Rewrite Cost Optimal Rank Modulation Codes in S_4 and S_5

Arijit Dutta and Saravanan Vijayakumaran
Department of Electrical Engineering
Indian Institute of Technology Bombay, Mumbai 400076, India
Email: arijit.dutta@iitb.ac.in, sarva@ee.iitb.ac.in

***Abstract*—In this paper, we have found all possible largest permutation codes in S_4 and S_5 under Kendall τ-distance. We consider two rewrite operations namely push-to-the-top and minimal-push-up and give the optimum codes in terms of rewrite cost for both these techniques. These optimum codes can be obtained from a set of relatively smaller size for both the cases. We also give the largest single error correcting Gray code when minimal-push-up is used.**

I. Introduction

Rank modulation (RM) is a data representation technique for flash memories which was proposed to deal with the inherent problems posed by the flash device [1]. It proposes to replace a multiple level cell (MLC) by a group of n cells. In RM, the relative ranking of group of n cells represents one information symbol. One of the important parameters of error correcting codes for RM is rewrite cost. For storing new information, charge levels of cells are increased to change the ranking. This is called data rewrite operation. In a flash cell, storing charge is easy but removal is difficult. After multiple rewrites, when the charge level of one cell reaches to the maximum allowed level, a block of large number of cells has to be erased. It is a costly operation and it reduces the lifetime of the flash device itself. Rewrite cost reflects the increase in charge level due to each rewrite operation. The code having the lowest overall rewrite cost gives the largest number of rewrites before a block erasure is needed. In Figure 1, three codes have been separately used for storing and updating information symbols. Here, we assume that code 1 has the highest and code 3 has the lowest overall rewrite cost. So, code 1 allows the least number (five) of rewrites and code 3 allows the most number (eleven) of rewrites before a block erasure is needed (as one of the cells reaches to its maximum allowed level). Therefore, code 3 is better RM code than code 1 and code 2 in terms of rewrite cost and it is more suitable for practical use.

The main contribution of this paper is to find all possible largest codes of given minimum distance in S_4 and S_5 and to give the optimum codes in terms of rewrite cost. Contributions of this paper are listed in the following:

- We have found all possible largest codes with various minimum Kendall τ-distance in S_4 and S_5 by maximum clique approach.
- We have obtained the set of smallest possible size from which all codes can be found. We have defined this as the set of non-equivalent codes.
- We give an algorithm to compute the rewrite cost when push-to-the-top is used as rewrite operation.
- For both push-to-the-top and minimal-push-up, we give the optimum codes in terms of rewrite cost.
- We show that the optimum code is a member of set of non-equivalent codes for both rewrite operations.
- We also give the largest single error correcting Gray code when minimal-push-up is used.

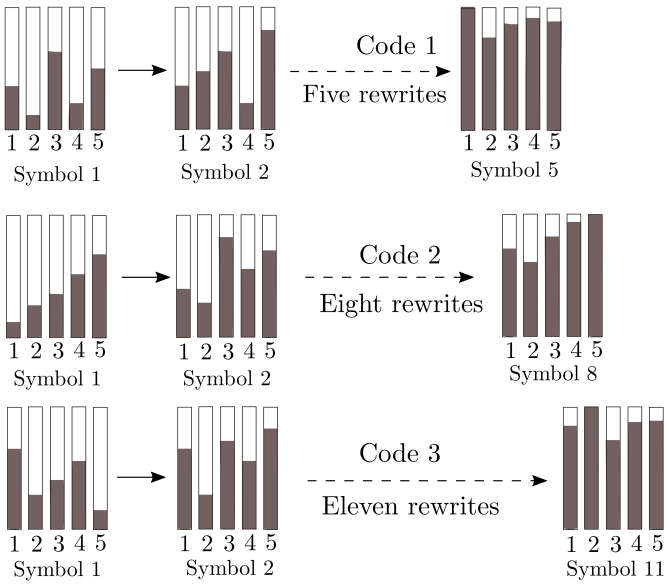

Figure 1: RM codes having different rewrite costs.

II. Basic concepts

As we have mentioned before, the relative ranking of charge levels of n cells represents a symbol in RM. These rankings are permutations in the symmetric group S_n. For example in Figure 1, symbol 1 in code 1 is represented by the permutation $(3, 5, 1, 4, 2) \in S_5$. This is because cell 3 has the highest charge level, cell 5 has the second highest charge level and so on. There are $n!$ number of permutations in S_n. Each permutation is a bijection from $[n] \to [n]$ where $[n] \coloneqq \{1, 2, ..., n\}$. For example, $\sigma = (\sigma(1), \sigma(2), ..., \sigma(n)) \in S_n$ is the bijection $1 \to \sigma(1), 2 \to \sigma(2), ..., n \to \sigma(n)$. The group operation in S_n is the composition of these bijections.

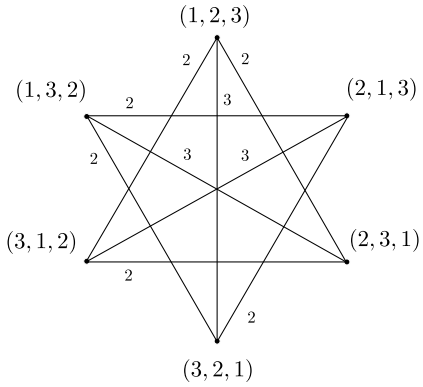

Figure 2: A graph whose vertex set is S_3 with edges between permutations at Kendall τ-distance of 2 and 3 from each other.

One error model in RM is *adjacent transposition*. If we apply adjacent transposition τ_i on $\sigma = (\sigma(1), ..., \sigma(i-1), \sigma(i), \sigma(i+1), \sigma(i+2), ..., \sigma(n)) \in S_n$, we get the permutation $(\sigma(1), ..., \sigma(i-1), \sigma(i+1), \sigma(i), \sigma(i+2), ..., \sigma(n))$. The metric which captures this error model is called the Kendall τ-distance [2]. Given $\sigma, \pi \in S_n$, the Kendall τ-distance $d_k(\sigma, \pi)$ is defined as the minimum number of adjacent transpositions to get π from σ or vice-versa. A closed form expression for the Kendall τ-metric [3] is given by

$$d_k(\sigma, \pi) = |\{(i, j) : \sigma^{-1}(i) < \sigma^{-1}(j) \wedge \pi^{-1}(i) > \pi^{-1}(j)\}|. \quad (1)$$

The Kendall τ-metric is invariant to right multiplication [4], where right multiplication is outer composition. For example, let $\sigma, \pi \in S_n$. Then $\sigma * \pi \coloneqq \pi(\sigma(i))$ for $i \in [n]$. The property says that for any $\sigma, \pi, \omega \in S_n$,

$$d_k(\sigma * \omega, \pi * \omega) = d_k(\sigma, \pi). \quad (2)$$

A permutation code under the Kendall τ-metric is a subset of S_n where the Kendall τ-distance between any two elements is at least d. Here d is the minimum distance of the code. Let $P(n, d)$ represents the size of the largest permutation code in S_n with minimum Kendall τ-distance d. It is defined as

$$P(n, d) \coloneqq \max\{|C| \mid C \subseteq S_n, |C| \geq 2, d_{min}(C) \geq d\}. \quad (3)$$

RM is likely to be implemented for smaller values of n. Therefore, finding codes of size $P(n, d)$ for smaller n has become an interesting problem. Buzaglo *et al.* [3] gives the bounds on $P(n, d)$ for $n = 5, 6, 7$. Vijayakumaran [5] gives the exact values of $P(n, d)$ for $n = 5, 6$ and codes which have size $P(n, d)$. Here, the author has taken the integer programming approach which gives one possible code of size $P(n, d)$.

A well known technique for finding all possible largest code is by finding maximum clique of a graph [6]. Let us consider a simple undirected graph $G = (V, E)$ with the node set V and the edge set $E \subseteq V \times V$. A clique C is a subset of V where the nodes are pairwise adjacent i.e. $C \subseteq V$ such that $(i, j) \in E$ for all $i, j \in C$. A maximum clique of G is a clique of maximum possible cardinality. An example is shown in Figure 2. Here, $G = (V, E)$, where $V = S_3$ and $E = \{(\sigma, \pi) \in S_n \times S_n \mid d_k(\sigma, \pi) \geq 2\}$. Corresponding Kendall τ-distances between any two permutations sharing an edge, are shown. There are multiple cliques of size 2 and 3. Size of a maximum clique is 3. $\{(1, 2, 3), (3, 2, 1)\}$, $\{(1, 3, 2), (2, 3, 1)\}$, $\{(3, 1, 2), (2, 3, 1)\}$ are cliques of size two. Maximum cliques are $\{(1, 2, 3), (3, 1, 2), (2, 3, 1)\}$, $\{(1, 3, 2), (3, 2, 1), (2, 1, 3)\}$. Notice that these maximum cliques are subsets of S_3 such that the Kendal τ-distance between any two elements is two or more. So by (3), they are codes of size $P(3, 2)$. If we remove all the edges with Kendall τ-distance 2, then there will be cliques of size two only. They will be maximum cliques in this case. By similar arguments, they represent codes of size $P(3, 3)$. In this case, the edge set $E = \{(\sigma, \pi) \in S_n \times S_n \mid d_k(\sigma, \pi) \geq 3\}$.

III. Enumeration of codes of size $P(4, d)$ and $P(5, d)$

Consider the graph $G = (S_n, E)$. We define E as

$$E = \{(\sigma, \pi) \in S_n \times S_n \mid d_k(\sigma, \pi) \geq d\}. \quad (4)$$

A maximum clique of G is a code of size $P(n, d)$. This is because a maximum clique in this case is a subset of S_n of maximum possible cardinality where the distance between any two elements of it is at least d. As maximum cliques in a graph are not unique, we can have many codes of size $P(n, d)$ for a given n and d. Cliquer [7] is an open source software which finds clique of a given graph. Using Cliquer, we have enumerated all possible largest codes in S_4 and S_5. By Theorem 10 of [3], we know

$$P(n, d) = 2 \quad \text{for} \quad d > \frac{2}{3}\binom{n}{2}. \quad (5)$$

So, $P(3, d) = 2$ for $d > 3$. We found all possible codes of size $P(n, d)$ by Cliquer for $d \in \{3, 4\}$ in S_4 and for $d \in \{3, 4, 5, 6\}$ in S_5. Cliquer could not give us results in S_6.

Kendall τ-metric is invariant to right multiplication. So, given a code of size $P(n, d)$, if we multiply to the right its elements with any $\sigma \in S_n$, we get a new code of size $P(n, d)$ with the same relative distances among codewords. We say that these two codes are equivalent. We observed that many codes found by Cliquer can be obtained from one another by right multiplication. So, among this abundance of codes of size $P(n, d)$ in S_n, we have found which of them are not equivalent i.e. one that cannot be obtained by right multiplication by any element in S_n from any other codes. Collectively we define them as the set of non-equivalent codes. Figure 3 gives an illustration. Here, $C_1 = \{c_1, c_2, c_3, c_4\}$ is a member of the set of non-equivalent codes. We get another code $C_2 = \{c'_1, c'_2, c'_3, c'_4\}$ by right multiplication by some $\sigma \in S_n$ i.e. $c'_i = c_i * \sigma$ for $i \in [4]$. Owing to the right invariance of Kendall τ-metric, we have

$$d_k(c_i, c_j) = d_k(c'_i, c'_j) \quad \text{for} \quad \forall (i, j) \in [4] \times [4]. \quad (6)$$

We say code C_1 and code C_2 are *equivalent*. Any code of size $P(n, d)$ can be obtained from this set of non-equivalent codes by right multiplication by some $\sigma \in S_n$.

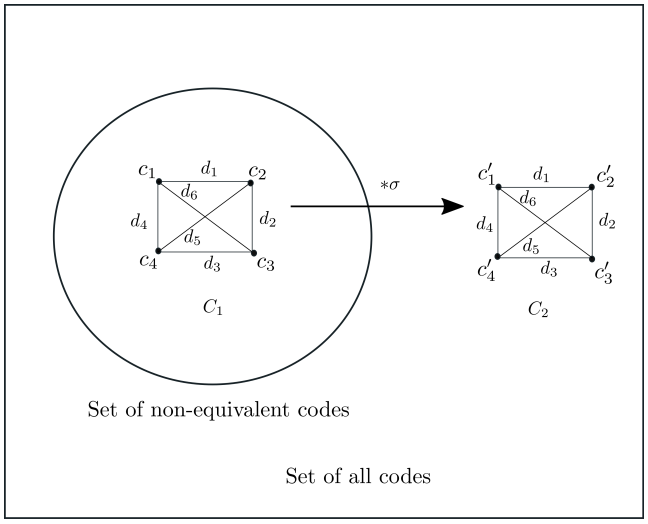

Figure 3: Set of non-equivalent codes.

n	d	$P(n,d)$	$L(n,d)$	$L'(n,d)$	$S(n,d)$
4	3	5	48	2	0
4	4	3	40	3	2
5	3	20	3192	38	1
5	4	12	51280	443	0
5	5	6	8160	68	0
5	6	5	168	3	2

Table I: Observation on largest codes in S_4 and S_5.

Let $L(n,d)$ denotes the total number of largest codes in S_n with minimum distance d, $L'(n,d)$ denotes the number of non-equivalent largest codes in S_n with minimum distance d, and $S(n,d)$ denotes the number of largest codes present in the set of non-equivalent codes which are subgroups of S_n. Table I shows our observation.

As shown in Table I, a large number of codes can be generated from the set of non-equivalent codes. We have observed that all codes of size $P(n,d)$ have the same minimum distance d for all n and d. Weight distribution of codes are sometimes different. However, comparison on weight distribution can be done on set of non-equivalent codes instead of set of all codes of size $P(n,d)$.

IV. Best Code in Terms of Rewrite Cost for Minimal-Push-Up Operation

There are multiple ways to choose the *best* code among all possible codes. One possibility could be choosing the code which gives optimum error performance. However we could not find any proper channel model for RM in the literature to do this analysis. In this paper, we find the optimum codes in terms of rewrite cost. As discussed in Section I, during data rewrite operation in RM, the charge levels of cells are monotonically increased. The data rewrite operation for RM proposed by Jiang *et al.* [1] is called push-to-the-top operation. In their work, they have given the scheme for constructing the code which gives the optimal performance in terms of rewrite cost when there is random data modification. However they did not consider error correction. Therefore, finding rewrite cost optimal error correcting codes for RM is an interesting problem. En Gad *et al.* [8] have proposed a better rewrite technique for RM namely minimal-push-up operation. Let us consider both the rewrite techniques. Suppose the present state of a group of 4 cells is $(2,1,3,4)$ and we want to change it to $(2,1,4,3)$. For minimal-push-up, the charge level of cell 4 is to be increased above only the charge level of cell 3. For push-to-the-top technique, the charge level of cell 4 is to be increased above the highest charge level i.e. the charge level of cell 2 in this case. Similarly, we move from back to the front and increase the charge level of cell 1 above cell 3 and 4 (if it is already not) and so on.

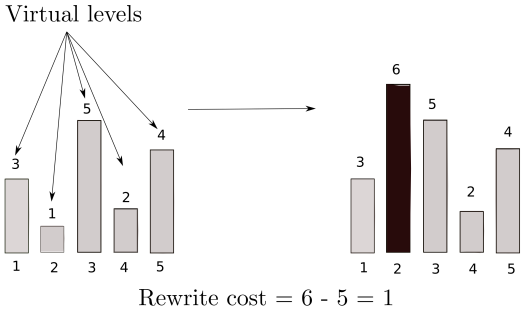

Figure 4: Virtual level and rewrite cost.

We can only do data rewrite on a code till a cell reaches it's physical limit of charge level. After that the whole block of cells is to be erased. We refer to the interval before a costly block erasure process is needed as *operation period*. Out of all these codes, the code which gives the longest operation period, gives maximum rewrites before block erasure. This code has the minimum average rewrite cost (to be defined shortly) and hence is the optimum code in terms of rewrite cost.

Exact increase in charge level after rewrite operation is a real random variable. We do not know its value. To quantify the improvement of minimal-push-up over push-to-the-top, En Gad *et al.* [8] have introduced the concept of *virtual level* of charge of the group of cells. Further, they have defined the cost of rewrite as the difference between the highest virtual levels of the cells before and after rewrite. Figure 4 gives an example of virtual level and rewrite cost. Given a group of five cells, the initial state is $(3,5,1,4,2)$ and we need to change it to $(2,3,5,1,4)$. Clearly, the charge level of cell 2 has to be pushed above the charge level of cell 3 in this case. The rewrite cost in this case is the difference between the highest virtual levels before and after rewrite i.e. $6-5=1$. En Gad *et al.* proposed an algorithm for calculating rewrite cost for any two given permutations when minimal-push-up operation is done for rewrite. We reproduce their algorithm in the following.

Let $u, v \in S_n$ and l_i denotes the virtual level of cell i. Let the cost of rewrite from u to v is given by $\phi(u \to v)$. Then the algorithm to calculate $\phi(u \to v)$ is:

1) For $i = 1, 2, ..., n$ do:
   $l_{u(i)} \leftarrow n + 1 - i$

n	d	Minimum ARC	Maximum ARC	Best Code
4	3	1.76	1.76	$\mathcal{C}^{\text{mpu}}_{(4,3)}$
4	4	1.55	1.66	$\mathcal{C}^{\text{mpu}}_{(4,4)}$
5	3	2.55	2.60	$\mathcal{C}^{\text{mpu}}_{(5,3)}$
5	4	2.51	2.58	$\mathcal{C}^{\text{mpu}}_{(5,4)}$
5	5	2.39	2.50	$\mathcal{C}^{\text{mpu}}_{(5,5)}$
5	6	2.40	2.44	$\mathcal{C}^{\text{mpu}}_{(5,6)}$

Table II: Best codes in S_4 and S_5 in terms of minimum ARC for minimal-push-up operation.

2) For $i = n-1, n-2, ..., 1$ do:
$l_{v(i)} \leftarrow \max\{l_{v(i+1)} + 1, l_{v(i)}\}$
3) $\phi(u \to v) = l_{v(1)} - n$.

Using this cost function, we define average rewrite cost of a given code as follows.

Let us consider that there is one rewrite operation. We assume that all the codewords of the given code ($\mathcal{C}$) are equiprobable to be chosen to represent the initial data and the data after rewrite. So for any $x, y \in \mathcal{C}$, the probability that x is chosen as the first codeword and y is chosen as the next codeword for rewrite is

$$p(x, y) = p(x)p(y) = \frac{1}{|\mathcal{C}|^2}. \tag{7}$$

The cost of rewrite from codeword x to codeword y is given by $\phi(x \to y)$. Clearly, when $x = y$, $\phi(x \to y) = 0$. Now we define Average Rewrite Cost (ARC) of the given code $\mathcal{C}$ as:

$$\begin{aligned} \text{ARC}(\mathcal{C}) &:= \sum_{x\in\mathcal{C}}\sum_{y\in\mathcal{C}} p(x, y)\phi(x \to y) \\ &= \frac{1}{|\mathcal{C}|^2}\sum_{x\in\mathcal{C}}\sum_{y\in\mathcal{C}} \phi(x \to y). \end{aligned} \tag{8}$$

If we assume that probability $p(x, y)$ is same for every rewrite, then the same definition of ARC is applicable for multiple rewrites.

We compare all the cliques representing codes of size $P(n, d)$ and choose the code that gives us minimum ARC. If we use this code then we will get the longest operation period before a costly cell erase operation is needed. This is because the rewrite cost reflects the change in the highest charge levels before and after rewrite. Higher the cost the sooner we need a costly cell erase operation as the highest charge level reaches the maximum allowed level. Table II shows the observation we have made. Here, the minimum ARC is same as the maximum ARC for $n = 4$, $d = 3$. So, every code is optimal in terms of rewrite cost in this case. We also do not observe much difference between the minimum and the maximum ARC. However, for enterprise uses where the number of rewrite and block erasure is huge, optimum codes might be very useful. The best codes are given in the appendix. We observe the same minimum ARC when we consider the set of non-equivalent codes for each n and d. This is because of the following lemma.

Lemma 1. *The cost function $\phi(u \to v)$ is invariant to right multiplication i.e. $\phi(u \to v) = \phi(u*w \to v*w)$ for all $u, v, w \in S_n$.*

Proof: From theorem 1 of En Gad *et al.* [8], we know $\phi(u \to v) = \max_{i\in[n]}(v^{-1}(i) - u^{-1}(i))$. The cost is the maximal increase in rank among cells (for example, increase in rank of cell 1 for $u \to v$ is $v^{-1}(1) - u^{-1}(1)$).

Now we claim that, the rank of cell i in v is equal to the rank of cell $w(i)$ in $v * w$. This is true because the rank of cell i in v is $v^{-1}(i)$

$$\begin{aligned} &= v^{-1}(w^{-1}(w(i))) \\ &= (w^{-1} * v^{-1})(w(i)) \\ &= (v * w)^{-1}(w(i)) \\ &= \text{rank of cell } w(i) \text{ in } v * w. \end{aligned}$$

Hence, the increase in rank for cell i in $u \to v$ is equal to the increase in rank for cell $w(i)$ in $u * w \to v * w$. As w is a bijection over $[n] \to [n]$, the cost function ϕ i.e. the maximal increase in rank among cells remains the same. ■

Every possible code of size $P(4, d)$ and $P(5, d)$ can be obtained by right multiplication from the set of non-equivalent codes. So the set of non-equivalent codes gives us the code with the minimum ARC.

A. Largest Single Error Correcting Gray Code

The Gray code proposed by Jiang *et al.* [1] is a sequence of elements in S_n, where we can traverse from one element to the element adjacent to it by a single push-to-the-top operation. The traversing in Gray code represents the gradual increase in the charge level of an MLC by the smallest possible amount. Error correcting Gray code is a Gray code where the distance between any two elements of the code is lower bounded by some positive integer (say d). For $d = 2$, such Gray codes are called *snake-in-the-box codes* [9]. In snake-in-the-box codes, the next codeword is obtained from the current codeword by a single push-to-the-top operation and it detects a single error. We have found the largest error correcting Gray code where the minimum distance of the code is 3. Therefore it can correct a single error. Here, the rewrite operation is minimal-push-up and adjacent elements can be obtained from one to the next incurring unit rewrite cost.

We propose to define a Gray code as a sequence of elements of S_n where the next element (say y) can be written from the current element (say x) by minimal-push-up technique incurring unit rewrite cost i.e. $\phi(x \to y) = 1$. We observe that there exists an order in $\mathcal{C}^{\text{mpu}}_{(5,3)}$ (given in appendix) following which we obtain a Gray code as defined above. If we read in $\mathcal{C}^{\text{mpu}}_{(5,3)}$ row-wise from 1 to 20 i.e. the 1st element is (1, 2, 3, 4, 5), 2nd element is (4, 5, 1, 2, 3), 5th element is (1, 5, 2, 4, 3) and so on, then the Gray sequence is $\{13, 8, 1, 20, 18, 15, 5, 16, 9, 4, 2, 3, 19, 11, 14, 6, 7, 17, 10, 12\}$. This Gray code consists of all the elements of $\mathcal{C}^{\text{mpu}}_{(5,3)}$. Therefore, this is the largest single error correcting Gray code in S_5

under Kendall τ-distance. We did not find any other Gray code of significant length.

V. Best code in terms of rewrite cost for push-to-the-top operation

In the case of push-to-the-top operation, the rewrite cost of changing one permutation to the other is defined as the minimum number of push-to-the-top operations needed for the change [1]. Let $u, v \in S_n$ and l_i denotes the virtual level of cell i. Let the minimum number of push-to-the-top operations needed to change u to v be minptt($u \to v$). Now, we propose an algorithm as follows.

1) For $i = 1, 2, ..., n$ do:
   $l_{u(i)} \leftarrow n + 1 - i$
2) For $i = n-1, n-2, ..., 1$ do:
   if $l_{v(i)} > l_{v(i+1)}$
   continue (2)
   else
   $l_{v(i)} \leftarrow \max\{(\max_{j \in [n] \setminus \{v(i)\}} l_j + 1), l_{v(i)}\}$
3) $\xi(u \to v) = l_{v(1)} - n$.

Example 1. *Let $u = (1, 2, 3, 4)$ and $v = (2, 1, 4, 3)$. From step 1 of the algorithm, we have $l_1 = 4$, $l_2 = 3$, $l_3 = 2$, $l_4 = 1$. From step 2, we have the following updates in the values of l_is.*

$$\begin{aligned} l_4 &\not> l_3 \implies l_4 = 5 \\ l_1 &\not> l_4 \implies l_1 = 6 \\ l_2 &\not> l_1 \implies l_2 = 7. \end{aligned}$$

So we have

$$\xi(u \to v) = l_{v(1)} - n = l_2 - 4 = 3.$$

We observe that it takes at least 3 push-to-the-top operations to get v from u as follows:

$$(1, 2, 3, 4) \to (4, 1, 2, 3) \to (1, 4, 2, 3) \to (2, 1, 4, 3)$$

So minptt($u \to v$) is also 3 in this case.

Theorem 1. *$\xi(u \to v)$ = minptt($u \to v$)*

Proof: According to the algorithm, if for some i, the virtual level of $v(i)$ is less than the virtual level of $v(i+1)$, then cell i is pushed to the top. Total number of such pushes is given by $\xi\ (u \to v) = l_{v(1)} - n$. This has to be done at least so that we can transform u to v. So, $\xi(u \to v) \leq$ minptt($u \to v$).

But by the nature of the algorithm, $l_{v(i)}$ is increased every time after the first push-to-the-top operation. So, $\xi(u \to v) \geq$ minptt($u \to v$). Hence, $\xi(u \to v) =$ minptt($u \to v$). ■

When we use push-to-the-top operation, minptt($u \to v$) = $\xi(u \to v)$ is the rewrite cost for changing u to v. Using this cost function we calculate ARC of a code using a process similar to the one described in section IV. Table III shows our observation. In this case also we do not observe much difference between the minimum and the maximum ARC. However, for enterprise uses where the numbers of rewrites and block erasures are huge, optimum codes might be very useful.

n	d	Minimum ARC	Maximum ARC	Best Code
4	3	1.96	2.00	$\mathcal{C}^{\text{ptt}}_{(4,3)}$
4	4	1.67	2.00	$\mathcal{C}^{\text{ptt}}_{(4,4)}$
5	3	3.13	3.23	$\mathcal{C}^{\text{ptt}}_{(5,3)}$
5	4	3.05	3.16	$\mathcal{C}^{\text{ptt}}_{(5,4)}$
5	5	2.75	3.03	$\mathcal{C}^{\text{ptt}}_{(5,5)}$
5	6	2.60	2.80	$\mathcal{C}^{\text{ptt}}_{(5,6)}$

Table III: Best codes in S_4 and S_5 in terms of minimum ARC for push-to-the-top operation.

Best codes are given in the appendix. Also we get the same minimum ARC if we consider the set of all codes instead of the set of non-equivalent codes as before. This is because of the following lemma.

Lemma 2. *minptt($u \to v$) = minptt($u * w \to v * w$) for any $u, v, w \in S_n$.*

Proof: Let $u = (u(1), u(2), ..., u(n))$, $v = (v(1), v(2), ..., v(n))$, and $w = (w(1), w(2), ..., w(n))$. Suppose, we need a push-to-the-top operation on cell i for $u \to v$. Then the following conditions have to be satisfied

$$u(i) = v(1) \tag{9}$$
$$u(j) = v(j+1) \text{ for } 2 \leq j < i \tag{10}$$
$$u(j) = v(j) \text{ for } i < j \leq n \tag{11}$$

Now, let j be a positive integer such that $2 \leq j < i$. Then,

$$\begin{aligned} (u * w)(j) &= w(u(j)) \\ &= w(v(j+1)) \\ &= (v * w)(j+1). \end{aligned}$$

Therefore, Condition 10 is satisfied if we multiply w with u and v. Similarly, other conditions are also satisfied for the corresponding values of j. Hence, we need push-to-the-top operation on cell i for $u * w \to v * w$ similar to $u \to v$. This argument works for any conditions on $u(i)$s and $v(i)$s corresponding to the push-to-the-top operations requirement for $u \to v$. So we conclude that, the same push-to-top operations are needed for $u * w \to v * w$ and $u \to v$. Therefore, minptt($u \to v$) = minptt($u * w \to v * w$) for any $u, v, w \in S_n$. ■

Following the same argument made in section IV, we conclude that the set of non-equivalent code gives the minimum ARC in case of push-to-the-top also.

VI. Conclusion

In this work, we have enumerated all possible permutation codes of size $P(n, d)$ in S_4 and S_5 by using the software Cliquer [7]. We reduce the set of all possible codes to the set of non-equivalent codes. All possible codes can be obtained by right multiplication from this set of non-equivalent codes. We have shown that the code that gives minimum ARC belongs to this smaller set for both the cases when push-to-the-top and

minimal-push-up are used. We also give the largest single error correcting Gray code in S_5 when minimal-push-up is used.

ACKNOWLEDGEMENTS

We would like to acknowledge the support of the Bharti Centre for Communication at IIT Bombay which made this work possible.

APPENDIX

This appendix contains all the rewrite cost optimal codes (code with minimum ARC) which appear in Table II and Table III.

A. Optimum codes for minimal-push-up

- Code $\mathcal{C}^{\text{mpu}}_{(4,3)}$ is

  (1, 2, 3, 4) (3, 1, 4, 2) (3, 2, 4, 1)

  (2, 4, 1, 3) (4, 1, 3, 2)
- Code $\mathcal{C}^{\text{mpu}}_{(4,4)}$ is

  (1, 2, 3, 4) (3, 4, 1, 2) (4, 2, 1, 3)
- Code $\mathcal{C}^{\text{mpu}}_{(5,3)}$ is

  (1, 2, 3, 4, 5) (4, 5, 1, 2, 3) (4, 3, 5, 1, 2)

  (4, 1, 2, 3, 5) (1, 5, 2, 4, 3) (5, 3, 2, 1, 4)

  (2, 5, 3, 4, 1) (2, 3, 4, 1, 5) (1, 4, 3, 5, 2)

  (3, 4, 2, 5, 1) (5, 1, 4, 3, 2) (3, 1, 4, 2, 5)

  (3, 2, 1, 5, 4) (3, 5, 1, 4, 2) (5, 2, 4, 1, 3)

  (1, 3, 5, 2, 4) (4, 2, 5, 3, 1) (2, 4, 1, 5, 3)

  (5, 4, 3, 2, 1) (2, 1, 5, 3, 4)
- Code $\mathcal{C}^{\text{mpu}}_{(5,4)}$ is

  (1, 2, 3, 4, 5) (2, 3, 4, 5, 1) (3, 4, 5, 1, 2)

  (2, 4, 1, 5, 3) (2, 5, 1, 3, 4) (4, 5, 2, 3, 1)

  (4, 1, 5, 3, 2) (1, 5, 3, 2, 4) (5, 3, 2, 4, 1)

  (3, 2, 1, 5, 4) (4, 3, 2, 1, 5) (5, 1, 4, 2, 3)
- Code $\mathcal{C}^{\text{mpu}}_{(5,5)}$ is

  (1, 2, 3, 4, 5) (5, 3, 4, 1, 2) (1, 4, 5, 3, 2)

  (3, 2, 5, 1, 4) (4, 3, 2, 1, 5) (5, 2, 1, 4, 3)
- Code $\mathcal{C}^{\text{mpu}}_{(5,6)}$ is

  (1, 2, 3, 4, 5) (4, 1, 5, 3, 2) (3, 4, 2, 5, 1)

  (5, 3, 1, 2, 4) (2, 5, 4, 1, 3)

B. Optimum codes for push-to-the-top

- Code $\mathcal{C}^{\text{ptt}}_{(4,3)}$ is

  (1, 2, 3, 4) (3, 1, 4, 2) (3, 2, 4, 1)

  (2, 4, 1, 3) (4, 1, 3, 2)
- Code $\mathcal{C}^{\text{ptt}}_{(4,4)}$ is

  (1, 2, 3, 4) (3, 4, 1, 2) (4, 2, 1, 3)
- Code $\mathcal{C}^{\text{ptt}}_{(5,3)}$ is

  (1, 2, 3, 4, 5) (3, 4, 5, 1, 2) (4, 5, 1, 2, 3)

  (5, 1, 2, 3, 4) (2, 5, 3, 4, 1) (2, 3, 4, 1, 5)

  (4, 1, 3, 5, 2) (3, 5, 2, 4, 1) (3, 1, 4, 2, 5)

  (5, 3, 1, 4, 2) (3, 2, 1, 5, 4) (1, 5, 4, 3, 2)

  (1, 4, 2, 5, 3) (5, 2, 4, 1, 3) (1, 3, 5, 2, 4)

  (4, 2, 5, 3, 1) (2, 4, 1, 5, 3) (4, 3, 2, 1, 5)

  (5, 4, 3, 2, 1) (2, 1, 5, 3, 4)
- Code $\mathcal{C}^{\text{ptt}}_{(5,4)}$ is

  (1, 2, 3, 4, 5) (2, 3, 4, 5, 1) (3, 4, 5, 1, 2)

  (2, 4, 1, 5, 3) (2, 5, 1, 3, 4) (4, 5, 2, 3, 1)

  (4, 1, 5, 3, 2) (1, 5, 3, 2, 4) (5, 3, 2, 4, 1)

  (3, 2, 1, 5, 4) (4, 3, 2, 1, 5) (5, 1, 4, 2, 3)
- Code $\mathcal{C}^{\text{ptt}}_{(5,5)}$ is

  (1, 2, 3, 4, 5) (3, 4, 2, 1, 5) (3, 5, 1, 4, 2)

  (2, 5, 3, 1, 4) (4, 5, 2, 3, 1) (1, 5, 4, 2, 3)
- Code $\mathcal{C}^{\text{ptt}}_{(5,6)}$ is

  (1, 2, 3, 4, 5) (4, 1, 5, 3, 2) (3, 4, 2, 5, 1)

  (5, 3, 1, 2, 4) (2, 5, 4, 1, 3)

REFERENCES

[1] 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, June 2009.

[2] M. Kendall and J. D. Gibbons, *Rank Correlation Methods*, New York: Oxford Univ. Press, 1990.

[3] S. Buzaglo and T. Etzion, "Bounds on the size of permutation codes with the Kendall τ-metric," *IEEE Transactions on Information Theory*, vol. 61, no. 6, pp. 3241–3250, June 2015.

[4] M. Deza and H. Huang, "Metrics on permutations, a survey," in *J. Combinat., Inf., Syst. Sci.*, 1998, vol. 23, pp. 173–185.

[5] S. Vijayakumaran, "Largest permutation codes with the Kendall τ-metric in S_5 and S_6," *IEEE Communications Letters*, vol. 20, no. 10, pp. 1912–1915, October 2016.

[6] I. M. Bomze, M. Budinich, P. M. Pardolos, and M. Pellilo, *The maximum clique problem*, in: D.-Z. Du and P. M. Pardolos (Eds.), Handbook of Combinatorial Optimization, Supplement Volume A, Kluwer, Dordrecht, 1999, pp. 1-74.

[7] S. Niskanen and P. R. J. Östergård, *Cliquer User's Guide, Version 1.0*, Communications Laboratory, Helsinki University of Technology, Espoo, Finland, Tech. Rep. T48, 2003.

[8] E. En Gad, A. Jiang, and J. Bruck, "Compressed encoding for rank modulation," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, July 2011, pp. 884–888.

[9] Y. Yehezkeally and M. Schwartz, "Snake-in-the-box codes for rank modulation," *IEEE Transactions on Information Theory*, vol. 58, no. 8, pp. 5471–5483, August 2012.