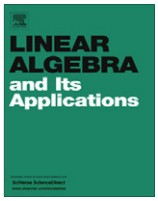
Linear Algebra and its Applications 437 (2012) 324–332

Contents lists available at SciVerse ScienceDirect

Linear Algebra and its Applications

journal homepage: www.elsevier.com/locate/laa



Computing the ball size of frequency permutations under Chebyshev distance ☆

Min-Zheng Shieh, Shi-Chun Tsai *

Department of Computer Science, National Chiao Tung University, 1001 University Road, Hsinchu City, Taiwan

ARTICLE INFO

Article history:
Received 6 December 2011
Accepted 18 February 2012
Available online 18 March 2012

Submitted by R. Brualdi

AMS classification:
15A15
15A36

Keywords:
Permanent
Permutation
Coding theory
Sphere-packing

ABSTRACT

Let S_n^λ be the set of all permutations over the multiset $\{\overbrace{1,\ldots,1}^{\lambda},\ldots,\overbrace{m,\ldots,m}^{\lambda}\}$ where $n = m\lambda$. A frequency permutation array (FPA) of minimum distance d is a subset of S_n^λ in which every two elements have distance at least d. FPAs have many applications related to error correcting codes. In coding theory, the Gilbert–Varshamov bound and the sphere-packing bound are derived from the size of balls of certain radii.

We propose two efficient algorithms that compute the ball size of frequency permutations under Chebyshev distance. Here it is equivalent to computing the permanent of a special type of matrix, which generalizes the Toepliz matrix in some sense. Both methods extend previous known results. The first one runs in $O\left(\binom{2d\lambda}{d\lambda}^{2.376}\log n\right)$ time and $O\left(\binom{2d\lambda}{d\lambda}^2\right)$ space. The second one runs in $O\left(\binom{2d\lambda}{d\lambda}\binom{d\lambda+\lambda}{\lambda}\frac{n}{\lambda}\right)$ time and $O\left(\binom{2d\lambda}{d\lambda}\right)$ space. For small constants λ and d, both are efficient in time and use constant storage space.

☆ The research was supported in part by the National Science Council of Taiwan under contracts NSC-97-2221-E-009-064-MY3 and NSC-98-2221-E-009-078-MY3. A preliminary version of this paper was presented in 2011 IEEE International Symposium on Information Theory.

* Corresponding author.
E-mail addresses: mzshieh@nctu.edu.tw (M.-Z. Shieh), sctsai@cs.nctu.edu.tw (S.-C. Tsai).

0024-3795/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.laa.2012.02.016

1. Introduction

Frequency permutation arrays (FPAs) of frequency λ and length n are sets of permutations over the multiset $\{\overbrace{1,\ldots,1}^{\lambda},\ldots,\overbrace{m,\ldots,m}^{\lambda}\}$ where $n = m\lambda$. In particular, FPAs of frequency $\lambda = 1$ are called *permutation arrays* (PAs). FPA is a special case of Slepian's codes [11] for permutation modulation. Many applications of FPAs in various areas, such as power line communication (see [9,14–16]), multi-level flash memories (see [3,4,12]) and computer security (see [13]), are found recently. In many applications, we use FPAs as error correcting codes. It is well-known that the capability against errors of a code is mainly determined by its minimum distance. Similar to traditional codes, the minimum distance of an FPA C is d under metric δ if $\min_{\rho,\pi\in C;\rho\neq\pi}\delta(\rho,\pi) = d$. A (λ, n, d)-FPA under some metric δ is a FPA of frequency λ and length n which has minimum distance at least d under δ. A (λ, n, d)-FPA is often considered to be better if it has larger cardinality. When we evaluate the quality of a certain design of (λ, n, d)-FPA, we often compare it with the maximum cardinality of (λ, n, d)-FPAs. Generally speaking, computing the maximum size of codes of certain parameters is hard. In coding theory, the Gilbert–Varshamov bound and the sphere-packing bound are famous lower and upper bounds on the code size, respectively. They are derived from the size of balls of certain radii. We focus on the efficiency of computing the ball size in this paper.

Shieh and Tsai [13] showed that the cardinality of d-radius balls can be obtained by computing the permanent of a matrix. It is #P-complete to compute the permanent of a general matrix. However, the matrix for estimating the ball size of FPAs has a special structure. Kløve [5] used the property and proposed a method to efficiently compute the cardinality of balls of radius 1 by solving recurrence relations. Meanwhile, based on Schwartz's result [10], one can compute the size of d-radius balls in $O\left(\binom{2d}{d}^{2.376}\log n\right)$ time for $\lambda = 1$.

In this paper, we give two algorithms to compute the cardinality of a d-radius ball under Chebyshev distance for $d > 1$ and $\lambda > 1$. The first one runs in $O\left(\binom{2d\lambda}{d\lambda}^{2.376}\log n\right)$ time and $O\left(\binom{2d\lambda}{d\lambda}^{2}\right)$ space, and the second one in $O\left(\binom{2d\lambda}{d\lambda}\binom{d\lambda+\lambda}{\lambda}\frac{n}{\lambda}\right)$ time and $O\left(\binom{2d\lambda}{d\lambda}\right)$ space. These algorithms are generalization of Schwartz's result [10]. They are efficient in time and space when d and λ are small fixed constants.

The rest of the paper is organized as follows. In Section 2, we define some notations. In Section 3, we introduce a recursive algorithm EnumV for enumerating permutations in a d-radius ball. Then, based on the property of EnumV, we give two methods to obtain the ball size efficiently in Section 4. We compare previous results with ours in Section 5. Then, we conclude this paper briefly.

2. Notations

We set $n = m\lambda$ throughout the work unless stated otherwise. For positive integers a and b with $a < b$, $[a]$ represents the set $\{1,\ldots,a\}$ and $[a, b]$ represents $\{a, a+1,\ldots,b-1, b\}$. For convenience, $(-\infty, b]$ represents the set of integers at most b. The Chebyshev distance of two k-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as $d_{\max}(\boldsymbol{x},\boldsymbol{y}) = \max_{i\in[k]}|x_i - y_i|$, where x_i and y_i are the ith entries of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. For permutations $\boldsymbol{x}$ and $\boldsymbol{y}$, they are said to be d-close to each other under metric $\delta(\cdot,\cdot)$ if $\delta(\boldsymbol{x},\boldsymbol{y}) \leq d$. We use S_n^{λ} to denote the set of all frequency permutations with each symbol appearing λ times. The identity frequency permutation $\boldsymbol{e}$ in S_n^{λ} is $(\overbrace{1,\ldots,1}^{\lambda},\ldots,\overbrace{m,\ldots,m}^{\lambda})$, i.e., the ith entry of $\boldsymbol{e}$ is $e_i = \left\lceil\frac{i}{\lambda}\right\rceil$. A *partial frequency permutation* can be derived from a frequency permutation in S_n^{λ} with some entries replaced with $*$. The symbol $*$ does not contribute to the distance. I.e., the distance between two k-dimensional partial frequency permutations, $\boldsymbol{x}$ and $\boldsymbol{y}$, is defined as $d_{\max}(\boldsymbol{x},\boldsymbol{y}) = \max_{i\in[k],x_i\neq *,y_i\neq *}|x_i - y_i|$.

Under Chebyshev distance, a ball of radius r centered at π is defined as $B(r,\pi) = \{\rho : d_{\max}(\rho,\pi) \le r\}$. We can obtain $B(r,\pi)$ from $B(r,\boldsymbol{e})$ by rearranging the indices of the entries, therefore $|B(r,\pi)| = |B(r,\boldsymbol{e})|$ for $\pi \in S_n^\lambda$. Let $V_{\lambda,n,d}$ be the size of a ball of radius d in S_n^λ under Chebyshev distance.

3. Enumerate *d*-close permutations

In this section, we give a recursive algorithm to enumerate all frequency permutations in $B(d,\boldsymbol{e})$. First, we investigate $\boldsymbol{e}$ closely.

i	$\cdots$	$k\lambda-\lambda$	$k\lambda-\lambda+1$	$\cdots$	$k\lambda$	$k\lambda+1$	$\cdots$
$\boldsymbol{e}_i$	$\cdots$	$k-1$	k	$\cdots$	k	$k+1$	$\cdots$

Observe that symbol k appears at the $(k\lambda-\lambda+1)$th, $\ldots$, $(k\lambda)$th positions in $\boldsymbol{e}$. Therefore, π is d-close to $\boldsymbol{e}$ if and only if $\pi_{k\lambda-\lambda+1},\ldots,\pi_{k\lambda} \in [k-d,k+d]$. (Note that we simply ignore the non-positive values when $k \le d$.) In other words, $d_{\max}(\boldsymbol{e},\pi) \le d$ if and only if symbol k only appears in the $(k\lambda-d\lambda-\lambda+1)$th, $\ldots$, $(k\lambda+d\lambda)$th positions of π. This observation leads us to define the shift operator $\oplus$.

Definition 1. For an integer set S and an integer z, define $S \oplus z = \{s+z : s \in S\}$.

Fact 1. *Suppose that π is d-close to $\boldsymbol{e}$, then $\pi_i = k$ implies $i \in [-d\lambda+1, d\lambda+\lambda] \oplus (k\lambda-\lambda)$.*

The above fact is useful to capture the frequency permutations that are d-close to $\boldsymbol{e}$. Note that the set $S = [-d\lambda+1, d\lambda+\lambda]$ is independent of k. We give a recursive algorithm EnumV$_{\lambda,n,d}$ in Fig. 1. It enumerates all frequency permutations $\pi \in S_n^\lambda$ in $B(d,\boldsymbol{e})$. It is a depth-first-search style algorithm. Basically, it first tries to put 1's into λ proper vacant positions of π. Then, it tries to put 2's, $\ldots$, m's into the partial frequency permutations recursively. According to Fact 1, symbol k is assigned to positions of indices in $[-d\lambda+1, d\lambda+\lambda] \oplus (k\lambda-\lambda)$, and these positions are said to be *valid* for k.

The basic idea of our algorithms is that the set of valid positions being considered for symbol $k+1$ will be affected by the arrangement of symbols $1,\ldots,k$. Therefore, the number of frequency permutations in $B(d,\boldsymbol{e})$ is equal to the number of such arrangements of symbols. We model each set of valid positions as a vertex and each arrangement of a symbol as an edge in a graph, then we can compute the size of $B(d,\boldsymbol{e})$ by the number of paths between two specific vertices, which can be evaluated in polynomial time.

```
EnumV_{λ,n,d}(k, P)
1.  if k ≤ m then
2.      for each partition (X, X') of P with |X| = λ do
3.          if X' ∩ (−∞, −dλ + λ] = ∅ then
            // Make sure it is a proper partition
4.              for i ∈ X ⊕ (kλ − λ) do
5.                  π_i ← k; // Assign k to the i-th position
6.              Y ← (X' ⊕ (−λ)) ∪ [dλ + 1, dλ + λ];
7.              EnumV_{λ,n,d}(k + 1, Y);
8.              for i ∈ X ⊕ (kλ − λ) do
9.                  π_i ← 0; // Reset π_i to be vacant
10. else
11.     if P = [dλ + λ] then output (π_1, ..., π_n);
```

Fig. 1. EnumV$_{\lambda,n,d}(k,P)$.

ENUMV$_{\lambda,n,d}$ takes an integer k and a subset P of $[-d\lambda+1, d\lambda+\lambda]$ as its input, and ENUMV$_{\lambda,n,d}$ uses an $(n+2d\lambda)$-dimensional vector π as a global variable. For convenience, we extend the index set of π to $[-d\lambda+1, n+d\lambda]$ and every entry of π is initialized to 0, which indicates that the entry is vacant. We use P to trace the indices of valid vacant positions for symbol k, and the set of such positions is exactly $P \oplus (k\lambda - \lambda)$.

We call ENUMV$_{\lambda,n,d}(1, [d\lambda+\lambda])$ to enumerate all frequency permutations which are d-close to $\boldsymbol{e}$. During the enumeration, ENUMV$_{\lambda,n,d}(k, P)$ assigns symbol k into some λ positions, indexed by $X \oplus (k\lambda - \lambda)$, of a partial frequency permutation, then it recursively invokes ENUMV$_{\lambda,n,d}(k+1, (X' \oplus (-\lambda)) \cup [d\lambda+1, d\lambda+\lambda])$, where X and X' form a partition of P and $|X| = \lambda$. After the recursive call is done, ENUMV$_{\lambda,n,d}(k, P)$ resets the positions indexed by $X \oplus (k\lambda - \lambda)$ as vacant. Then, it repeats to search another choice of λ positions until all possible combinations of λ positions are investigated. For $k = m+1$, ENUMV$_{\lambda,n,d}(k, P)$ outputs π if $P = [d\lambda+\lambda]$. Given $n = \lambda m$, k is initialized to 1 and P to $[d\lambda+\lambda]$, we have the following claims.

Claim 1. *In each of the recursive calls of* ENUMV$_{\lambda,n,d}$, *in line 6 we have* $\max(Y) = d\lambda+\lambda$.

Proof. By induction, it is clear for $k = 1$. Suppose $\max(P) = d\lambda+\lambda$. Since $Y = (X' \oplus (-\lambda)) \cup [d\lambda+1, d\lambda+\lambda]$ and $\max(X') \le d\lambda+\lambda$, we have $\max(Y) = d\lambda+\lambda$. □

Claim 2. *In line 6, for each* $k \in [m+1]$ *and each* $i \in Y \oplus ((k+1)\lambda - \lambda)$, *we have* $\pi_i = 0$.

Proof. We prove this by induction on k. It is clear for $k = 1$. Assume the claim is true up to $k < m+1$, i.e., for each $i \in P \oplus (k\lambda - \lambda)$, $\pi_i = 0$. Now, consider the following scenario, ENUMV$_{\lambda,n,d}(k, P)$ invokes ENUMV$_{\lambda,n,d}(k+1, Y)$.

Since $Y = (X' \oplus (-\lambda)) \cup [d\lambda+1, d\lambda+\lambda]$, we have $Y \oplus ((k+1)\lambda - \lambda) = (X' \oplus (k\lambda - \lambda)) \cup [k\lambda+d\lambda+1, k\lambda+d\lambda+\lambda]$. While $X' \subset P$ and $[k\lambda+d\lambda+1, k\lambda+d\lambda+\lambda]$ are new vacant positions, it is clear $\pi_i = 0$ in these entries. □

Claim 3. *In each recursive call of* ENUMV$_{\lambda,n,d}(k, P)$, *P must be a subset of* $[-d\lambda+1, d\lambda+\lambda]$ *of cardinality* $d\lambda+\lambda$. *This implies,* $|e_i - k| \le d$ *for* $i \in P \oplus (k\lambda - \lambda)$.

Proof. We prove this by induction on k. For $k = 1$, P is $[d\lambda+\lambda]$, and the claim is obvious. Assume the claim is true up to k, and ENUMV$_{\lambda,n,d}(k, P)$ invokes ENUMV$_{\lambda,n,d}(k+1, Y)$. Thus $Y = (X' \oplus (-\lambda)) \cup [d\lambda+1, d\lambda+\lambda]$. Due to the constraint on X' in line 3 and the induction hypothesis, we have $X' \oplus (-\lambda) \subseteq [-d\lambda+1, d\lambda]$. We conclude that $Y \subseteq [-d\lambda+1, d\lambda] \cup [d\lambda+1, d\lambda+\lambda]$ and $|Y| = |X'| + \lambda = d\lambda+\lambda$. Since $[-d\lambda+1, d\lambda+\lambda] \oplus (k\lambda) = [k\lambda-d\lambda+1, k\lambda+d\lambda+\lambda]$, we know $\boldsymbol{e}$ has values from $[k-d+1, k+d+1]$ in these positions. I.e., $|e_i - (k+1)| \le d$ for $i \in Y \oplus (k\lambda)$. Hence, the claim is true. □

Claim 4. *At the beginning of the invocation of* ENUMV$_{\lambda,n,d}(k, P)$, $i \in P \oplus (k\lambda - \lambda)$ *implies* $i > 0$.

Proof. It is clear for $k = 1$. Observe that $\min(Y) \ge \min(P) - \lambda$. Since, $(\min(P) - \lambda) \oplus (k\lambda) = \min(P) \oplus (k\lambda - \lambda)$, the claim holds for $k > 1$. □

Claim 5. *For* $k \in [m]$, *when* ENUMV$_{\lambda,n,d}(k, P)$ *invokes* ENUMV$_{\lambda,n,d}(k+1, Y)$ *in line 7, there are exactly* λ *entries of* π *equal i for* $i \in [k-1]$.

Proof. It is implied by lines 4 and 5. □

Lemma 1. *At the beginning of the execution of* ENUMV$_{\lambda,n,d}(k, P)$, $P \oplus (k\lambda - \lambda) = \{i : i > 0 \wedge \pi_i = 0 \wedge i \in [-d\lambda+1, d\lambda+\lambda] \oplus (k\lambda - \lambda)\}$.

Proof. The lemma holds by Claims 2–4. □

Let ρ be one of the outputs of $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$. These facts ensure that $\rho \in S_n^\lambda$ and $d_{\max}(\boldsymbol{e}, \pi) \leq d$.

Lemma 2. *For $k \in [m+1]$, let τ_k be a partial frequency permutation d-close to $\boldsymbol{e}$ and with each symbol $1, \ldots, k-1$ appearing exactly λ times in τ_k. If $\text{ENUMV}_{\lambda,n,d}(k, P)$ is invoked with π such that*

$$\pi_i = \begin{cases} 0, & (\tau_k)_i = *, \\ (\tau_k)_i, & (\tau_k)_i \neq *, \end{cases}$$

then each frequency permutation ρ in the output satisfies the following conditions:

1. *ρ is consistent with τ_k over the entries with symbols $1, \ldots, k-1$.*
2. *ρ is d-close to $\boldsymbol{e}$.*

Proof. We prove this by reverse induction. First, we consider the case $k = m+1$. Note that $\text{ENUMV}_{\lambda,n,d}(m+1, P)$ outputs $(\pi_1, \ldots, \pi_n)$ only if $P = [d\lambda + \lambda]$, otherwise there is no output. Observe that $(\tau_{m+1})_i \neq *$ for $i \in [n]$. By the definition of π, we know that $\pi_i = (\tau_{m+1})_i$ for $i \in [n]$. Hence, $(\pi_1, \ldots, \pi_n)$ is a frequency permutation in S_n^λ, and the first condition holds. Since τ_{m+1} is d-close to $\boldsymbol{e}$, the claim is true for $k = m+1$.

Assume the claim is true down to $k+1$. For k, by Lemma 1, $P \oplus (k\lambda - \lambda)$ is exactly the set of all positions which are vacant and valid for k. Line 3 of $\text{ENUMV}_{\lambda,n,d}(k, P)$ ensures that X is properly selected. Then symbol k is assigned to $X \oplus (k\lambda - \lambda)$. There are two possible cases:

- For every $i \in X \oplus (k\lambda - \lambda)$, $i \leq n$. Define partial frequency permutation $\tau_{k+1} = ((\tau_{k+1})_1, \ldots, (\tau_{k+1})_n)$ by setting $(\tau_{k+1})_i = *$ for $\pi_i = 0$ and $(\tau_{k+1})_i = \pi_i$ for the others. By Claim 5, each of symbol $1, \ldots, k$ appears exactly λ times in τ_{k+1}. By the induction hypothesis, it is clear that the frequency permutations, generated by $\text{ENUMV}_{\lambda,n,d}(k+1, Y)$ on line 7, satisfy both conditions.
- There is some $i > n$ and $i \in X \oplus (k\lambda - \lambda)$, so we have $\pi_i = k \neq 0$. By Lemma 1, $\text{ENUMV}_{\lambda,n,d}(k, P)$ never recursively invokes $\text{ENUMV}_{\lambda,n,d}(m+1, [d\lambda + \lambda])$. Therefore, nothing will be output.

We conclude that $\text{ENUMV}_{\lambda,n,d}(k, P)$ outputs only frequency permutation satisfying these two conditions, and this lemma is true. □

We have the following theorem as an immediate result of Lemma 2.

Theorem 1. $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ *enumerates exactly all frequency permutations d-close to $\boldsymbol{e}$ in S_n^λ.*

Proof. Let ρ be a frequency permutation d-close to $\boldsymbol{e}$ in S_n^λ. Let $\tau_{m+1} = \rho$. For $k \in [m]$, define τ_k by

$$(\tau_k)_i = \begin{cases} (\tau_{k+1})_i, & \tau_{k+1} < k. \\ 0, & (\tau_{k+1})_i \geq k. \end{cases}$$

Note that τ_1 has 0 in all of its entries. There exists a sequence $P_1, \ldots, P_{m+1}$ such that $\text{ENUMV}_{\lambda,n,d}(k, P_k)$ with $\pi = \tau_{k+1}$ invokes $\text{ENUMV}_{\lambda,n,d}(k+1, P_{k+1})$ with $\pi = \tau_{k+1}$ for every $k \in [m]$. To see this fact, we simply set $P_{m+1} = [d\lambda + \lambda]$ and $P_k = ((P_{k+1} \backslash [d\lambda + 1, d\lambda + \lambda]) \oplus \lambda) \cup \{i - k\lambda + \lambda : \rho_i = k\}$. Therefore, ρ will be enumerated eventually.

Lemma 2 states that $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ only outputs frequency permutations d-close to $\boldsymbol{e}$ in S_n^λ, thus the theorem is true. □

4. Computing $V_{\lambda,n,d}$

The number of elements generated by $\textsc{EnumV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ is clearly $V_{\lambda,n,d}$. However, the enumeration is not efficient, since $V_{\lambda,n,d}$ is usually a very large number. In this section, we give two efficient implementations to compute $V_{\lambda,n,d}$. Especially, $V_{\lambda,n,d}$ can be computed in polynomial time for constant d and λ.

From the algorithm EnumV, we see that whether $\textsc{EnumV}_{\lambda,n,d}(k, P)$ invokes $\textsc{EnumV}_{\lambda,n,d}(k+1, Y)$ or not depends only on k, P and Y. During the execution of $\textsc{EnumV}_{\lambda,n,d}(1, [d\lambda+\lambda])$, $\textsc{EnumV}_{\lambda,n,d}(k, P)$ is invoked recursively only when $[d\lambda+1, d\lambda+\lambda] \subset P \subset [-d\lambda+1, d\lambda+\lambda]$, due to line 6. Therefore, we can construct a directed acyclic graph $G_{\lambda,n,d} = \langle V_G, E_G\rangle$ where

- $V_G = \{(k, U) : k \in [m+1], U \subset [-d\lambda+1, d\lambda] \text{ and } |U| = d\lambda\}$.
- $((k, U), (k+1, V)) \in E_G$ if and only if $\textsc{EnumV}_{\lambda,n,d}(k, U \cup [d\lambda+1, d\lambda+\lambda])$ invokes $\textsc{EnumV}_{\lambda,n,d}(k+1, V \cup [d\lambda+1, d\lambda+\lambda])$.

For example, Fig. 2 shows the structure of $G_{2,6,1}$.

$V_{\lambda,n,d}$ equals the number of invocations of $\textsc{EnumV}_{\lambda,n,d}(m+1, [d\lambda+\lambda])$. With this observation, $V_{\lambda,n,d}$ also equals the number of paths from $(1, [d\lambda])$ to $(m+1, [d\lambda])$ in $G_{\lambda,n,d}$. By the definition of $G_{\lambda,n,d}$, it is a directed acyclic graph. The number of paths from one vertex to another in a directed acyclic graph can be computed in $O(|V|+|E|)$, where $|V| = (m+1)\binom{2d\lambda}{d\lambda}$ and $|E| = O(|V|^2)$. So $V_{\lambda,n,d}$ can be calculated in polynomial time with respect to n if λ and d are constants.

The computation actually can be done in $O(\log n)$ time for constant λ and d. Define $H_{\lambda,d} = \langle V_H, E_H\rangle$ where $V_H = \{P : |P| = d\lambda \text{ and } P \subseteq [-d\lambda+1, d\lambda]\}$ and $(P, P') \in E_H$ if and only if there is some

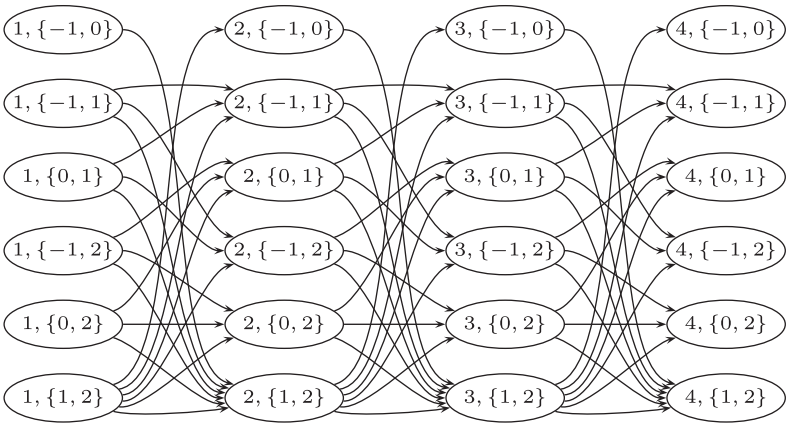


Fig. 2. Graph $G_{2,6,1}$.

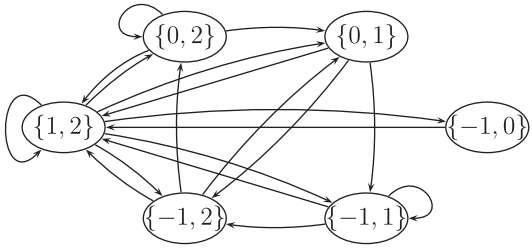


Fig. 3. Graph $H_{2,1}$.

$k \in [m]$ such that $\textsc{EnumV}_{\lambda,n,d}(k, P \cup [d\lambda+1, d\lambda+\lambda])$ invokes $\textsc{EnumV}_{\lambda,n,d}(k+1, P' \cup [d\lambda+1, d\lambda+\lambda])$. Note that $|V_H| = \binom{2d\lambda}{d\lambda}$. Fig. 3 shows $H_{2,1}$ as an example.

Theorem 2. *$V_{\lambda,n,d}$ can be found in $O\left(\binom{2d\lambda}{d\lambda}^{2.376} \log m\right)$ time and $O\left(\binom{2d\lambda}{d\lambda}^2\right)$ space.*

Proof. Observe that the value of $k \in [m]$ is independent of the invocation of $\textsc{EnumV}_{\lambda,n,d}(k+1, P' \cup [d\lambda+1, d\lambda+\lambda])$ by $\textsc{EnumV}_{\lambda,n,d}(k, P \cup [d\lambda+1, d\lambda+\lambda])$, where P and $P' \subset [-d\lambda+1, d\lambda]$ with $|P| = |P'| = d\lambda$. Therefore, the number of paths of length m from $[d\lambda]$ to itself in $H_{\lambda,d}$ is equal to the number of paths from $(1, [d\lambda])$ to $(m+1, [d\lambda])$ in $G_{\lambda,n,d}$.

Let $V_H = \{v_1, \ldots, v_{|V_H|}\}$, where $v_1 = [1, d\lambda]$. The number of paths of length m from v_1 to v_1 is the first entry of the first column of the mth power of A_H, where A_H is the adjacency matrix of $H_{\lambda,d}$. Since mth power can be computed in $O\left(f\left(\binom{2d\lambda}{d\lambda}\right) \log m\right)$, where $O(f(x))$ is the time cost of multiplying two $x \times x$ matrices. It is well-know that $f(x) = O(x^{2.376})$ by the Coppersmith–Winograd algorithm. This algorithm needs a space of $O\left(\binom{2d\lambda}{d\lambda}^2\right)$ entries for storing A_H and the intermediate results. □

With constants λ and d, we actually show that $V_{\lambda,n,d}$ can be computed in $O(\log n)$ time. However, the space to store the adjacency matrix A_H can be too large to execute the above algorithm in practice, even for small d and λ. Note that at least $2 \times \binom{2d\lambda}{d\lambda}^2$ entries and $\binom{2d\lambda}{d\lambda}^{2.375}$ multiplications are required when the above implementation uses the Coppersmith-Winograd algorithm to multiply matrices. Now we turn to an example showing that the above implementation can be practically inefficient. By setting $d = 3$, $\lambda = 3$ and $m = 100$, we need at least $2 \times \binom{18}{9}^2 \approx 4.73 \times 10^9$ entries to store A_H and the intermediate results of A_H^m. In this case, a space for 4.73×10^9 integers and at least $\binom{18}{9}^{2.375} \times \log_2 100 \approx 8.99 \times 10^{11}$ multiplications are required to compute A_H^m in this way, so it is too hard to carry out with an ordinary PC. Hence, we provide an alternative implementation which runs in $O\left(\binom{2d\lambda}{d\lambda} \cdot \binom{d\lambda+\lambda}{\lambda} \cdot m\right)$ time and $O\left(\binom{2d\lambda}{d\lambda}\right)$ space. This allows us to compute more efficiently for the cases with smaller m and larger d and λ. To achieve the $O\left(\binom{2d\lambda}{d\lambda}\right)$-space complexity, we do not store the adjacency matrix A_H in the memory. Since A_H is the adjacency matrix of $H_{\lambda,d}$, for $\boldsymbol{y} = (y_1, \ldots, y_{|V_H|})$ and $\boldsymbol{y}' = A_H\boldsymbol{y} = (y'_1, \ldots, y'_{|V_H|})$, we have $y'_i = \sum_{(i,j)\in E_H} y_j$. Hence, if enumerating all edges in E_H takes S space and T time, then we can compute $A_H\boldsymbol{y}$ in $O(|V_H| + S)$ space and $O(T)$ time for any $|V_H|$-dimensional $\boldsymbol{y}$.

Lemma 3. *$|E_H| \le |V_H| \cdot \binom{d\lambda+\lambda}{\lambda}$ and E_H can be enumerated in $O(d\lambda)$ space and $O(|E_H|)$ time.*

Proof. For $P \in V_H$ such that $|P \cap (-\infty, -d\lambda - \lambda]| = r$, P has $\binom{d\lambda+\lambda-r}{\lambda-r}$ out-going edges, since every partition (X, X') of $P \cup [d\lambda+1, d\lambda+\lambda]$ satisfies the condition in line 3 if and only if $P \cap (-\infty, -d\lambda - \lambda] \subset X$, i.e., every choice of $(\lambda - r)$-element subset of $P \setminus (-\infty, -d\lambda - \lambda]$ will invoke a recursive call. Since $\binom{d\lambda+\lambda-r}{\lambda-r} \le \binom{d\lambda+\lambda}{\lambda}$, the number of edges has an upper bound $|V_H|\binom{d\lambda+\lambda}{\lambda}$.

To enumerate all λ-element subsets of a $(d\lambda+\lambda)$-element set, we need $O(d\lambda+\lambda) = O(d\lambda)$ space and $O(\binom{d\lambda+\lambda}{\lambda})$ time. Since we can recycle the space, the enumeration of edges in E_H can be done in $O(d\lambda)$ space and $O(|E_H|)$ time. □

Now, we give the alternative implementation.

Theorem 3. *$V_{\lambda,n,d}$ can be computed in $O\left(\binom{2d\lambda}{d\lambda}\binom{d\lambda+\lambda}{\lambda} m\right)$ time and $O(\binom{2d\lambda}{d\lambda})$ space.*

Proof. Let $\boldsymbol{x} = (1, 0, \ldots, 0)^T$. Since $A_H^m\boldsymbol{x}$ is the first column of A_H^m, $V_{\lambda,n,d}$ is the first entry of $A_H^m\boldsymbol{x}$. The alternative evaluates $A_H^1\boldsymbol{x}, \ldots, A_H^m\boldsymbol{x}$ iteratively. Instead of storing the whole adjacency matrix, it only uses two $|V_H|$-dimensional vectors $\boldsymbol{y}$ and $\boldsymbol{y}'$ for storing $A_H^i\boldsymbol{x}$ and the intermediate result of $A_H^{i+1}\boldsymbol{x}$, respectively. Initially, $\boldsymbol{y} = \boldsymbol{x}$ and $i = 0$. We compute $A_H\boldsymbol{y}$ by the algorithm described in Lemma 3 and using $\boldsymbol{y}'$ to store the intermediate result. Then, we copy the result of $A_H\boldsymbol{y}$ back to $\boldsymbol{y}$. After m repetitions, we have $\boldsymbol{y} = A_H^m\boldsymbol{x}$, and the first entry of $\boldsymbol{y}$ is $V_{\lambda,n,d}$. Therefore, the space complexity can be reduced to $O(|V_H| + d\lambda) = O(\binom{2d\lambda}{d\lambda})$. The running time is $m \cdot O(|E_H|) = O\left(\binom{2d\lambda}{d\lambda} \cdot \binom{d\lambda+\lambda}{\lambda} \cdot m\right)$. □

Here, we briefly compare this implementation with the previous one. In this implementation, the required space is dominated by $3 \times \binom{2d\lambda}{d\lambda}$ integral entries, since there are only two $\binom{2d\lambda}{d\lambda}$-dimensional vectors, and the others need less than $\binom{2d\lambda}{d\lambda}$ integers. Computing $\sum_{(i,j)\in E_H} y_j$ for every $i \in V_H$ consumes almost all execution time of the multiplication $A_H\boldsymbol{y}$, and the multiplication operations dominate the others in time consumption. Therefore, the execution time should be no more than $4 \times \binom{2d\lambda}{d\lambda}\binom{d\lambda+\lambda}{\lambda}m$ additions. For $d = 3$, $\lambda = 3$, and $m = 100$, this implementation needs a space for storing at most $3 \times \binom{18}{9} \approx 1.46 \times 10^5$ integers and an execution time for at most $4 \times \binom{18}{9}\binom{12}{3} \times 100 \approx 4.28 \times 10^9$ additions to compute $V_{3,300,3}$. It is much more efficient than the previous one under such configuration.

5. Comparison with previous results

In this section, we compare our results with previous ones. Shieh and Tsai [13] showed that $(\lambda!)^m V_{\lambda,n,d}$ equals the permanent of 0-1 matrix $A^{(\lambda,n,d)} = (a_{i,j}^{(\lambda,n,d)})$ where $a_{i,j}^{(\lambda,n,d)} = 1$ if and only if $\left|\left\lceil \frac{i}{\lambda} \right\rceil - \left\lceil \frac{j}{\lambda} \right\rceil\right| \le d$. For example,

$$A^{(2,8,1)} = \begin{pmatrix} 1&1&1&1&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&1&1&0&0 \\ 1&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&1 \\ 0&0&1&1&1&1&1&1 \\ 0&0&0&0&1&1&1&1 \\ 0&0&0&0&1&1&1&1 \end{pmatrix}.$$

This gives some simple case of computing $V_{\lambda,n,d}$.

Fact 2. *For $m \le d + 1$ and $n = m\lambda$, $V_{\lambda,n,d} = \frac{n!}{(\lambda!)^m}$.*

Proof. In such case, every entry of $A^{\lambda,n,d}$ equals 1. Therefore, the permanent of $A^{\lambda,n,d}$ is $n!$, and $V_{\lambda,n,d} = \frac{n!}{(\lambda!)^m}$. □

However, it is not always easy to compute the permanent of an arbitrary matrix. A naive approach to evaluate the permanent of an n-by-n matrix takes $O(n!)$ time. In practice, $\Theta((\lambda!)^m V_{\lambda,n,d})$ time is still required when using a backtracking algorithm. It is clear that both of our methods are much faster.

Kløve [5] solved the recurrence of $V_{\lambda,n,1}$, and he gave the value of $V_{\lambda,\lambda m,1}$ for $\lambda \in [10]$ and $m \in [20]$. Schwartz [10] gave an algorithm which can be applied for computing $V_{1,n,d}$. In this paper, we provide solutions to computing $V_{\lambda,n,d}$ for $\lambda > 1$ and $d > 1$, which is not known in their works. We list the values of $V_{\lambda,\lambda m,d}$ for $\lambda > 1$, $m \in [20]$, $d > 1$, and $d\lambda \leq 10$ in http://www.csie.nctu.edu.tw/^mzhsieh/balltable.pdf.

6. Conclusion

We extend Schwartz's result [10] to solve the ball size of frequency permutations under Chebyshev distance. We give efficient algorithms for the cases of larger frequency, larger minimum distance and smaller symbol set.

References

[1] J.C. Chang, R.J. Chen, T. Kløve, S.C. Tsai, Distance-preserving mappings from binary vectors to permutations, IEEE Trans. Inform. Theory 49 (2003) 1054–1059.
[2] Sophie Huczynska, Gary L. Mullen, Frequency permutation arrays, J. Combin. Des. 14 (2006) 463–478.
[3] A. Jiang, R. Mateescu, M. Schwartz, J. Bruck, Rank modulation for flash memories, in: Proc. of the 2008 IEEE Internat. Symp. on Inform. Theory (ISIT2008), Toronto, Canada, 2008, pp. 1731–1735.
[4] A. Jiang, M. Schwartz, J. Bruck, Error-correcting codes for rank modulation, in: Proc. of the 2008 IEEE Internat. Symp. on Inform. Theory (ISIT2008), Toronto, Canada, 2008, pp. 1736–1740.
[5] T. Kløve, Frequency permutation arrays within distance one, Reports in Informatics, Dept. of Informatics, Univ. Bergen, Report no. 382, 2009.
[6] T. Kløve, Lower bounds on the size of spheres of permutations under the Chebychev distance, Des. Codes Cryptogr. 58 (2011) 183–191.
[7] T. Kløve, T.-T. Lin, S.-C. Tsai, W.-G. Tzeng, Permutation arrays under the Chebyshev distance, IEEE Trans. Inform. Theory 56 (2010) 2611–2617.
[8] T.T. Lin, S.C. Tsai, W.G. Tzeng, Efficient encoding and decoding with permutation arrays, in: Proc. of the 2008 IEEE Internat. Symp. on Inform. Theory (ISIT2008), Toronto, Canada, 2008, pp. 211–214.
[9] K.W. Shum, Permutation coding and MFSK modulation for frequency selective channel, IEEE Personal, Indoor and Mobile Radio Communications 13 (2002) 2063–2066.
[10] M. Schwartz, Efficiently computing the permanent and Hafnian of some banded Toeplitz matrices, Linear Algebra Appl. 430 (2009) 1364–1374.
[11] D. Slepian, Permutation modulation, Proc. IEEE 53 (1965) 228–236.
[12] I. Tamo, M. Schwartz, Correcting limited-magnitude errors in the rank-modulation scheme, IEEE Trans. Inform. Theory 56 (2010) 2551–2560.
[13] M.-Z. Shieh, S.-C. Tsai, Decoding frequency permutation arrays under Chebyshev distance, IEEE Trans. Inform. Theory 56 (2010) 5730–5737.
[14] A.J.H. Vinck, J. Häring, Coding and modulation for power-line communications, in: Proc. of the 2000 Internat. Symp. on Power Line Commun, Limerick, Ireland, 2000, pp. 265–272.
[15] A.J.H. Vinck, J. Häring, T. Wadayama, Coded M-FSK for power line communications, in: Proc. of the 2000 IEEE Internat. Symp. on Inform. Theory (ISIT2000), Sorrento, Italy, 2000, p. 137.
[16] A.J.H. Vinck, Coded modulation for power line communications, AEÜ Int. J. Electron. Commun. 54 (2000) 45–49.