed under component-wise addition, and hence it is a subgroup of the additive group \mathbb{Z}_n^n . Therefore, cosets of C may be defined, and note that each such coset has the same locality d as the code C .

Theorem 3 $A(n, d) \geq n! / n^{n/(d + 1)}$.

Proof Let C be an LRC of optimal rate, length n , and locality d over \mathbb{Z}_n . Since C is a subgroup of the additive group \mathbb{Z}_n^n , it has $\frac{n^n}{n^{dn/(d+1)}} = n^{n/(d+1)}$ cosets, and each of which has locality d as well. Since $n!$ of the words in \mathbb{Z}_n^n are permutations, it follows by the pigeonhole principle that one of the cosets of C contains a set of at least $n!/n^{n/(d+1)}$ permutations. \square

The rate which is implied by Theorem 3 asymptotically attains the rate of the upper bound from Theorem 1, given in (3), since

$$\frac{\log\left(\frac{n!}{n^{\frac{n}{d+1}}}\right)}{\log(n!)} = 1 - \frac{\frac{n}{d+1} \cdot \log n}{\log(n!)} \xrightarrow{n \rightarrow \infty} \frac{d}{d+1}.$$

On the other hand, Theorem 3 provides a set with higher redundancy than the potential upper bound, where the redundancy of a set S is defined as $\log(n!) - \log(|S|)$. Using the same techniques as in (3), we have that the redundancy of the set from Theorem 3 is

$$\log(n!) - \log\left(\frac{n!}{n^{n/(d+1)}}\right) \approx \frac{n}{d+1} \log n,$$

where the potential redundancy implied by Theorem 1 is

$$\log(n!) - \log\left(\frac{n!}{\lceil \frac{n}{d+1} \rceil}\right) \approx \frac{n}{d+1} \log \frac{n}{d+1}.$$

Therefore, it may be possible to achieve sets with redundancy up to $n \cdot \frac{\log(d+1)}{d+1}$ bits *smaller* than the one obtained in Theorem 3.

Remark 1 The proof of Theorem 3 relies on a simple construction of C , an LRC with optimal rate but with low minimum Hamming distance. Similarly, it is possible to replace C with an LRC of higher minimum Hamming distance (e.g., [26]) and obtain an existence proof for a set of permutations with locality *and* minimum Hamming distance. Since minimum distance constraints are not discussed in this paper, we choose the former approach for simplicity.

For certain small values of d , a similar lower bound can be derived from a well-studied problem in combinatorics, which is described in the remainder of this subsection. A *Latin square* of order n is a square $n \times n$ matrix with entries in $\{0, \dots, n-1\}$, such that in each row and in each column, all entries are distinct. A *cyclic* Latin square is a Latin square such that entry (i, j) equals $(i-j) \bmod n$ [19]. A *transversal* in a Latin square is a set of positions such that no two share the same row, column, or value. A transversal in a cyclic Latin square is equivalent to the following chess problem. A *semi-queen* is a queen that cannot move on the north-east south-west diagonal. In an $n \times n$ *toroidal chessboard*, it is possible to move across the generalized diagonals $\{(i, j) | i-j \equiv t \bmod n\}$ for all $t \in \{0, \dots, n-1\}$, even if the positions are not connected in

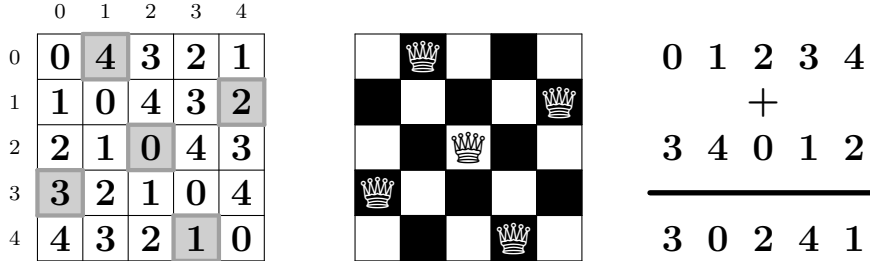


Fig. 1 The equivalence of a transversal in a cyclic Latin square, an arrangement of non attacking semi-queens in a toroidal chessboard, and a permutation σ such that $\text{Id} + \sigma$ is a permutation.

the ordinary chessboard. It is readily verified that a transversal in a cyclic Latin square of order n corresponds to a configuration of n non-attacking semi-queens in an $n \times n$ toroidal chess board, which in turn corresponds to a permutation $\sigma \in S_n$ such that $\text{Id} + \sigma \in S_n$, where the sum is taken mod n (see Figure 1).

For a permutation $\pi \in S_n$ let $P(\pi) \triangleq \{\sigma \in S_n \mid \pi + \sigma \in S_n\}$. For any two permutations $\pi_1, \pi_2 \in S_n$, by applying a proper permutation of entries we get a bijection between $P(\pi_1)$ and $P(\pi_2)$, and hence, $|P(\pi_1)| = |P(\pi_2)|$. Therefore, for any $\pi \in S_n$, the size of $P(\pi)$ depends only on n , and is thus denoted by t_n . Estimating t_n is a well-known problem which was resolved only recently [13]. However, given $\pi \in S_n$, constructing the set $P(\pi)$ efficiently is still an open problem.

Theorem 4 [13, Theorem 1.2] *If n is an odd integer then*

$$t_n = \left(e^{-1/2} + o(1) \right) n!^2 / n^{n-1}.$$

Assume that there exists an efficient algorithm B that on input (p, i) , where $p \in S_n$ and $1 \leq i \leq t_n$, outputs the i -th permutation $\sigma \in S_n$ such that $p + \sigma \in S_n$. The existence of B provides an efficient algorithmic construction of a set of permutations with locality and asymptotically optimal rate. In the following theorem, all additions are taken modulo $\frac{n}{d+1}$.

Lemma 5 For an odd n , an integer $d = 2^{o(\log n)}$ such that $d+1 \mid n$, and integers i_1, \dots, i_{d-1} such that $1 \leq i_j \leq t_n$ for all $j \in [d-1]$, let

$$\begin{aligned} p_1 &\in S_{\frac{n}{d+1}} \\ p_2 &\triangleq B(p_1, i_1) \\ p_3 &\triangleq B(p_1 + p_2, i_2) \\ p_4 &\triangleq B(p_1 + p_2 + p_3, i_3) \\ &\dots \\ p_d &\triangleq B\left(\sum_{j=1}^{d-1} p_j, i_{d-1}\right) \\ p_{d+1} &\triangleq \sum_{j=1}^d p_j, \end{aligned}$$

and define

$$\begin{aligned} \pi_{p_1, i_1, \dots, i_{d-1}} &\triangleq p_1 \circ \left(p_2 + \bar{1} \cdot \frac{n}{d+1}\right) \circ \left(p_3 + \bar{1} \cdot \frac{2n}{d+1}\right) \circ \left(p_4 + \bar{1} \cdot \frac{3n}{d+1}\right) \circ \dots \\ &\quad \circ \left(p_d + \bar{1} \cdot \frac{(d-1)n}{d+1}\right) \circ \left(p_{d+1} + \bar{1} \cdot \frac{dn}{d+1}\right) \end{aligned}$$

where $\bar{1}$ is the all 1's vector of length $\frac{n}{d+1}$. The resulting set

$$S \triangleq \{\pi_{p_1, i_1, \dots, i_{d-1}} \mid p_1 \in S_{n/(d+1)}, 1 \leq i_1, \dots, i_{d-1} \leq t_n\}$$

is a set of permutations in S_n with locality d , and optimal asymptotic rate $\frac{d}{d+1}$.

Proof Since for any k , $2 \leq k \leq d$, we have that $p_k = B(\sum_{j=1}^{k-1} p_j, i_{k-1})$, it follows from the definition of the algorithm B that $p_i \in S_{\frac{n}{d+1}}$ for all $i \in \{1, \dots, d+1\}$. Therefore, any $\pi \in S$ results from a concatenation of $d+1$ permutations on disjoint $\frac{n}{d+1}$ -subsets of $\{0, \dots, n-1\}$, and thus $S \subseteq S_n$. Since $p_{d+1} = \sum_{j=1}^d p_j$, it follows that any symbol of any $\pi \triangleq (\pi_0, \dots, \pi_{n-1}) \in S$ can be computed from d other symbols, since for all $j \in \{0, \dots, \frac{n}{d+1} - 1\}$ we have

$$\pi_{\frac{dn}{d+1}+j} - \frac{dn}{d+1} = \sum_{i=1}^d \left(\pi_{\frac{in}{d+1}+j} - \frac{in}{d+1} \right).$$

According to Theorem 4, the size of S is

$$\frac{n}{d+1}! (t_{\frac{n}{d+1}})^{d-1} = \Theta \left(\frac{\frac{n}{d+1}!^{2d-1}}{\left(\frac{n}{d+1}\right)^{\left(\frac{n}{d+1}-1\right)(d-1)}} \right).$$

Since $d = 2^{o(\log n)}$ we have that $\frac{\log d}{\log n} \xrightarrow{n \rightarrow \infty} 0$, and thus the asymptotic rate of S is

$$\begin{aligned} \frac{\log |S|}{\log(n!)} &\xrightarrow{n \rightarrow \infty} \frac{(2d-1) \frac{n}{d+1} \log \frac{n}{d+1} - \left(\frac{n}{d+1} - 1\right) (d-1) \log \frac{n}{d+1}}{n \log n} \\ &= \frac{\frac{d}{d+1} \cdot n + d - 1}{n} \cdot \frac{\log \frac{n}{d+1}}{\log n} \xrightarrow{n \rightarrow \infty} \frac{d}{d+1}. \end{aligned}$$

□

A non-efficient implementation of A may be obtained simply by traversing all permutations in S_n . However, providing an efficient implementation of A requires a rigorous understanding of the structure of the permutations in $P(\text{Id})$, which seems beyond the scope of contemporary knowledge. A subset of $P(\text{Id})$ of approximate size $\sqrt{n}^{\sqrt{n}}$ is given in [10], but it is too small to provide a non-trivial construction.

Remark 2 We note that a converse claim may also be made. That is, given an optimal set of permutations with constant locality $d \geq 2$, which is constructed according to the outline of Lemma 5, one may explicitly construct the set $P(\text{Id})$. Since an explicit construction of $P(\text{Id})$ is not known, this may serve as a hardness result for the construction of a set of permutations with constant locality $d \geq 2$. However, since the outline of the construction in Lemma 5 is highly restrictive, such a hardness result might not seem insightful enough.

4 High Rate Constructions

This section presents several simple constructions of sets of permutations with locality, some of which attain the upper bound given in Section 3.1. The well-known *alternating group* and its complement will be shown in Subsection 4.1 to have locality of $n - 2$, and attain the upper bound given in Theorem 1. Another simple set of permutations, discussed in Section 4.2, is those that may be seen as a concatenation of n/h permutations in S_h , for some h which divides n . The locality of the latter relies on the trivial observation that any single erasure in a permutation may be corrected without requiring additional redundancy. For $h = 2$, this set attains the upper bound given in Theorem 2. Subsection 4.3 shows a similar technique which achieves high locality. Subsection 4.4 and Subsection 4.5 enhance the construction of Subsection 4.2 by using Reed-Solomon codes over permutation polynomials, and by using multi-permutations. The results of this section are summarized in Table 1, in which three locality regimes are considered for comparison.

4.1 The Alternating Group

It is widely known [7] that any permutation may be represented as a product of transpositions (cycles of length two). Although many different products of

Table 1 Summary of the results in Section 4.

Section	Technique	Locality	Asymptotic Rate	Comments
4.1	The alternating group.	$n - 2$	1	Strictly optimal.
4.2	Concatenation.	$d = O(1)$	$\frac{1}{d+1}$	Strictly optimal for $d = 1$. At least n times smaller than Subsection 4.4.
		$\Theta(n^\epsilon)$	ϵ	-
		$\Theta(n)$	1	-
4.3	Range-restriction.	$\Theta(n)$	1	-
4.4	Reed-Solomon codes.	6	1/2	$n = 2^k$ for some k .
		7	1/2	$n = \pm 2 \pmod{5}$, and a prime power.
		7	1/2	$n = 5^k$ for some k .
4.5	Multi-permutations.	$\Theta(1)$	1/2	Incomparable with Subsection 4.4.
		$\Theta(n^\epsilon)$	$(1 + \epsilon)/2$	Larger rate than Subsection 4.2 for same locality.
		$\Theta(n)$	1	-

transpositions may represent the same permutation, all representations of a given permutation either contain an even or an odd number of transpositions. For a permutations $\pi \in S_n$, if the number of transpositions in any representation is even, we say that π is even and its *sign* is 1. Otherwise it is odd, and its sign is -1. The set of all even permutations, which forms a subgroup of S_n of size $n!/2$, is called *the alternating group* and denoted by A_n . In what follows we show that the sets A_n and $S_n \setminus A_n$ have locality of $n - 2$. This fact will follow from the next simple lemma.

Lemma 6 *If π and σ are two distinct permutations in S_n whose one-liners agree on $n - 2$ entries, then one of $\{\pi, \sigma\}$ is odd and the other is even.*

Proof Let $\{i_j\}_{j \in [n-2]}$ be the set of entries on whom π and σ agree, and let $\{\alpha_j\}_{j \in [n-2]}$ be the subset of $[n]$ such that for all $j \in [n - 2]$, $\pi_{i_j} = \sigma_{i_j} = \alpha_j$. Clearly, π and σ differ only in the arrangement of the elements in $[n] \setminus \{\alpha_j\}_{j \in [n-2]}$. Therefore, π may be obtained from σ by applying a single transposition which switches between the elements of $[n] \setminus \{\alpha_j\}_{j \in [n-2]}$, and hence π and σ have opposite signs. \square

Corollary 1 *The sets A_n and $S_n \setminus A_n$ have locality of $n - 2$.*

Proof If a symbol of the stored permutation π is missing, by observing any $n - 2$ of the remaining symbols there exists exactly two possibilities for π . According to Lemma 6, one of these options is an odd permutation and the other is even. Hence, restricting the system to store only permutations from either A_n or $S_n \setminus A_n$, we have only one possible permutation, and thus both A_n or $S_n \setminus A_n$ admit locality of $n - 2$. \square

Although the results in this subsection are rather simple, they shed some light on the tightness of the bound given in Theorem 1. Since $d = n - 2$ we have that $n!/([n/(d+1)]!) = n!/([n/(n-1)]!) = n!/2$, and thus A_n and $S_n \setminus A_n$ are optimal sets with locality $n - 2$.

4.2 Concatenation of Short Permutations

Obviously, in the one-line representation, any single symbol may easily be computed from all other symbols. This principle leads to simple sets of permutations which can be stored efficiently.

Consider the set S of permutations in S_n which may be viewed as a concatenation of n/h shorter permutations on h elements, for some integer h which divides n . That is, their one-liner may be viewed as a concatenation of n/h one-liners, each of which is a permutation of either of the sets $\{1, \dots, h\}, \{h+1, \dots, 2h\}$, etc. Clearly, S contains $(h!)^{n/h} \cdot (n/h)!$ permutations. A subset of S , in which the i -th permutation is on the set $\{(i-1)h+1, \dots, i \cdot h\}$, was considered in [27, Corollary 19], where it was shown to be an optimal anticode under the infinity metric d_∞ (see Definition 7 in Section 5.1 to follow).

Lemma 7 *If $\pi \in S$ then any symbol π_i can be computed from $h-1$ other symbols, i.e., the set S has locality $d = h-1$.*

Proof Since $\pi \in S$ it follows that $\{\pi_{h \cdot \lfloor i/h \rfloor + 1}, \dots, \pi_i, \dots, \pi_{(h+1) \cdot \lfloor i/h \rfloor}\} = \{jh+1, \dots, (j+1)h\}$. Hence, observing the value of $\pi_{h \cdot \lfloor i/h \rfloor + 1}, \dots, \pi_{(h+1) \cdot \lfloor i/h \rfloor}$ (excluding π_i), the range $\{jh+1, \dots, (j+1)h\}$ can be identified, and π_i is the missing value in it. \square

Note that multiple erasures can be corrected simultaneously, as long as they do not reside in the same short permutation. Two erasures from the same short permutation cannot be corrected simultaneously. In addition, Q1 can be answered trivially, and Q2 requires finding the suitable sub-permutation in n/h queries, and additional h queries to locate the desired element.

Since $d = h-1$, we have that $|S| = (d+1)!^{n/(d+1)} \cdot (n/(d+1))!$, and for $d = 1$ we have that

$$\begin{aligned} |S| &= 2^{n/2} \cdot (n/2)! = \left(\frac{n}{2}\right) \cdot 2 \cdot \left(\frac{n}{2} - 1\right) \cdot 2 \cdot \dots \cdot (1) \cdot 2 \\ &= n \cdot (n-2) \cdot (n-4) \cdot \dots \cdot 2 = n!! \end{aligned}$$

Hence, for $d = 1$ this construction attains the bound of Theorem 2 with equality. However, for any $d = O(1)$, $d \geq 2$, it can be shown that these sets *do not* attain the optimal rate, since they are superseded by the existential lower bound of Theorem 3.

Lemma 8 *If $h = O(1)$, the set S (of locality $d = h-1$) have rate $\frac{1}{d+1}$ as n tends to infinity.*

Proof By the construction above, we have that

$$\frac{\log \left((d+1)!^{\frac{n}{d+1}} \cdot \left(\frac{n}{d+1}\right)! \right)}{\log(n!)} = \frac{\log((d+1)!)}{d+1} \cdot \frac{n}{\log(n!)} + \frac{\log \left(\left(\frac{n}{d+1}\right)! \right)}{\log(n!)},$$

and since d is constant, it follows that

$$\begin{aligned}
& \xrightarrow{n \rightarrow \infty} \frac{\log((d+1)!)}{\log(n) \cdot (d+1)} + \frac{\frac{n}{d+1} \cdot \log\left(\frac{n}{d+1}\right)}{n \log(n)} \\
&= \frac{\log((d+1)!)}{\log(n) \cdot (d+1)} + \frac{1}{d+1} - \frac{\log(d+1)}{\log(n) \cdot (d+1)} \\
&= \frac{\log(d!)}{\log(n) \cdot (d+1)} + \frac{1}{d+1} \xrightarrow{n \rightarrow \infty} \frac{1}{d+1}.
\end{aligned}$$

□

By choosing $h = \Theta(n^\epsilon)$ for some constant $0 < \epsilon < 1$, we achieve a non-vanishing rate.

Corollary 2 *If $h = \Theta(n^\epsilon)$, the sets S (of locality $d = h - 1$) have rate ϵ as n tends to infinity.*

Proof

$$\frac{\log\left((d+1)!^{\frac{n}{d+1}} \cdot \left(\frac{n}{d+1}\right)!\right)}{\log(n!)} \xrightarrow{n \rightarrow \infty} \frac{n(d+1) \log(d+1) + n \log\left(\frac{n}{d+1}\right)}{(d+1)n \log n} \xrightarrow{n \rightarrow \infty} \epsilon.$$

□

In the high locality regime, where $d = \Theta(n)$, we may similarly prove that the rate of these codes approaches 1 as n approaches infinity.

Corollary 3 *If $h = \Theta(n)$, the sets S (of locality $d = h - 1$) have rate 1 as n tends to infinity.*

4.3 Concatenation of Range-Restricted Permutations

In this subsection we provide a technique for producing sets of permutations with high locality $d \geq n/2$. For a set of symbols Σ let $S(\Sigma)$ denote the set of all permutations of Σ , that is, the set of all injective functions from $[\Sigma]$ to Σ . In this subsection we use the alphabet $\Sigma = \{0, \dots, n-1\}$, and hence $S(\Sigma) = S_n$. Let h be an integer which divides n , and for $i \in \{0, \dots, n/h - 1\}$ let

$$K_i \triangleq S(\{ih, ih+1, \dots, (i+1)h-1\}) \circ S([n] \setminus \{ih, ih+1, \dots, (i+1)h-1\}).$$

Lemma 9 *The set $S^h \triangleq \cup_{i=0}^{n/h-1} K_i$ has locality $d = n - h - 1$.*

Proof To repair a missing symbol $\pi_j, 0 \leq j \leq n-1$ in $\pi \in S^h$, distinguish between the cases $j \leq h-1$ and $j \geq h$. If $j \leq h-1$, π_j may clearly be computed from $\{\pi_i\}_{i \in \{0, \dots, h-1\} \setminus \{j\}}$. If $j \geq h$, the set of symbols $\{\pi_i\}_{i \in \{h, \dots, n-1\} \setminus \{j\}}$ must contain a gap of h consecutive numbers, which are located in the prefix of π . After identifying this gap, the missing symbol π_j may easily be deduced. □

The set S^h contains $\frac{n}{h} \cdot h! \cdot (n-h)! = n \cdot (h-1)! \cdot (n-h)!$ and it does not attain the upper bound given in Theorem 1. For constant h the rate of S^h asymptotically approaches 1 as n goes to infinity, since

$$\frac{\log(n \cdot (h-1)! \cdot (n-h)!)}{\log(n!)} \geq \frac{\log((n-h)!)}{\log(n!)} \xrightarrow{n \rightarrow \infty} 1.$$

Equal rate may be obtained for lower locality, where $h = \Theta(n)$; if $h = \delta n$ for some constant $0 < \delta < 1$, then

$$\begin{aligned} \frac{\log(n \cdot (h-1)! \cdot (n-h)!)}{\log(n!)} &\xrightarrow{n \rightarrow \infty} \frac{\delta n \log(\delta n) + (1-\delta)n \log((1-\delta)n)}{n \log n} \\ &= \delta + (1-\delta) = 1. \end{aligned}$$

An identical rate is also obtained by choosing $h = \Theta(n^\epsilon)$. Hence, the best choice of parameters for this technique seems to be $h = \Theta(n)$, since it results in low locality and optimal rate.

4.4 Extended Construction from Error-Correcting Codes

This section provides a construction of a set of permutations in S_n with locality, from two constituent ingredients. The first ingredient is a set of permutations $S \subseteq S_{n-t}$ with locality d , for some given t and d . The second ingredient is an error-correcting code T , in which all codewords consist of t distinct symbols.

A *symbol replacement function* f is an injective function which maps one alphabet to another. Given a permutation π and a symbol replacement function f let $f(\pi)$ be the result of replacing the symbols of π according to f . For a set of permutations S let $f(S) \triangleq \{f(\pi) | \pi \in S\}$. The construction of this section relies on the following observation.

Observation 1 *If $S \subseteq S_{n-t}$ is a set of permutations with locality d and f is a symbol replacement function, then $f(S)$ is a set of permutations with locality d as well.*

Using a proper symbol replacement function f , a permutation $f(\pi)$ for $\pi \in S$ is concatenated to a codeword from T to create a permutation in S_n . This symbol replacement function is given in the following definition, which is followed by an example. In this definition, for a set of integers $E \triangleq \{e_1, e_2, \dots, e_{|E|}\}$ such that $e_1 < e_2 < \dots < e_{|E|}$ and an integer s , $s \leq |E|$, the s -smallest element of E is e_s .

Definition 4 For any integers $1 < t < n$, let π be a permutation in S_{n-t} and $e \in [n]^t$ be a word with t distinct symbols $\{\sigma_1, \dots, \sigma_t\} \triangleq E \subseteq [n]$. Let f_E be the following symbol replacement function

$$f_E : [n-t] \rightarrow ([n-t] \setminus E) \cup \{n-t+1, \dots, n-t+|E| \cap [n-t]\}$$

$$f_E(i) = \begin{cases} i, & i \notin E. \\ j, & \text{For some integer } s, i \text{ and } j \text{ are the } s\text{-smallest numbers} \\ & \text{in } E \cap [n-t] \text{ and } \{n-t+1, \dots, n\} \setminus E, \text{ respectively.} \end{cases}$$

That is, f_E maps each element which does not appear in E to itself, and each element which appears in E is mapped to a symbol in $\{n - t + 1, \dots, n\}$ which does not appear in E , in an increasing manner. Using f_E , define the operator \odot as

$$\pi \odot e \triangleq f_E(\pi) \circ e,$$

where \circ denotes the ordinary concatenation of strings.

Since symbols in π which occur in both π and e are replaced with symbols that do not appear in either π nor e , we have that $\pi \odot e$ is a permutation in S_n , as illustrated by the following example.

Example 2 For $n = 7$ and $t = 3$, let $\pi = (1, 2, 3, 4)$, $e = (3, 4, 7)$, and $E = \{3, 4, 7\}$. By Definition 4 we have that

$$\begin{aligned} f_E(1) &= 1, \quad f_E(2) = 2, \quad f_E(3) = 5, \quad f_E(4) = 6, \quad \text{and} \\ \pi \odot e &= f_E(\pi) \circ e = (1, 2, 5, 6, 3, 4, 7) \in S_7. \end{aligned}$$

The operation \odot is used to extend an existing set $S \subseteq S_{n-t}$ with locality to a subset of S_n with a larger locality by using an error-correcting MDS code T .

Lemma 10 *For integers $1 < t < n$, if $S \subseteq S_{n-t}$ is a set with locality d and T is an MDS code in $[n]^t$ with minimum distance δ and distinct symbols, then $S \odot T \triangleq \{s \odot e \mid s \in S, e \in T\} \subseteq S_n$ is a set of permutations with locality $d + t - \delta + 1$.*

Proof Let $\pi = s \odot e$ be a permutation in $S \odot T$. To repair a missing symbol π_j for $1 \leq j \leq n$ we distinguish between the cases $j \leq n - t$ and $j > n - t$. If $j > n - t$, by the minimum distance property of the MDS code T we may obtain π_j by accessing $t - \delta + 1$ symbols from e . If $j \leq n - t$, then by accessing $t - \delta + 1$ symbols from e we may identify the function f_E used to define the operator \odot (Definition 4). Once f_E is known, the locality follows from Observation 1. \square

This technique can be used to obtain explicit sets with constant locality $d \geq 2$, which are the largest ones in this paper for this locality. Unfortunately, to the best of our knowledge the asymptotic rate of these sets does not exceed $\frac{1}{2}$, and hence they are not optimal. Moreover, since a set with locality 1 also has locality $d \geq 2$ for any d , the sets of locality 1 from Subsection 4.2 can be used for any locality greater than 1, while obtaining rate of $\frac{1}{2}$ as well. Nevertheless, for small values of d we are able to construct explicit sets with locality d which contain more permutations than the sets with locality 1 from Subsection 4.2.

To provide good examples by this technique, we must construct error-correcting codes in which all codewords consist of distinct symbols. For this purpose, assume that n is a power of prime, and $\{0, 1, \dots, n - 1\}$ are representations of the elements of \mathbb{F}_n .

Recall that a *Reed-Solomon* code is given by evaluations of degree restricted polynomials on a fixed set of distinct elements from a large enough finite field.

These codes contain sub-codes which are suitable for our purpose. The code-words in these sub-codes are obtained by evaluations of *permutation polynomials*. A permutation polynomial is a polynomial which represents an injective function from \mathbb{F}_n to itself. In spite of the very limited knowledge on permutation polynomials in general, all permutation polynomials of degree at most 5 are known (see [9, Table 2]). For example, we have the following lemma.

Lemma 11 [9, Table 2]

1. If n is a power of 2, then there exist at least $(n-1)(2n + \frac{n(n^2+2)}{3})$ permutation polynomials of degree at most 4 over \mathbb{F}_n .
2. If n is a prime power and $n \equiv \pm 2 \pmod{5}$, then there exist at least $n^3(n-1)$ permutation polynomials of degree at most 5 over \mathbb{F}_n .
3. If n is a prime power and $n \equiv 0 \pmod{5}$, then there exist at least $\frac{1}{2}n^2(n-1)^2$ permutation polynomials of degree at most 5 over \mathbb{F}_n .

As a corollary, we obtain the following constructions.

Example 3

1. Let n be an integer power of 2, and let $S \subseteq S_{n-6}$ be an optimal set with locality 1 (which exists by Subsection 4.2, since $n-6$ is even). Let T be a subset of a Reed-Solomon code of dimension 5 and length 6 over \mathbb{F}_n , which corresponds to all permutation polynomials of degree at most 4. According to Lemma 10 and Lemma 11 (part 1), the set $B_1 \triangleq S \odot T$ contains $(n-6)!! \cdot (n-1)(2n + \frac{n(n^2+2)}{3})$ permutations, and has locality 6.
2. Let n be an odd prime power and $n \equiv \pm 2 \pmod{5}$ (such as 7, 17, 23, ... etc.), and let $S \subseteq S_{n-7}$ be an optimal set with locality 1, as above. Let T be a subset of a Reed-Solomon code of dimension 6 and length 7 over \mathbb{F}_n , which corresponds to all permutation polynomials of degree at most 5. According to Lemma 10 and Lemma 11 (part 2), the set $B_2 \triangleq S \odot T$ contains $(n-7)!! \cdot n^3(n-1)$ permutations, and has locality 7.
3. Let n be a prime power and $n \equiv 0 \pmod{5}$ (i.e., n is a power of 5), and let $S \subseteq S_{n-7}$ be an optimal set with locality 1, as above. Let T be a subset of a Reed-Solomon code of dimension 6 and length 7 over \mathbb{F}_n , which corresponds to all permutation polynomials of degree at most 5. According to Lemma 10 and Lemma 11 (part 3), the set $B_3 \triangleq S \odot T$ contains $(n-7)!! \cdot \frac{1}{2}n^2(n-1)^2$ permutations, and has locality 7.

Notice that an optimal set $A \subseteq S_n$ with locality 1, which may be seen as having any larger locality, contains $n!!$ permutations (see Section 4.2). The set B_1 is larger, since $n!! = (n-6)!! \cdot \Theta(n^3)$ and $|B_1| = (n-6)!! \cdot \Theta(n^4)$. Similarly, for odd values of n , Section 4.2 provides a set A of size $(n-1)!!$ and locality 1. The sets B_2 and B_3 are larger, since $|A| = (n-1)!! = (n-7)!! \cdot \Theta(n^3)$, where both $|B_2|$ and $|B_3|$ equal $(n-7)!! \cdot \Theta(n^4)$. Hence, the construction of this section provides sets which are at least n times larger than those given in Section 4.2, and have larger constant locality.

4.5 High-Locality Construction From Multi-Permutations

While constructing sets of permutations with constant locality $d \geq 2$ and rate above $\frac{1}{2}$ seems hard, it is fairly easy to construct sets with such rate and locality $d = \Theta(n^\epsilon)$, for constant $0 < \epsilon < 1$. Such a set is obtained from Section 4.2 by taking $h = \Theta(n^\epsilon)$. However, the resulting rate is

$$\frac{\log((h!)^{n/h} \left(\frac{n!}{h!}\right))}{\log n!} \xrightarrow{n \rightarrow \infty} \frac{\log n^\epsilon}{\log n} + \frac{n^{1-\epsilon}(1-\epsilon) \log n}{n \log n} = \epsilon,$$

where Theorem 3 guarantees that for this locality, there exist sets with rate which tends to 1 as n tends to infinity.

In this subsection it is shown that the construction from Section 4.2 may be enhanced by using multi-permutations, achieving a rate of $\frac{1}{2} + \frac{\epsilon}{2}$ for locality $d = \Theta(n^\epsilon)$. The methods and notations in this subsection are strongly based on [8].

For nonnegative integers ℓ and m , a *balanced multi-set* $\{1^m, 2^m, \dots, \ell^m\}$ is a collection of the elements in $[\ell]$, where each element appears m times. A *multi-permutation* on a balanced multi-set is a string of length ℓm , which is given by a function $\sigma : [\ell m] \rightarrow [\ell]$ such that for all $i \in [\ell]$, $|\{j | \sigma(j) = i\}| = m$. The set of all multi-permutations is denoted by $S_{\ell, m}$, and its size is $\frac{(m\ell)!}{(m!)^\ell}$. To distinguish between different appearances of the same element in a multi-permutation σ , for $j \in [m\ell]$, $i \in [\ell]$, and $r \in [m]$ we denote $\sigma(j) = i_r$ and $\sigma^{-1}(i_r) = j$ if the j -th position of σ contains the r -th appearance of i .

Example 4 If $m = 2$ and $\ell = 3$ then $\pi = (1, 1, 2, 3, 2, 3)$ is a multi-permutation on the balanced multi-set $\{1, 1, 2, 2, 3, 3\}$. To refer to the second appearance of 2 we say that $\pi(5) = 2_2$.

We are interested in multi-permutations with *two* appearances of each element, and therefore assume that $m = 2$ and $\ell = n/2$ (although some of the definitions in the sequel have counterparts for any m and ℓ such that $n = m\ell$). In particular, we consider such multi-permutations in which any two appearances of the same element are not too far apart. To this end, the following definition is required.

Definition 5 If $\pi \in S_{n/2, 2}$ and $t \in [n]$ then,

$$w(\pi) \triangleq \max_{i \in [n/2]} |\pi^{-1}(i_1) - \pi^{-1}(i_2)|, \text{ and} \\ B_t \triangleq \{\pi \in S_{n/2, 2} | w(\pi) \leq t\}.$$

That is, $w(\pi)$ indicates the maximum distance between two appearances of the same element, or alternatively, $w(\pi) - 1$ indicates the maximum number of elements between two appearances of the same element in π . For a given t , B_t is the set of all multi-permutations in $S_{n/2, 2}$ in which every two identical elements are separated by at most $t - 1$ other elements. Clearly, the multi-permutation π which was given in Example 4 is in B_2 .

To construct “ordinary” permutations in S_n from multi-permutations in $S_{n/2,2}$ we use the term *assignment of permutations*. As in Subsection 4.3, for a set of elements Σ we denote by $S(\Sigma)$ the set of all permutations of Σ (that is, the set of all injective functions $f : \{1, \dots, |\Sigma|\} \rightarrow \Sigma$).

Definition 6 If $\pi \in S_{n/2,2}$ and $\gamma_1, \dots, \gamma_{n/2}$ are permutations such that $\gamma_i \in S(\{2i-1, 2i\})$ for all i , then $\sigma = \pi(\gamma_1, \dots, \gamma_{n/2})$ is the permutation in S_n such that for all $1 \leq j \leq n$, if $\pi(j) = i_r$ then $\sigma(j) = \gamma_i(r)$.

Example 5 If $\pi = (1, 1, 2, 3, 2, 3) \in S_{3,2}$ and $\gamma_1 = (1, 2)$, $\gamma_2 = (4, 3)$, and $\gamma_3 = (6, 5)$ then $\sigma = \pi(\gamma_1, \gamma_2, \gamma_3) = (1, 2, 4, 6, 3, 5) \in S_6$.

Note that by choosing $h = 2$ in the construction which appears in Subsection 4.2, the resulting set S can be described as

$$S = \{\pi(\gamma_1, \dots, \gamma_{n/2}) \mid \forall i, \gamma_i \in S(\{2i-1, 2i\}) \text{ and } \pi \in B_1\}.$$

Hence, the construction in the following lemma may be seen as a generalization of the construction from Subsection 4.2.

Lemma 12 *For a nonnegative integer t , the set*

$$A_t \triangleq \{\pi(\gamma_1, \dots, \gamma_{n/2}) \mid \forall i, \gamma_i \in S(\{2i-1, 2i\}), \text{ and } \pi \in B_t\}$$

has locality $4t$.

Proof For $i \in [n/2]$ we say that the elements $\{2i-1, 2i\}$ are *counterparts*. Assume that a symbol π_i is erased, and let

$$\begin{aligned} D &\triangleq \{\pi_{i-t}, \dots, \pi_{i-1}, \pi_{i+1}, \dots, \pi_{i+t}\} \\ \overline{D} &\triangleq \{\pi_{i-2t}, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_{i+2t}\}, \end{aligned}$$

where we omit all π_j 's for which $j \notin [n]$. By the definition of A_t , all the counterparts of the elements in D are in \overline{D} . However, there exists $\sigma \in D$ which is the counterpart of π_i , and hence, σ 's counterpart is not to be found in $\overline{D} \setminus \{\pi_i\}$. Therefore, computing π_i is possible by inspecting $\overline{D} \setminus \{\pi_i\}$, and returning the counterpart of the only element in D whose counterpart is not in $\overline{D} \setminus \{\pi_i\}$. The claim follows since $|\overline{D} \setminus \{\pi_i\}| = 4t$. \square

Using this lemma, we are able to provide a set with high locality $\Theta(n^\epsilon)$, and asymptotic rate strictly above $\frac{1}{2}$. To bound this rate from below we require a lower bound on the size of B_t from Definition 5.

Lemma 13 $|B_t| \geq \left(\frac{t!}{(t/2)!}\right)^{n/t} \cdot \frac{n}{2}! \cdot 2^{-n/2}$.

Proof Since our goal is an asymptotic computation, we may w.l.o.g. assume that $t|n$ and $2|t$. Let E be the set of multi-permutations $\pi \in S_{n/2,2}$ that can be written as a concatenation $\pi = \pi^{(1)} \circ \dots \circ \pi^{(n/t)}$ of $\frac{n}{t}$ multi-permutations

on $t/2$ pairs of identical elements. A simple combinatorial calculation shows that

$$\begin{aligned}
|E| &\geq \prod_{i=1}^{n/t} (\text{no. of options to choose } \pi^{(i)}) \\
&\geq \left(\binom{n/2}{t/2} \cdot \frac{t!}{2^{t/2}} \right) \cdot \left(\binom{n/2 - t/2}{t/2} \cdot \frac{t!}{2^{t/2}} \right) \cdot \dots \cdot \left(1 \cdot \frac{t!}{2^{t/2}} \right) \\
&= \left(\frac{t!}{2^{t/2}} \right)^{n/t} \cdot \frac{(n/2)!}{(n/2 - t/2)!(t/2)!} \cdot \frac{(n/2 - t/2)!}{(n/2 - 2 \cdot t/2)!(t/2)!} \cdot \dots \cdot \frac{(t/2)!}{(t/2)!} \\
&= \left(\frac{t!}{2^{t/2}} \right)^{n/t} \cdot \frac{(n/2)!}{(t/2)!^{n/t}} = \left(\frac{t!}{(t/2)!} \right)^{n/t} \cdot \frac{n!}{2!} \cdot 2^{-n/2}.
\end{aligned}$$

Since clearly $E \subseteq B_t$, the claim follows. \square

Lemma 13 allows us to bound the asymptotic rate of the set A_t which was defined in Lemma 12.

Theorem 5 *If $t = \Theta(n^\epsilon)$ then $\lim_{n \rightarrow \infty} \frac{\log |A_t|}{\log n!} \geq \frac{1}{2} + \frac{\epsilon}{2}$.*

Proof According to Lemma 13, we have that $|A_t| \geq |B_t| \cdot 2^{n/2}$. Using the fact that $t = \Theta(n^\epsilon)$ and the approximation $\log(n!) \approx n \log n$, we have

$$\begin{aligned}
\frac{\log |A_t|}{\log n!} &\geq \frac{\log(|B_t| \cdot 2^{n/2})}{\log n!} \\
&= \frac{\log \left(\left(\frac{t!}{(t/2)!} \right)^{n/t} \cdot \frac{n!}{2!} \right)}{\log n!} \\
&= \frac{n \log(t!) - n \log((t/2)!) + \log(n!)}{t \log n!} \\
&\xrightarrow{n \rightarrow \infty} \frac{nt \log t - n \frac{t}{2} \log(t/2)}{tn \log n} + \frac{\frac{n}{2} \log((n/2))}{n \log n} \\
&= \frac{\epsilon \log n - \frac{\epsilon}{2} \log n + \frac{1}{2}}{\log n} + \frac{\log n - 1}{2 \log n} \xrightarrow{n \rightarrow \infty} \frac{1}{2} + \frac{\epsilon}{2}.
\end{aligned}$$

\square

5 Construction of Specific Families

5.1 Light Permutations by the Infinity Norm

In this subsection it is shown that permutations which involve small-magnitude shifts (in comparison with the original file), can be stored efficiently. Answering

Q1 requires downloading one symbol, whereas answering Q2 requires downloading a small number of adjacent symbols. An alternative and shorter representation, in which Q2 requires downloading one symbol and Q1 requires downloading a few, will be discussed in Subsection 6.1.

These permutations arise naturally when considering any ranking of items, in which an initial conjectured ranking is imposed, and any item is not expected to exceed its initial ranking by more than a certain bound.

Definition 7 If π and σ are two permutations in S_n , then

$$d_\infty(\pi, \sigma) \triangleq \max \{|\pi(i) - \sigma(i)|\}_{i \in [n]}.$$

If e is the identity permutation in S_n then $\ell_\infty(\pi) \triangleq d_\infty(\pi, e)$, and for any $r \in \{0, \dots, n-1\}$, let $\mathcal{B}_\infty(e, r)$ (\mathcal{B}_r in short) be the ball of radius r around e , that is, $\mathcal{B}_\infty(e, r) \triangleq \{\pi \in S_n \mid \ell_\infty(\pi) \leq r\}$.

Recall that we denote $\pi = (\pi_1, \dots, \pi_n) \triangleq (\pi^{-1}(1), \dots, \pi^{-1}(n))$, and for convenience, we say that $\pi_j = 0$ for any $j \notin [n]$.

Providing a closed-form expression for $|\mathcal{B}_r|$ is a notorious open problem. An efficient algorithm for computing $|\mathcal{B}_r|$ is given in [25, Corollary 5], from which the estimation $|\mathcal{B}_2| \approx 2.16^n$ follows [25, Example 2]. This estimation clearly implies that $|\mathcal{B}_r|$ is (at least) exponential in n for any r . On the other hand, by [27, Theorem 18] we have that the size of an *anti-code* of maximum d_∞ -distance $2r$ is at most $(2r+1)!^{n/(2r+1)}$. As \mathcal{B}_r is clearly an anti-code of maximum d_∞ -distance $2r$, we have that $|\mathcal{B}_r| \leq (2r+1)!^{n/(2r+1)}$. Since the exact size of \mathcal{B}_r is not known, the rate of this set is also unknown. However, for constant r , the upper bound implies that the rate of \mathcal{B}_r goes to zero as n goes to infinity.

The locality of the set \mathcal{B}_r relies on the following series of lemmas. The first lemma shows that an erasure in $\pi \in \mathcal{B}_r$ can be corrected by inspecting the entries in radius $2r$ from it.

Lemma 14 *If $\pi \in \mathcal{B}_r$ then for all $j \in [n]$, π_j is a function of $\pi_{j-2r}, \dots, \pi_{j-1}, \pi_{j+1}, \dots, \pi_{j+2r}$.*

Proof Since $\pi \in \mathcal{B}_r$, for any $t \in [n]$ we have that $t \in \{\pi_{t-r}, \dots, \pi_{t+r}\}$ and $\pi_t \in \{t-r, \dots, t+r\}$. Therefore,

$$\pi_j \in \{j-r, \dots, j, \dots, j+r\}, \text{ and} \\ \{j-r, \dots, j, \dots, j+r\} \subseteq \{\pi_{j-2r}, \dots, \pi_j, \dots, \pi_{j+2r}\}.$$

This implies that

$$\{j-r, \dots, j, \dots, j+r\} \setminus \{\pi_{j-2r}, \dots, \pi_{j-1}, \pi_{j+1}, \dots, \pi_{j+2r}\} = \{\pi_j\},$$

and hence, π_j may be computed given $\pi_{j-2r}, \dots, \pi_{j-1}, \pi_{j+1}, \dots, \pi_{j+2r}$. \square

Lemma 14 implies a correction algorithm for simultaneous erasures, in scenarios where the correction by Lemma 14 is impossible.

Lemma 15 *If $\pi \in \mathcal{B}_r$, then any set of erasures in which any two are separated by at least $2r - 1$ non-erased symbols can be corrected.*

Proof Let $E \triangleq \{s_1, \dots, s_t\}$ be the set of erased symbols in π . For each $s_i \in E$ define the *radius of possibility* $R(s_i) \triangleq \{s_i - r, \dots, s_i + r\}$ (a radius, in short), which is the set of the possible locations of the missing symbol s_i . Clearly, if π contains a single erasure in locations $R(s_i)$, then this erasure can be corrected to s_i . Hence, it suffices to show that for every t , the set $\{R(s_i)\}_{i=1}^t$ contains at least one radius which contains a single erasure.

Assume for contradiction that every radius in $\{R(s_i)\}_{i=1}^t$ contains at least two erasures. Notice that since the erasures are at least $2r - 1$ apart, every radius contains *exactly* two erasures. Let j be the location of the leftmost erasure, and assume w.l.o.g that it is contained in two radii $R(s_1)$ and $R(s_2)$. Since both radii contain two erasures, they both must contain the erasure located to the right of j , at $j + 2r$. Hence, since the size of both radii is $2r + 1$, we have that $R(s_1) = R(s_2) = \{j, \dots, j + 2r\}$, and thus $s_1 = s_2$, a contradiction. \square

Since permutations in \mathcal{B}_r involve small-magnitude shifts, Q2 can be answered by inspecting the values of $2r$ locations, r to the right and r to the left of location i . Permutations in \mathcal{B}_r admit a shorter representation, for which Q2 can be answered immediately, and Q1 requires inspecting $2r$ other symbols (see Subsection 6.1).

5.2 MinMax Permutations

The following set of permutations, called “MinMax”, includes many natural ones. E.g., all single left (right) cyclic shifts of any prefix (suffix) of the file. These permutations arise in the context of content consumption, such as news or tweets, in which the consumer begins with an arbitrary item of a feed, and either proceeds forward or backwards from the set of consecutive items which he read so far.

Definition 8 The set of MinMax permutations \mathcal{M} consists of two subsets denoted \mathcal{M}_L and \mathcal{M}_R . The subset \mathcal{M}_L (\mathcal{M}_R) consists of all permutations in which every element is either greater than the maximal element to his left (right) by 1, or smaller than the minimal element to his left (right) by 1. That is,

$$\begin{aligned}\mathcal{M}_L &\triangleq \left\{ \pi \in S_n \mid \forall i, \pi_i \in \left\{ \max_{j < i} \pi_j + 1, \min_{j < i} \pi_j - 1 \right\} \right\}, \\ \mathcal{M}_R &\triangleq \left\{ \pi \in S_n \mid \forall i, \pi_i \in \left\{ \max_{j > i} \pi_j + 1, \min_{j > i} \pi_j - 1 \right\} \right\}, \text{ and} \\ \mathcal{M} &\triangleq \mathcal{M}_L \cup \mathcal{M}_R.\end{aligned}$$

Example 6 For $n = 7$,

- The permutation $(3, 4, 5, 2, 1, 6, 7)$ is in \mathcal{M}_L but not in \mathcal{M}_R .
- The permutation $(1, 2, 3, 4, 5, 6, 7)$ is in \mathcal{M}_L and in \mathcal{M}_R .
- The permutation $(3, 5, 1, 2, 7, 6, 4)$ is neither in \mathcal{M}_L nor in \mathcal{M}_R .

The following lemma provides an alternative to Definition 8, and will be used in the sequel.

Lemma 16 $\pi \in \mathcal{M}_L$ if and only if for all $i \in [n]$, the set $\{\pi_1, \dots, \pi_i\}$ contains consecutive numbers. Similarly, $\pi \in \mathcal{M}_R$ if and only if for all $i \in [n]$, the set $\{\pi_i, \dots, \pi_n\}$ contains consecutive numbers.

Proof For $\pi \in \mathcal{M}_L$ we prove by induction on i that the set $\{\pi_1, \dots, \pi_i\}$ contains consecutive numbers. For $i = 1$ the claim is clear. For an arbitrary $i > 1$, by the induction hypothesis we have that $\{\pi_1, \dots, \pi_{i-1}\}$ is a set of consecutive numbers. By the definition of \mathcal{M}_L we have that $\pi_i \in \{\max_{j < i} \pi_j + 1, \min_{j < i} \pi_j - 1\}$, and hence the set $\{\pi_1, \dots, \pi_i\}$ is a set of consecutive numbers as well.

On the other hand, if for all i the set $\{\pi_1, \dots, \pi_i\}$ contains consecutive numbers, we show by induction on i that $\pi_i \in \{\max_{j < i} \pi_j + 1, \min_{j < i} \pi_j - 1\}$. For $i = 1$ the claim is clear. For an arbitrary $i > 1$, by the induction hypothesis we have that $\pi_{i-1} \in \{\max_{j < i-1} \pi_j + 1, \min_{j < i-1} \pi_j - 1\}$. W.l.o.g assume that $\pi_{i-1} = \max_{j < i-1} \pi_j + 1$, and hence $\pi_{i-1} > \pi_j$ for all $j < i - 1$. Therefore, since the set $\{\pi_1, \dots, \pi_i\}$ contains consecutive numbers, we have that either $\pi_i > \pi_{i-1}$, and hence $\pi_i = \max_{j < i} \pi_j + 1$, or $\pi_i < \pi_{i-1}$, and hence $\pi_i = \min_{j < i} \pi_j - 1$. For the case $\pi_{i-1} = \min_{j < i-1} \pi_j - 1$, the proof is similar, and if $\pi \in \mathcal{M}_R$, the proof is similar as well. \square

Corollary 4 If $\pi \in \mathcal{M}_L$ then $\pi_n \in \{1, n\}$, and if $\pi \in \mathcal{M}_R$ then $\pi_1 \in \{1, n\}$.

Proof If $\pi \in \mathcal{M}_L$, by Lemma 16 we have that $\{\pi_1, \dots, \pi_{n-1}\}$ is a set of consecutive numbers from $[n]$, and hence it is either $\{1, \dots, n-1\}$ or $\{2, \dots, n\}$. Therefore, $\pi_n \in \{1, n\}$. If $\pi \in \mathcal{M}_R$, the proof is similar. \square

Albeit the simple structure of their elements, the sets \mathcal{M}_L and \mathcal{M}_R are rather large, as shown below.

Lemma 17 $|\mathcal{M}_L| = |\mathcal{M}_R| = 2^{n-1}$.

Proof Let $\mathcal{M}_{L,i}$ be the set \mathcal{M}_L for $n = i$, and let $M_{L,i}$ be its size. According to Corollary 4, for any i , if $\pi \in \mathcal{M}_{L,i}$ then $\pi_i \in \{1, i\}$. Therefore, $\pi \in \mathcal{M}_{L,i}$ corresponds to exactly two permutations $\pi^{(1)}, \pi^{(2)} \in \mathcal{M}_{L,i+1}$. The first permutation is $\pi^{(1)} = (\pi_1, \dots, \pi_i, i+1)$, where the second is $\pi^{(2)} = (\pi_1 + 1, \dots, \pi_i + 1, 1)$. Since this mapping clearly covers the entire set $\mathcal{M}_{L,i+1}$, we have that $M_{L,i+1} = 2M_{L,i}$, and since $M_{L,1} = 1$, the claim follows. The proof for \mathcal{M}_R is symmetric. \square

Lemma 18 $\mathcal{M}_L \cap \mathcal{M}_R = \{(1, \dots, n), (n, \dots, 1)\}$.

Proof If $\pi \in \mathcal{M}_L \cap \mathcal{M}_R$, then according to Corollary 4 we have that $\{\pi_1, \pi_n\} = \{1, n\}$. By Definition 8, if $(\pi_1, \pi_n) = (1, n)$ then $\pi = (1, \dots, n)$, and if $(\pi_1, \pi_n) = (n, 1)$ then $\pi = (n, \dots, 1)$. \square

Corollary 5 $|\mathcal{M}| = 2^n - 2$.

Proof By Lemma 17 and Lemma 18, $|\mathcal{M}| = |\mathcal{M}_L| + |\mathcal{M}_R| - |\mathcal{M}_L \cap \mathcal{M}_R| = 2^n - 2$. \square

The locality of \mathcal{M}_L and \mathcal{M}_R relies on the following lemma.

Lemma 19 *If $\pi \in \mathcal{M}_L$, then every π_i is a function of at most three other symbols of π .*

Proof We distinguish between the following three cases.

- i = 1.** We compute π_1 from π_2 and π_3 . Notice that by Definition 8, $|\pi_3 - \pi_2| \leq 2$, since otherwise π_3 does not comply with the definition. Given the value of π_2 , we have that either $\pi_1 = \pi_2 + 1$ or $\pi_1 = \pi_2 - 1$. If $\pi_3 = \pi_2 \pm 1$, we are done. If $\pi_3 - \pi_2 = 2$, then by Definition 8 we have that $\pi_3 = \max\{\pi_1, \pi_2\} + 1$, and hence $\pi_1 = \pi_2 + 1$. Similarly, if $\pi_2 - \pi_3 = 2$ we have that $\pi_3 = \min\{\pi_1, \pi_2\} - 1$, and hence $\pi_1 = \pi_2 - 1$.
- i = n.** We compute π_n from π_{n-1} and π_{n-2} . By Corollary 4, we have that $\pi_n \in \{1, n\}$, and hence, if $\{1, n\} \cap \{\pi_{n-1}, \pi_{n-2}\} \neq \emptyset$, we are done. Else, if $\pi_{n-1} > \pi_{n-2}$, we have that $\pi_{n-1} > \pi_j$ for all $j < n-1$. Since π_{n-1} is larger than $n-2$ numbers in $[n]$ we have that $\pi_{n-1} \in \{n-1, n\}$. Since the case $\pi_{n-1} = n$ was already considered, we have that $\pi_{n-1} = n-1$, and hence, $\pi_n = n$. On the other hand, if $\pi_{n-1} < \pi_{n-2}$, then similarly, $\pi_{n-1} < \pi_j$ for all $j < n-1$, and hence $\pi_n = 1$.
- i $\notin \{1, n\}$.** We compute π_i from π_{i-1}, π_{i+1} , and if $i > 2$ we require an arbitrary π_j for $j < i-1$. Let $A \triangleq \{\pi_1, \dots, \pi_i\}$ and $n \triangleq \{\pi_1, \dots, \pi_{i-2}\}$. According to Lemma 16 and Definition 8, if $\pi_{i-1} < \pi_{i+1}$ we have that $A = \{\pi_{i+1} - i, \dots, \pi_{i+1} - 1\}$, and if $\pi_{i-1} > \pi_{i+1}$, we have that $A = \{\pi_{i+1} + 1, \dots, \pi_{i+1} + i\}$. If $i = 2$, and thus $n = \emptyset$, we have that π_i is the only element in the singleton $A \setminus \{\pi_{i-1}\}$. Otherwise, we examine *any* π_j for $j < i-1$ in order to distinguish between the cases $n = \{\pi_{i-1} + 1, \dots, \pi_{i-1} + i - 2\}$ and $n = \{\pi_{i-1} - (i-2), \dots, \pi_{i-1} - 1\}$. Knowing n , we have that π_i is the only symbol in the singleton $A \setminus (n \cup \{\pi_{i-1}\})$. \square

Since permutations in \mathcal{M}_R are entirely symmetric to the ones in \mathcal{M}_L , we have that by replacing i for $n - i + 1$ in the proof of Lemma 19, we are able to prove the following.

Lemma 20 *If $\pi \in \mathcal{M}_R$, then every π_i is a function of at most three other symbols of π .*

Notice that the repair algorithms, which are induced by Lemma 19, slightly differ for $\pi \in \mathcal{M}_L$ and $\pi \in \mathcal{M}_R$. Therefore, the user must know if the stored

permutation belongs to \mathcal{M}_L or to \mathcal{M}_R . Clearly, this information may be stored in the system with *one* additional bit. Alternatively, it can also be diffused by querying either one of π_1, π_n , as shown in the following lemma.

Lemma 21 *For $\pi \in \mathcal{M}$, if $\pi_1 \notin \{1, n\}$ then $\pi \in \mathcal{M}_L$, and if $\pi_1 \in \{1, n\}$ then $\pi \in \mathcal{M}_R$. Similarly, if $\pi_n \notin \{1, n\}$ then $\pi \in \mathcal{M}_R$, and if $\pi_n \in \{1, n\}$ then $\pi \in \mathcal{M}_L$.*

Proof If $\pi_1 \notin \{1, n\}$ then by Corollary 4 we have that $\pi \notin \mathcal{M}_R$, and hence $\pi \in \mathcal{M}_L$. On the other hand, if $\pi_1 \in \{1, n\}$ then π may be either in \mathcal{M}_L or in \mathcal{M}_R . However, if $\pi \in \mathcal{M}_L$ and $\pi_1 \in \{1, n\}$, it follows by Definition 8 that either $\pi = (1, \dots, n)$ or $\pi = (n, \dots, 1)$, which by Lemma 18 implies that $\pi \in \mathcal{M}_R$. The second part of the proof is similar. \square

As a corollary of Lemma 19, Lemma 20, and Lemma 21, we have the following.

Corollary 6 *If $\pi \in \mathcal{M}$, then every π_i is a function of at most four other symbols of π .*

In the sequel we analyse patterns of erasures that can be corrected simultaneously. To this end, we devise the following graph G (Figure 2), which follows from Lemma 19.

The vertices of G correspond to subsets of locations in the permutation, where we denote i instead of $\{i\}$ for convenience. A directed edge (i, S) exists if the repair of π_i requires precisely one (arbitrary) symbol π_j , $j \in S$. An analogue graph may be achieved for $\pi \in \mathcal{M}_R$ by replacing i for $n - i + 1$ in each set.

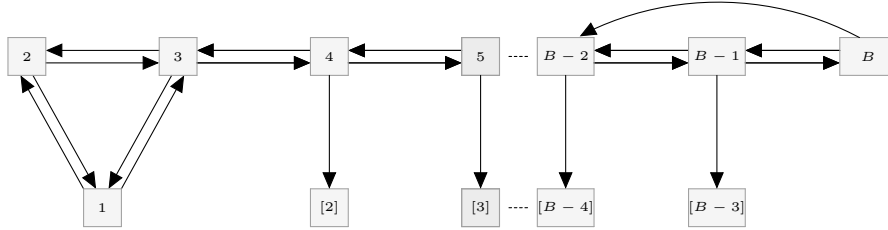


Fig. 2 The dependency graph between the symbols of a MinMax permutation $\pi \in \mathcal{M}_L$. Nodes represent subsets of symbol locations, where $\{i\}$ is denoted by i . An edge (i, S) indicates that the repair of symbol π_i requires exactly one arbitrary symbol $\pi_j, j \in S$.

For a vertex v in G which corresponds to a singleton, let $\Gamma(v)$ be its set of outgoing neighbors. A vertex u which corresponds to a subset S is called *active*, if there exists $j \in S$ such that π_j is available to the user. According

to Lemma 19, a symbol π_i is repairable if all nodes in $\Gamma(i)$ are active. The following lemma presents the sets of erasures that may be simultaneously corrected. A symmetric analogue of Lemma 22 for $\pi \in \mathcal{M}_R$ may be proved similarly.

Lemma 22 *Given a set of erased symbols, a permutation $\pi \in \mathcal{M}_L$ can be reconstructed if every path of erased symbols in G is finite, and terminates with a node i such that every node in $\Gamma(i)$ is active.*

Proof Assume that $\{\pi_i\}_{i \in E}$ is a set of missing symbols, such that every path in G whose vertex set is contained in E , terminates with a node i such that $\Gamma(i)$ are active. Using induction on the maximum length of those paths, we get that π can be reconstructed. If this maximum length is 1, π clearly can be reconstructed. If it is greater than 1, we may repair all terminal nodes in all maximum length paths, and use the induction hypothesis. \square

6 Efficient Representation of Permutations

6.1 Light Permutations by the Infinity Norm - An Alternative Representation

A permutation $\pi \in \mathcal{B}_r$ can be mapped to a list of size n , which indicates the displacement $\pi(i) - i$ for each $i \in [n]$. This mapping, denoted by $s(\pi)$, requires $n(\log r + 1)$ bits. By storing this representation rather than the one-liner, we may answer Q2 simply by inspecting $s(\pi)_i$ and returning $i + s(\pi)_i$. Since permutations in \mathcal{B}_r involve small-magnitude shifts, to answer Q1 we may look at $s(\pi)_j$ for $j \in \{i - r, \dots, i + r - 1\}$, return the unique j such that $j + s(\pi)_j = i$, or return $i + r$ if no such j exists. The equivalent claims of Lemma 14 and Lemma 15 are given below.

Lemma 23 *If $\pi \in \mathcal{B}_r$ then for all $j \in [n]$, $s(\pi)_j$ is a function of $s(\pi)_{j-2r}, \dots, s(\pi)_{j-1}, s(\pi)_{j+1}, \dots, s(\pi)_{j+2r}$.*

Proof The missing entry $s(\pi)_j$, specifying the shift of the element j , is computed by maintaining a binary array A which indicates which are the known “occupied” positions in the one-liner of π . Once we are left with a single non-occupied position A_ℓ for some ℓ , we infer that element j could only be located at ℓ , and compute its shift.

Formally, initialize an array A of $2r + 1$ zeros, whose entries are indexed by $\{j - r, \dots, j, \dots, j + r\}$, which are all possible positions of j . Set entry A_i to one if and only if there exists $t \in \{j - 2r, \dots, j - 1, j + 1, \dots, j + 2r\}$ such that $t + s(\pi)_t = i$. Consequently, entry A_i will indicate if one of the elements $\{j - 2r, \dots, j - 1, j + 1, j + 2r\}$, whose location is known from $s(\pi)_{j-2r}, \dots, s(\pi)_{j-1}, s(\pi)_{j+1}, \dots, s(\pi)_{j+2r}$, is located in position i .

Since $\pi \in \mathcal{B}_r$, for each $i \in \{j - r, \dots, j, \dots, j + r\}$ there exist a unique $t \in \{j - 2r, \dots, j, \dots, j + 2r\}$ such that $t + s(\pi)_t = i$. Therefore, there exists a unique entry ℓ in A which remains zero, and hence, $s(\pi)_j = \ell - j$. \square

Lemma 24 *If $\pi \in \mathcal{B}_r$, then any set of erasures in $s(\pi)$, in which any two are separated by at least $2r - 1$ non-erased symbols, can be corrected.*

Proof As in the proof of Lemma 23, we use an auxiliary array whose entries specify which are the occupied positions in the one-liner of π .

Initialize an array A of n zeros, and for each non-erased symbol $s(\pi)_j$ set $A_{s(\pi)_j+j}$ to one. Let $E = \{u_i\}_{i=1}^t$ be the set of indices of zeros in A , that is, $A_\ell = 0$ if and only if $\ell \in E$, and notice that the number of zeros in A equals the number of erasures. For each $i \in [t]$ define the radius of possibility $R(i) \triangleq \{\max\{u_i - \ell, 1\}\}_{\ell=r}^{-r}$ (radius, in short). Clearly, if there exists a single erasure among $\{s(\pi)_\ell \mid \ell \in R(i)\}$, say in $s(\pi)_{\ell_0}$, then it may be corrected to $u_i - \ell_0$. Hence, it suffices to show that for any t , there exists an $i \in [t]$ such that there is a single erasure among $\{s(\pi)_\ell \mid \ell \in R(i)\}$.

Assume for contradiction that all radii contain at least two erasures. Since any two erasures have at least $2r - 1$ non-erased symbol between them, and since all radii consist of $2r + 1$ consecutive integers, we have that each radius contains *exactly* two erasures. Now let $e_1 < \dots < e_t$ be the indices of the erasures in $s(\pi)$, and notice that each radius must contain exactly e_i and e_{i+1} for some $i \in [t - 1]$. Therefore, the maximum number of radii is $t - 1$, a contradiction. \square

6.2 Supporting Arbitrary Powers - a framework

In this section we present a framework for storing a permutation π from a set T , while allowing the user to query $\pi^k(i)$ for any $i \in [n]$ and any integer k (positive or negative, where $\pi^0(i) = i$ for all $i \in [n]$). This framework will be presented with respect to a general set T , and the properties of the resulting system will depend on the specific choice of T .

This framework is strongly based on [20], which provides an algorithm for representing any permutation $\pi \in S_n$. In [20], the queries $\pi^k(i) = ?$ are answered using a *rank-select* data structure which is maintained separately from π . This data structure requires as much as $n + o(n)$ additional bits of storage, and is required in its entirety for any operation. For this reason, we modify the techniques of [20] to achieve a scheme which is applicable for distributed storage systems. In addition, to obtain locality, this scheme can be combined with the techniques that were developed in earlier sections.

A permutation $\pi \in S_n$ may be given in its disjoint cycle representation. This representation is not unique, as each cycle may be cyclically shifted, and cycles may be permuted. For a disjoint cycle representation $y(\pi)$ of a permutation π , let $\overline{y(\pi)}$ denote the string obtained by omitting all brackets from $y(\pi)$, e.g.,

$$\begin{aligned} y(\pi) &= (123)(45) \\ \overline{y(\pi)} &= 1\ 2\ 3\ 4\ 5. \end{aligned}$$

Clearly, $\overline{y(\pi)}$ may be considered as a one-liner of a different permutation. Relying on this principle, we present a framework for storing permutations.

Let $S \subseteq S_n$ be a set of permutations with locality d , enabling an answer for Q1 by downloading q_1 symbols, and an answer for Q2 by downloading q_2 symbols. In addition, let t be the maximum length of a cycle in a permutation in S . Let $\gamma(S)$ be the set of all permutations $\pi \in S_n$ that have a disjoint cycle representation $y(\pi)$, such that the one-liner $\overline{y(\pi)}$ is in S . That is,

$$\gamma(S) \triangleq \left\{ \pi \in S_n \mid \exists y(\pi) \text{ s.t. } \overline{y(\pi)} \in S \right\}.$$

Notice that S and $\gamma(S)$ are not equal. E.g., for $n = 4$, by Corollary 5 there are 14 MinMax permutations, but a simple computer search shows that 23 permutations $\pi \in S_4$ have a disjoint cycle representation $y(\pi)$ such that $\overline{y(\pi)}$ is a one-liner of a MinMax permutation.

We encode a permutation $\pi \in \gamma(S)$ to a sequence of n triples $\{(\psi_i, c_i, \ell_i)\}_{i=1}^n$, where for all i , $\psi_i \in [n]$ and $c_i, \ell_i \in [t]$. Each triple is then placed in a storage node. Let i_1, i_2, \dots be integers such that the representation

$$y(\pi) = (i_1, \pi(i_1), \pi^2(i_1), \dots)(i_2, \pi(i_2), \pi^2(i_2), \dots) \dots$$

satisfies $\overline{y(\pi)} \in S$, and let $\psi \in S$ be the permutation whose one-liner is $\overline{y(\pi)}$. That is,

$$\psi = (\psi_1, \dots, \psi_n) \triangleq (i_1, \pi(i_1), \pi^2(i_1), \dots, i_2, \pi(i_2), \pi^2(i_2), \dots).$$

For a symbol ψ_i , let c_i be the length of the cycle in $y(\pi)$ that contains ψ_i , and let ℓ_i be the location of ψ_i in this cycle.

If node v_i fails, use the repair algorithm for S to obtain ψ_i . In addition, download $c_{i-1}, \ell_{i-1}, c_{i+1}, \ell_{i+1}$ from nodes v_{i-1}, v_{i+1} and obtain c_i, ℓ_i by the following guidelines.

$$\begin{aligned} \text{if } c_{i-1} = \ell_{i-1} \text{ then } & \begin{cases} c_i = \ell_i = 1 & \text{if } \ell_{i+1} = 1 \\ c_i = c_{i+1}, \ell_i = 1 & \text{if } \ell_{i+1} \neq 1 \end{cases}, \text{ and} \\ \text{if } c_{i-1} \neq \ell_{i-1} \text{ then } & c_i = c_{i-1}, \ell_i = \ell_{i-1} + 1. \end{aligned}$$

Lemma 25 *This system is capable of providing $\pi^k(i)$ for any $\pi \in S_n$, $i \in [n]$, and any integer k by downloading $(q_1 + q_2) \log n + 2 \log t$ bits.*

Proof Given i and k , perform the following algorithm.

1. Find $j \triangleq \psi(i)$ by downloading $q_1 \log n$ bits.
2. Download c_j and ℓ_j from v_j ($2 \log t$ bits).
3. Compute $s \triangleq j - \ell_j + (\ell_j + k) \bmod c_j$.
4. Return $\psi_s = \psi^{-1}(s)$ by downloading $q_2 \log n$ bits.

Clearly, these steps require downloading $(q_1 + q_2) \log n + 2 \log t$ bits. We are left to show that $\psi_s = \pi^k(i)$. Step 1 of the algorithm provides the *location* of symbol i in the one-liner ψ , since $\psi_j = \psi^{-1}(j) = i$. In step 3, the expression $j - \ell_j$ is the starting point of the cycle which contains symbol i , and by adding $(\ell_j + k) \bmod c_j$ we have the location s of the element $\pi^k(i)$. \square

Example 7 Let $y(\pi) = (1\ 0\ 2)(4\ 3\ 5)(7\ 6)$ be a disjoint cycle representation of a permutation π on eight elements. Clearly, $y(\pi) = (1\ 0\ 2\ 4\ 3\ 5\ 7\ 6)$ is a one-liner of a permutation $\psi \in \mathcal{B}_\infty(e, 1) = \mathcal{B}_1$ (see Section 5.1), and thus $q_1 = 1$ and $q_2 = 2$. The permutation π is stored on v_0, \dots, v_7 as follows.

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
ψ_i	1	0	2	4	3	5	7	6
c_i	3	3	3	3	3	3	2	2
ℓ_i	0	1	2	0	1	2	0	1

For example, follow the algorithm in Lemma 25 for answering $\pi^2(3)$ (that is, $i = 3$ and $k = 2$). Using the algorithm for \mathcal{B}_1 , in step 1 the user finds the location of 3 in the one-liner, which is $j = 4$. The user later downloads $(c_4, \ell_4) = (3, 1)$ from v_4 , computes $s = 4 - 1 + (1 + 2) \bmod 3 = 3$. In step 4 the user downloads $\psi_3 = 4$ from v_3 , which equals $\pi^2(3)$.

The obvious drawback of this system is the large storage overhead of $2n \log t$, on top of the $n \log n$ bits which are required to store ψ . With no known restriction on t , the total storage in this system is approximately $3n \log n$. However, a considerable advantage is the ability to obtain any power of the stored permutation, with a small computational overhead.

For a given set $S \subseteq S_n$, natural questions to ask are what is the size of $\gamma(S)$, and what is the structure of the permutations in $\gamma(S)$. The answers to these two questions seem rather involved. Therefore, in Table 2 we provide an insight towards the answer to the former, using a computer search, and for small values of n . It is evident from this table that the size of $\gamma(S)$ is most likely much larger than the size of S .

n	$n!$	$ \mathcal{M} $ (Definition 8)	$ \gamma(\mathcal{M}) $	$ \mathcal{B}_1 $ (Definition 7)	$ \gamma(\mathcal{B}_1) $	$ \mathcal{B}_2 $	$ \gamma(\mathcal{B}_2) $
3	6	6	6	3	6	6	6
4	24	14	23	5	22	14	24
5	120	30	99	8	66	31	117
6	720	62	400	13	192	73	567
7	5040	126	1532	21	560	172	2371
8	40320	254	5713	34	1660	400	9262

Table 2 Sizes of certain permutations sets for small values of n .

6.3 Beating LRCs - a proof of concept

Consider the scenario in which we would like to store *any* permutation using minimal redundancy. A possible simple solution is to use LRCs (Subsection 2), which will require at least $\frac{n}{d}$ additional symbols, or $\frac{n}{d} \log n$ additional bits, according to (2). This minimal redundancy may be achieved by adding a “parity check” symbol for any of n/d disjoint d -subsets of symbols. In this section, it is

shown that the fact that we are operating over permutations may be utilized to reduce the redundancy for $d \in \{2, 3\}$. Although the improvement is highly negligible, it may be shown to achieve the information theoretic lower bound, and may constitute a proof of concept for future research.

For locality $d = 2$, we present the following storage scheme, where we assume for simplicity that n is even. To store a permutation $\pi \in S_n$, encode it as follows

$$(\pi_1, \dots, \pi_n) \mapsto (\sigma_1, \dots, \sigma_{n+n/2}) \triangleq (\pi_1, \dots, \pi_n, \pi_1 - \pi_2, \pi_3 - \pi_4, \dots).$$

Notice that since π is a permutation, each of the elements $\sigma_{n+1}, \dots, \sigma_{n+n/2}$ may be represented using $\log(n-1)$ bits. For each $i \in \{1, \dots, \lceil n/2 \rceil\}$, any one erasure from $\{\sigma_{2i}, \sigma_{2i-1}, \sigma_{n+i}\}$ can be repaired using the other two non-erased symbols.

For locality $d = 3$, we assume that n is a power of prime and the entries $\{0, 1, \dots, n-1\}$ of the permutation are the elements of the finite field \mathbb{F}_n . We use the function $f(x, y, z) = \frac{x-y}{z-y}$, and the following lemma.

Lemma 26 *For field elements a_1, a_2, a_3 , $f(a_1, a_2, a_3) \in \{2, \dots, n-1\}$ if and only if a_1, a_2, a_3 are distinct.*

Proof If $f(a_1, a_2, a_3) \in \{2, \dots, n-1\}$ then f is well-defined over (a_1, a_2, a_3) , and hence $a_2 \neq a_3$. In addition, if $a_1 = a_2$ then $f(a_1, a_2, a_3) = 0$, and similarly, if $a_1 = a_3$ then $f(a_1, a_2, a_3) = 1$. Hence, a_1, a_2, a_3 are distinct. Conversely, if a_1, a_2, a_3 are distinct, then $f(a_1, a_2, a_3)$ is well-defined, and cannot output neither 0 nor 1. \square

According to Lemma 26, when f is applied over a subset of symbols of a permutation, the result may be represented by $\log(n-2)$ bits. This gives rise to the following scheme. For simplicity assume that $3|n$, although this scheme may be slightly changed to fit any prime power. To store $\pi \in S_n$, encode it as follows

$$(\pi_1, \dots, \pi_n) \mapsto (\sigma_1, \dots, \sigma_{n+n/3}) \triangleq (\pi_1, \dots, \pi_n, f(\pi_1, \pi_2, \pi_3), \dots, f(\pi_{n-2}, \pi_{n-1}, \pi_n)).$$

Clearly, for each $i \in \{1, \dots, n/3\}$, any one erasure from $\{\sigma_{3i}, \sigma_{3i-1}, \sigma_{3i-2}, \sigma_{n+i}\}$ can be repaired using the other three non-erased symbols.

The above schemes use $n \log n + \frac{n}{d} \log(n-d+1)$ bits to store the $n \log n$ bits of any permutation π , for $d \in \{2, 3\}$. For comparison, using the corresponding LRC requires $n \log n + \frac{n}{d} \log n$ bits.

Clearly, over an alphabet of size n , the information theoretic bound on the amount of required redundancy for a single-erasure correcting code is $\log n$. Note that with the additional assumption that the string contains d distinct symbols, this bound reduces to $\log(n-d+1)$. The latter bound is achieved by the above schemes for $d \in \{2, 3\}$. We conjecture that for any d , there exists a proper redundancy function f which allows single-erasure correction. That is, the function f provides output from a set of size $n-d+1$, when applied over inputs that contain distinct values.

7 Discussion and Open Problems

In this paper we discussed locality in permutations, motivated by applications in distributed storage and rank modulation coding. We have discussed two aspects of this problem, the combinatorial one and the coding one. In the combinatorial aspect, we discussed locality in permutations without any encoding, and in the coding aspect we allowed the permutation to be encoded in order to obtain this locality.

In the combinatorial aspect, we provided upper and lower bound for the maximal size of a set of permutations with locality, provided several simple constructions with high rate, and several interesting constructions with low rate. In the coding aspect we presented a method of encoding certain permutations in order to obtain locality, and to support arbitrary powers of the permutation. We have concluded with a proof of concept which shows that any permutation may be stored with smaller redundancy than an ordinary string.

Throughout the paper we assumed that low query complexity is to be maintained. Clearly, if no such constraint is assumed, any permutation can be represented using $\lceil \log(n!) \rceil$ bits, and stored using an LRC. However, when a query complexity requirement is imposed, there seems to be much more to be studied, and our results are hardly adequate comparing with the potential possibilities.

For simplicity, we assumed that each node stores a single symbol from $[n]$, and focused on symbol locality. This convention may be adjusted to achieve storage systems with different parameters. E.g., in Subsection 4.2 we considered permutation that can be considered as a concatenation of permutations on h elements. These permutations may alternatively be stored on h nodes, where node i stores all i -th elements of the concatenated permutations. In this system any single node failure may be corrected by downloading the content of all other nodes. To achieve better locality, the data may be partitioned among a larger number of nodes. Techniques such as this open a gateway towards the equivalent of *array codes* for permutations, which constitutes a vast area for future consideration as well.

Finally, we list herein a few specific open problems which were left unanswered in this paper.

1. Close the gap between the upper bound in Theorem 1 and the lower bound in Theorem 3, potentially by using the methods of Theorem 2.
2. Provide an explicit construction of sets with constant locality $d \geq 2$ and optimal rate $\frac{d}{d+1}$. The existence of these sets is guaranteed by Theorem 3.
3. Find the connection between a set of permutations S and its corresponding $\gamma(S)$ (Section 6.2).
4. Find a proper “parity” function f for any d in Section 6.3.
5. Find additional large sets of permutations that have good locality.
6. Explore the locality of permutations under different representation techniques.

7. Endow S_n with a suitable metric, and explore the locality of codes with a good minimum distance by this metric.

References

1. N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 2004.
2. A. Barg and A. Mazumdar, “Codes in permutations and error correction for rank modulation”, *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3158–3165, 2010.
3. M. Bellare, O. Goldreich, and S. Goldwasser, “Incremental cryptography: The case of hashing and signing”, *Advances in Cryptology (CRYPTO)*, Springer Berlin Heidelberg, pp. 216–233, 1994.
4. M. Bellare, O. Goldreich, and S. Goldwasser, “Incremental cryptography and application to virus protection”, *ACM 27th Annual Symposium on Theory of Computing (STOC)*, pp. 45–56, 1995.
5. I. Beyer, T. F. Smith, M. L. Stein, and S. M. Ulam, “Metrics in biology, an introduction”, *Los Alamos Scientific Laboratory report*, LA-4973, 1972.
6. I. F. Blake, G. Cohen, and M. Deza, “Coding with permutations”, *Information and Control*, vol. 43, no. 1, pp. 1–19, 1979.
7. M. Bóna. *Combinatorics of permutations*. CRC Press, 2012.
8. S. Buzaglo and E. Yaakobi, “On the capacity of constrained permutation codes for rank modulation”, *IEEE Transactions on Information Theory*, vol. 62, no. 4, pp. 1649–1666, 2016.
9. W. Chu, C. J. Colbourn, and P. Dukes, “Constructions for permutation codes in powerline communications”, *Designs, Codes and Cryptography*, vol. 32, no. 1-3, pp. 51–64, 2004.
10. C. Cooper, “A lower bound for the number of good permutations”, *Data Recording, Storage and Processing* (Nat. Acad. Sci. Ukraine), vol. 213, pp. 15–25, 2000.
11. M. Deza and T. Huang, “Metrics on permutations, a survey”, *Journal of Combinatorics, Information and System Sciences*, 1998.
12. D. S. Dummit and R. M. Foote. *Abstract algebra*. Englewood Cliffs, N.J.: Prentice Hall, 1991.
13. S. Eberhard, F. Manners, and R. Mrazović, “Additive triples of bijections, or the toroidal semiqueens problem”, *arXiv:1510.05987*, 2015.
14. F. Farnoud, V. Skachek, and O. Milenkovic, “Error-correction in flash memories via codes in the Ulam metric”, *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3003–3020, 2013.
15. R. Gabrys, E. Yaakobi, F. Farnoud, F. Sala, J. Bruck, and L. Dolecek, “Codes correcting erasures and deletions for rank modulation”, *IEEE Transactions on Information Theory*, vol. 62, no. 1, pp. 136–150, 2016.
16. A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, “Rank modulation for flash memories”, *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, 2009.

17. E. Konstantinova, V. Levenshtein, and J. Siemons, “Reconstruction of permutations distorted by single transposition errors”, *arXiv:math/0702191*, 2007.
18. R. J. McEliece, “A public-key cryptosystem based on algebraic coding theory”, *DSN progress report*, vol. 42, no. 44, pp. 114–116, 1978.
19. B. D. McKay, J. C. McLeod, and I. M. Wanless, “The number of transversals in a Latin square”, *Designs, Codes and Cryptography*, vol. 40, no. 3, pp. 269–284, 2006.
20. J. I. Munro, R. Raman, V. Raman, and S. Rao, “Succinct representations of permutations and functions”, *Theoretical Computer Science*, vol. 438, pp. 74–88, 2012.
21. M. Patrascu, “Succincter”, *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 305–313, 2008.
22. N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar, “Optimal linear codes with a local-error-correction property”, *IEEE International Symposium on Information Theory (ISIT)*, pp. 2776–2780, 2012.
23. N. Raviv, E. Yaakobi, and M. Médard, “Coding for locality in reconstructing permutations”, *IEEE International Symposium on Information Theory (ISIT)*, pp. 450–454, 2016.
24. S. E. Rouayheb, S. Goparaju, H. M. Kiah, and O. Milenkovic, “Synchronizing edits in distributed storage networks”, *arXiv:1409.1551*, 2014.
25. M. Schwartz, “Efficiently computing the permanent and Hafnian of some banded Toeplitz matrices”, *Linear Algebra and its Applications*, vol. 430, no. 4, pp. 1364–1374, 2009.
26. I. Tamo and A. Barg, “A family of optimal locally recoverable codes”, *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, 2014.
27. I. Tamo and M. Schwartz, “Correcting limited-magnitude errors in the rank-modulation scheme”, *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2551–2560, 2010.
28. I. Tamo, Z. Wang, and J. Bruck, “Long MDS codes for optimal repair bandwidth”, *IEEE International Symposium on Information Theory (ISIT)*, pp. 1182–1186, 2012.
29. A. Wu, A. Rosenfeld, “Cellular graph automata. I. basic concepts, graph property measurement, closure properties”, *Information and Control*, vol. 42, no. 3, pp. 305–329, 1979.

Acknowledgements The work of Netanel Raviv was supported in part by the Aharon and Ephraim Katzir study grant, the IBM Ph.D. fellowship, and the Israeli Science Foundation (ISF), Jerusalem, Israel, under Grant no. 10/12. This work was done while he was a visiting student at MIT, under the supervision of Prof. Médard. The work of Eitan Yaakobi was supported in part by the Israeli Science Foundation (ISF), Jerusalem, Israel, under grant no. 1624/14. The authors would like to thank Ahmad Beirami for fruitful discussions.