

國立交通大學

資訊科學與工程研究所

博 士 論 文



研 究 生：謝旻錚

指導教授：蔡錫鈞 教授

中 華 民 國 一 百 年 六 月



頻率排列碼

On Frequency Permutation Arrays

研 究 生：謝旻錚

Student : Min-Zheng Shieh

指導教授：蔡錫鈞

Advisor : Shi-Chun Tsai



Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering
June 2010
Hsinchu, Taiwan, Republic of China

中華民國一百年六月



令 n 為正整數 m 與 λ 的乘積。一個長度為 n 、最小距離為 d 的頻率排列碼(frequency permutation array, 簡稱FPA)是一個包含若干個由 m 個符號各重複出現 λ 次所形成的排列, 並且其中相異的兩個排列的距離至少為 d 。FPA是排列碼(permutation array, 簡稱PA)的一般化, PA即是FPA取 $\lambda = 1$ 時的特例。目前PA已有許多領域的應用, 諸如電力線通訊、快閃記憶體資料儲存以及密碼學。FPA具有比PA更大的設計彈性, 因此在各方面應用上均有較PA更高的潛力。

在本篇論文中, 吾人透過估計特定半徑內的頻率排列個數, 證明了在柴比雪夫距離(Chebyshev distance)下, FPA元素數量之Sphere-packing類型上界以及Gilbert-Varshamov類型下界。此外, 吾人亦提出兩個有效率的演算法, 用以正確計算出特定半徑內的頻率排列個數。其中之一能在 $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}^{2.376} \log n\right)$ 的時間複雜度以及 $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}^2\right)$ 的空間複雜度完成計算。另外一個則僅需 $\mathcal{O}\left(\binom{2d\lambda}{d\lambda} \binom{d\lambda+\lambda}{\lambda} \frac{n}{\lambda}\right)$ 的時間與 $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$ 的空間資源便能完成。當 λ 與 d 均為常數時, 這兩個演算法均能有效率的求出特定半徑內的頻率排列個數。

吾人發明數個建造FPA的方法, 並藉由此推導出對應的FPA元素數量下界。這些建造的方法中, 有些類型的FPA具有高效率的編碼與解碼方式, 而其中一個更具有區域可解碼演算法以及列表可解碼演算法。吾人展示了如何在僅讀取 $\lambda + 1$ 個符號的情況下, 解出一個資訊符號, 以及針對一個任意給定的排列, 如何找出所有特定距離內FPA中的元素。此外, 藉由區域可解碼演算法, 吾人亦造出一個私密資料取回的安全協定。

吾人透過研究子群碼(subgroup code)的性質, 證明了在一般情況下, 計算出任一FPA的最小距離是困難的。子群碼是PA的一個特例, 其中任兩個元素, 在函數合成的運算下, 具有封閉性。吾人證明, 在柴比雪夫距離下, 計算子群碼的最小距離, 等價於計算其中非(non-identity permutation)之最小權重。更進一步的, 吾人證明了後者係為一NP-complete問題, 以及對任一常數 ϵ , 逼近子群碼之最小距離至 $2 - \epsilon$ 的倍率, 亦為NP-hard。



ABSTRACT

Let m and λ be positive integers. A frequency permutation array (FPA) of length $n = m\lambda$ and distance d is a set of permutations on a multiset over m symbols, where each symbol appears exactly λ times and the distance between any two elements in the array is at least d . FPA generalizes the notion of permutation array (PA), which is a special case of FPA by choosing $\lambda = 1$. PAs are known for various applications in power line communication, flash memories and cryptography. FPA's are more flexible than PAs, since the length and the symbol set size of a PA must be equal. FPA's are potentially better than PAs in various of applications. For example, FPA's have higher information rate than PAs do when their symbol sets are the same.

In this thesis, under Chebyshev distance, we prove a Gilbert-Varshamov type lower bound and a sphere-packing type upper bound on the cardinality of FPA via bounding the size of balls of certain radii. Moreover, we propose two efficient algorithms that compute the ball size under Chebyshev distance. The first one runs in $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}^{2.376} \log n\right)$ time and $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}^2\right)$ space. The second one runs in $\mathcal{O}\left(\binom{2d\lambda}{d\lambda} \binom{d\lambda+\lambda}{\lambda} \frac{n}{\lambda}\right)$ time and $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$ space. For small constants λ and d , both are efficient in time and use constant storage space.

We give several constructions of FPA's, and we also derive lower bounds from these constructions. Some types of our FPA's come with efficient encoding and decoding capabilities. In addition, we show one of our designs is locally decodable and list decodable. In other words, we illustrate how to decode a message bit by reading at most $\lambda + 1$ symbols, and how to find the list of all codewords within a certain distance. Furthermore, we propose an FPA-based private information retrieval scheme, which follows from the locally decodable property.

On the other hand, we show that it is hard in general to determine the minimum distance of an arbitrary FPA by investigating subgroup codes. Subgroup permutation codes are permutation arrays exhibiting group algebra structures with the composition operator. Under Chebyshev distance, we prove that to determine the minimum distance of a subgroup code is equivalent to finding the minimum weight of non-identity codewords in a subgroup permutation code. Moreover, we prove the latter is NP-complete and it is NP-hard to approximate the minimum distance of a subgroup code within the factor $2 - \epsilon$ for any constant $\epsilon > 0$.



Acknowledgments

I would like to thank my parents for giving me a chance to accomplish this thesis. I am greatly indebted to Dr. Shi-Chun Tsai, my advisor, for guiding and encouraging me. I also wish to thank Dr. Wen-Guey Tzeng, Dr. Hsin-Lung Wu, Dr. Chia-Jung Lee, Mr. Ming Yu-Hsieh, Mr. Ying-Jie Liao, Mr. Te-Tsung Lin, Mr. Chang-Chun Lu, Mr. Jynn-Jy Lin, Mr. Min-Chuan Yang, Mr. Li-Rui Chen and all the other members in the CCIS research group for sharing their wisdom with me.

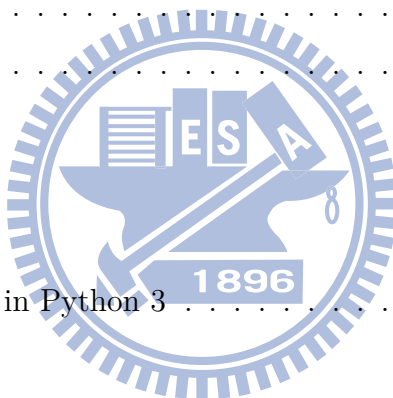




Contents

1	Introduction	1
1.1	Permutation Arrays and Their Applications	1
1.2	Frequency Permutation Arrays	4
1.3	Our Results	5
1.4	Organization of the Thesis	6
2	Preliminaries	7
2.1	Notations	7
2.2	Coding Theory	11
2.3	Computational Complexity	13
3	Explicit Lower and Upper Bounds	17
3.1	Gilbert Type and Sphere Packing Bounds	17
3.2	Enumerate Permutations in a Ball	22
3.3	Compute the Ball Size	27
4	Constructions and Related Bounds	33
4.1	An Explicit Construction	33
4.2	Recursive Constructions	34
5	Codes with Efficient Encoding and Decoding	41
5.1	Encoding Algorithm	41
5.2	Unique Decoding Algorithm	43
5.3	Local Decoding Algorithm	44

5.4	List Decoding Algorithm	46
5.5	Private Information Retrieval	48
6	Complexity Issues	51
6.1	Complexity Problems Related to FPAs	51
6.2	Minimum Distance of Subgroup Codes	53
6.3	Cameron-Wu's Reduction	68
7	Conclusion and Future Works	71
7.1	Conclusion	71
7.2	Future Works	71
7.3	Code-Anticode Type Bounds	73
7.4	Steganography	73
7.5	Covering Radius	74
A	Tables of Ball Size	83
B	Program Codes	93
B.1	Computing the Ball Size in Python 3	93



List of Tables

4.1	Compare C_1 and the bounds in Section ??	34
6.1	Weight contribution of different gadgets	57
6.2	Operation of Klein four-group.	58
6.3	Operation of basic construction blocks and their products.	60
6.4	Weights of the basic construction blocks	60
6.5	Weights of modified basic blocks	61
A.1	The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 2$	84
A.2	The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 3$	85
A.3	The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 4$	86
A.4	The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 5$	87
A.5	The table of ball size under ℓ_∞ -metric for $\lambda = 3, m \in [20], d = 2$	88
A.6	The table of ball size under ℓ_∞ -metric for $\lambda = 3, m \in [20], d = 3$	89
A.7	The table of ball size under ℓ_∞ -metric for $\lambda = 4, m \in [20], d = 2$	90
A.8	The table of ball size under ℓ_∞ -metric for $\lambda = 5, m \in [20], d = 2$	91



List of Figures

3.1	ENUMV $_{\lambda,n,d}(k, P)$	23
3.2	Graph $G_{2,6,1}$	28
3.3	Graph $H_{2,1}$	29
4.1	The extension algorithm $\phi_k(\mathbf{y}, \mathbf{x})$	35
4.2	Extension-based construction for code in Chapter ??	39
5.1	ENC $_{n,k}^\lambda$ encodes messages in Z_2^k with S_n^λ	42
5.2	UNIQUE $_{n,k}^\lambda$ decodes words in S_n^λ to messages in Z_2^k	43
5.3	LOCAL $_{n,k}^\lambda$ decodes one bit by reading at most $\lambda + 1$ symbols.	45
5.4	LIST $_{n,k}^\lambda$ gives a list of candidates of the closest codewords.	47
6.1	The sketch of reduction	54
6.2	Satisfiable ϕ is mapped into $(N, \alpha + 5)$ -PA $f(\phi)$	55
6.3	Unsatisfiable ϕ' is mapped into $(N, 2\alpha + 5)$ -PA $f(\phi')$	56
6.4	No position is permuted by 2 kinds of gadgets.	56
6.5	$\kappa'_1 \chi = \chi \kappa'_1$	59
6.6	Second layer blocks	62



Chapter 1

Introduction

Let S_n^λ be the set of all permutations on the multiset $\{\overbrace{1, \dots, 1}^\lambda, \dots, \overbrace{m, \dots, m}^\lambda\}$. A *frequency permutation array* (FPA) is a subset of S_n^λ for some positive integers m , λ and $n = m\lambda$. In this thesis, we investigate their properties and applications. In addition, we study how to construct FPAs with efficient encoding and decoding algorithms. On the other hand, we analyze the complexity of computational problems related to FPAs.

1.1 Permutation Arrays and Their Applications

A *permutation array* (PA) is simply a special case of an FPA by choosing $\lambda = 1$. PAs have been studied for a long time. Based on Slepian's [36] idea, the first error correcting PA is proposed by Blake [5] in 1974. Today, PAs have applications in various fields. Recently, researchers have found that PAs have applications in areas such as power line communication (e.g. [31], [41], [42], and [43]), block cipher (see [13]) and multi-level flash memories (see [20], [21] and [37]).

The communication devices will not work without electricity, so most of them are supposed to have a power line connected. As a consequence, power lines are considered as a highly potential solution for the last-mile connection. Vinck and Häring [41] proposed a transmission scheme using 4-Frequency-Shift-Keying (4-FSK) modulation coded by permutation arrays over the power lines. When transmitting data with m -FSK modulation, we send symbol $i \in \{1, \dots, m\}$ by a carrier wave of some unique frequency f_i . For example, we

send $(1, 3, 2, 4)$ by setting the frequency of the carrier wave to f_1, f_3, f_2 and f_4 in time slot 1, 2, 3 and 4, respectively. After the carrier wave passes through a noisy channel, the receiver may get extra symbols in some time slots. If there is a noise wave of frequency f_4 happening in time slot 1, then the receiver would get $\{f_1, f_4\}$ in time slot 1. There are two major noises in the channel over power lines,

1. Narrow band permanent noise. This kind of noise is generated by certain electronic devices such as television and computers. It often appears in every time slot. Thus, we always receive symbol i when a device generates a narrow band permanent noise of frequency f_i . The receiver may get $\{f_1, f_2\}, \{f_2, f_3\}, \{f_2\}$ and $\{f_2, f_4\}$ in time slot 1, 2, 3 and 4, respectively, if there is a permanent noise of frequency f_2 .
2. Broad band impulse noise. An impulse noise can be caused by unplugging a heavy load device or a power surge. It makes us to receive all symbols in a time slot. The receiver may get $\{f_1\}, \{f_3\}, \{f_2\}$ and $\{f_1, f_2, f_3, f_4\}$ in time slot 1, 2, 3 and 4, respectively, if there is an impulse noise of frequency in time slot 4.

The structure of permutation is robust against these noises, since every symbol appears exactly once in a permutation. For the narrow band permanent noise, we can find out the extra symbol, because the extra symbols are identical in every time slot. A permutation on $\{1, \dots, n\}$ can be uniquely determined if we know the symbols in $n - 1$ entries of it. Consequently, we can indicate the correct symbol in the time slot where the impulse noise happens. With error correcting capability, PAs can overcome others' minor noises to ensure the communication quality over power lines.

Flash memory has become one of the most important types of non-volatile data storage recently. The basic unit of a flash memory is a cell, which behaves like a battery. A multi-level flash memory cell can store m kinds of symbols where $m > 2$. The stored symbol is quantized by the discrete levels v_0, v_1, \dots, v_m of the cell voltage. For example, a cell represents symbol i if its voltage ranges from v_{i-1} to v_i . One of the noticeable properties of flash memory is the asymmetry of charging and discharging cells. It is possible to put electrons only into one cell in a short time, but removing electrons from a cell is not allowed. We can only relatively slowly remove all electrons in a block, which consists of about 10^5 cells (see [8]). This causes

some problems on data writing.

1. Even when we want to only write one cell, we need to erase a block, i.e., to remove all charges in it, then perform charging on each cell in it. This makes writing data into a cell slow. Moreover, the life time of a flash cell is heavily related to the number of discharges performed.
2. The overshooting problem. It is hard to accurately control the amount of electrons charged into a cell. There are often two charging strategies. The first one is to charge a large amount to the cell. If lucky, then we can write the cell correctly in one charge. Otherwise, we have to discharge the whole block slowly, and we need to charge every cell in the block. The second one is to charge a small amount to the cell, then check whether the voltage is high enough to represent the symbol. We might need to repeat for several times to accomplish the writing.

In order to accelerate the writing, Jiang *et al.* [20] illustrate the rank modulation for multi-level flash memory. Instead of symbol quantization with voltage levels, their concept is to represent symbols in accordance with the ranking of relative voltage levels. Therefore, we can change the symbol usually by several charging operations, and the frequency of discharging the block decreases significantly. Moreover, when the overshooting problem occurs, it can be fixed by several charging operation in general. Since the expected number of discharging decreases, the rank modulation scheme increases both the writing performance and the endurance of the multi-level flash memory. Furthermore, Jiang *et al.* [21] proposed some error correcting codes, which are exactly PAs, for storing data in multi-level flash memory. The overshooting problem can be solved directly by the error correcting scheme. The error correcting capability of PAs also provides immunity from the other problems.

In the past decade, many constructions have been proposed. These methods include applying distance preserving mapping (DPM) (see [9]), and distance increasing mapping (DIM) to linear block codes (e.g. [10] and [11]), exploiting the connection between PAs and mutually orthogonal latin squares (MOLS) [12], and the others (e.g. [26] and [37]). Conversely, researchers also studied the maximum cardinality of PAs under various settings. For instances, Deza and Vanstone [14] gave bounds on the maximum cardinalities of equidistant PAs un-

der Hamming distance, and recently, Kløve *et al.* [26] proposed bounds under Chebyshev distance.

There are plenty of choices for constructions of PAs, however, only few families of PAs known to have both efficient (in polynomial time of the codeword length) encoding and decoding algorithms, such as the codes proposed by Lin *et al.* [28] and by Swart and Ferreira [35]. Some constructions, such as Babaev's [3], consider encoding as computing the binary representation for some permutation, and decoding as computing the permutation represented by some binary string. The schemes in [3] are efficient, but they do not exhibit any error correcting capability. Swart and Ferreira [35] gave a decoding algorithm for a particular PA applied to power line communication. Lin *et al.* [28] proposed a couple of novel constructions with efficient encoding and decoding algorithms for PAs under Chebyshev distance.

1.2 Frequency Permutation Arrays

FPA was proposed by Huczynska and Mullen [17] as a generalization of PA. They gave several constructions of FPA under Hamming distance and bounds on the maximum array size. Some of their constructions extend the concept of MOLS. They also gave bounds on the maximum cardinality of FPAs under Hamming distance.

Similar to the application of PAs for power line communication, we can encode a message as a frequency permutation from S_n^λ . Then the message is transmitted as m -FSK signals. The nature of frequency permutations provides higher information rate without losing immunity to narrow band permanent frequency disturbances and broad band impulse noise mentioned in Vinck's work [43], since the numbers of different symbols are equal.

For flash memory applications, different from the approach by Jiang *et al.* [21], we can use FPA to provide multi-level flash memory with error correcting capabilities. For example, suppose a multi-level flash memory, where each cell has m states, which can be changed by injecting or removing charge into or from it. Over injecting or charge leakage will alter the state as well. We can use the charge ranks of n cells to represent a frequency permutation from S_n^λ , i.e., the cells with the lowest λ charge levels represent symbol 1, and so on.

In general, a longer code can resist stronger burst errors in communication and tolerate

more severe failure in data storage. However, the lengths of PAs are limited by the size of their symbol set, which can be the number of distinct frequencies of power line or the number of charge levels of flash memory. For FPAs, there is no such limitation. We can easily construct much longer FPAs, especially, the codeword length of an FPA can be as long as the block length of a multi-flash memory. Hence, FPAs are more suitable for these applications.

1.3 Our Results

First of all, we prove a Gilbert-Varshamov type lower bound and a sphere-packing type upper for the maximum cardinality of FPAs under Chebyshev distance. We obtain the close form by estimating the size of ball by bounding the permanent of a special family of matrices. We also give several efficient dynamic programming style algorithms to compute the exact ball size by investigating an enumeration algorithm closely.

Then, we give several methods to construct FPAs under Chebyshev distance. These methods include a direct construction and various recursive constructions. They mainly extend the ideas from the work of Kløve *et al.* [26]. We give efficient encoding and decoding algorithms for a family of FPAs. The decoding algorithms include a unique decoding algorithm, a local decoding algorithm and a list decoding algorithm. Therefore, FPAs exhibit strong capabilities in error correcting.

A locally decodable code has an extremely efficient decoding for any message bit by reading at most a fixed number of symbols from the received word. Suppose that an FPA is applied to a multi-level flash memory where the length of a codeword equals the block length. This feature allows us to retrieve the desired message bits from a multi-level flash without accessing the whole block. With the locally decodable property, we can raise the robustness of the code without loss of efficiency. Also, we show our construction of FPA can be used in cryptographic application. Locally decodable codes have been under study for years, see [38] for a survey and [45], [16] for recent progress. They are related to a cryptographic protocol called *private information retrieval* (PIR for short).

There are many complexity issues on the designs of codes. “Is determining the minimum

distance of a code hard?” and “How fast can we compute the closest codeword for a certain received string?” may be the top two problems frequently asked. For $\lambda = 1$, both problems are proved to be NP-complete under many metrics, such as Hamming distance, Chebyshev distance (ℓ_∞ -metric), Kendall’s tau, *et al.*[7, 6]. However, for Chebyshev distance, the NP-completeness proof by Cameron and Wu [7] fell apart on some instances. We give a correct proof for the NP-completeness. Moreover, we also show the problem is NP-hard to approximate within $2 - \epsilon$ for every constant $\epsilon > 0$.

1.4 Organization of the Thesis

In Chapter 2, we will give notations used throughout this thesis and some background knowledge of permutations, computational complexity and coding theory. In Chapter 3, we present how to obtain explicit bounds on the maximum cardinality of FPAs. In Chapter 4, we give both explicit and recursive methods to construct FPAs of certain parameters. In Chapter 5, we show an efficient encoding and three decoding algorithms for a family of FPAs constructed in a simple manner. In addition, we construct a protocol for private information retrieval based on the locally decodable property. In Chapter 6, we discuss issues in the aspect of computational complexity. Finally, we conclude this thesis with some future works in Chapter 7.

Chapter 2

Preliminaries

2.1 Notations

Here, we give useful notations. Let (a_1, a_2, \dots, a_n) denote an n -tuple. We use $[n]$ to represent the set $\{1, \dots, n\}$ and $[n_1, n_2]$ to denote the set $\{n_1, n_1 + 1, \dots, n_2\}$ where $n_1 < n_2$. A *permutation* on $[n]$ is a bijective function from $[n]$ to $[n]$. In this thesis, we use n -tuples to denote functions and permutations.

Definition 2.1.1. (*n-tuples*) A function $f = (a_1, a_2, \dots, a_n)$ if and only if $f(i) = a_i$ for every $i \in [n]$.

Let S_n denote the set of all permutations on $[n]$. S_n is a group with the composition operation, since the composition of two bijective functions from $[n]$ to $[n]$ is still a bijective function from $[n]$ to $[n]$. We define the *product* of two permutations f and $g \in S_n$ as

$$fg = (f(g(1)), \dots, f(g(n))).$$

The *identity* in S_n is $e_n = (1, \dots, n)$. f^m represents the m -th power of a permutation f on $[n]$, and we define $f^0 = e_n$ and $f^m = f f^{m-1}$ for $m > 0$. A permutation f of order k on $[n]$ if and only if k is the minimum positive integer such that $f^k = e_n$. We say that $\{g_1, \dots, g_k\}$ is a generator set for a subgroup $G \subseteq S_n$, if every permutation $g \in G$ can be written as a product of a sequence of compositions from elements in the generator set.

To represent a permutation, there is an alternative other than using an n -tuple. We can write a permutation as a product of cycles. A cycle $\overline{(a_1, a_2, \dots, a_p)}$ represents a permutation π such that

- $\pi(a_p) = a_1$.
- $\pi(a_i) = a_{i+1}$ for $i \in [p-1]$.
- $\pi(x) = x$ for $x \notin \{a_i : i \in [p]\}$. We say two cycles $\overline{(a_1, \dots, a_p)}$ and $\overline{(b_1, \dots, b_q)}$ are disjoint if $\{a_1, \dots, a_p\} \cap \{b_1, \dots, b_q\} = \emptyset$. Every permutation $\pi \in S_n$ can be written as a product of disjoint cycles.

Definition 2.1.2. (*Products of cycles*) For $0 = p_0 < p_1 < \dots < p_q < p_{q+1} = n$ and $\{a_1, \dots, a_n\} = [n]$, a permutation

$$\pi = \overline{(a_1, \dots, a_{p_1})} \overline{(a_{p_1+1}, \dots, a_{p_2})} \dots \overline{(a_{p_q+1}, \dots, a_n)}$$

if and only if

$$\pi(a_i) = \begin{cases} a_{i+1} & , i \notin \{p_1, \dots, p_{q+1}\}. \\ a_{p_{j-1}+1} & , i \in \{p_1, \dots, p_{q+1}\} \text{ and } i = p_j. \end{cases}$$

For example, $(3, 2, 1, 5, 6, 4, 7) = \overline{(1, 3)} \overline{(2)} \overline{(4, 5, 6)} \overline{(7)}$. For simplicity, we sometimes omit the cycles of one element, i.e., $\overline{(1, 3)} \overline{(2)} \overline{(4, 5, 6)} \overline{(7)} = \overline{(1, 3)} \overline{(4, 5, 6)}$. When written in the product of cycles notation, the structure of a permutation is clearer. Now, we show how to relabel a permutation without changing its structure.

Lemma 2.1.3. For permutations f and $g = \overline{(a_1, \dots, a_{p_1})} \overline{(a_{p_1+1}, \dots, a_{p_2})} \dots \overline{(a_{p_q+1}, \dots, a_n)}$ on $[n]$, we have

$$fgf^{-1} = \overline{(f(a_1), \dots, f(a_{p_1}))} \overline{(f(a_{p_1+1}), \dots, f(a_{p_2}))} \overline{(f(a_{p_q+1}), \dots, f(a_n))}$$

Proof. We rewrite g as

$$\overline{(a_1, g(a_1), \dots, g^{p_1-1}(a_1))} \overline{(a_{p_1+1}, g(a_{p_1+1}), \dots, g^{p_2-p_1-1}(a_{p_1+1}))} \dots \overline{(a_{p_q+1}, \dots, g^{n-p_q-1}(a_{p_q+1}))}$$

Consider the sequence $f(a_1), fgf^{-1}(f(a_1)), (fgf^{-1})^2(f(a_1)), \dots, (fgf^{-1})^{p_1}(f(a_1))$, they are actually

$$f(a_1), f(g(a_1)), f(g^2(a_1)), \dots, f(g^{p_1-1}(a_1)), f(a_1),$$

since $(fgf^{-1})^k = fg^k f^{-1}$ and $g^{p_1}(a_1) = a_1$. Therefore, the cycle structure of a_1, \dots, a_{p_1} in g and the cycle structure of $f(a_1), \dots, f(a_{p_1})$ in fgf^{-1} are the same. Apply this observation to $f(a_{p_1+1}), \dots, f(a_{p_q+1})$, then we know the lemma is true. \square

Here, we give a definition to metric functions. A function $\delta(\cdot, \cdot) : D \times D \rightarrow \mathbb{R}$ is a metric function if

- $\delta(x, y) \geq 0$ for every $x, y \in D$.
- $\delta(x, y) = 0$ if and only if $x = y$.
- $\delta(x, y) = \delta(y, x)$ for every $x, y \in D$.
- $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$ for every $x, y, z \in D$.

For convenience, when a bold-type alphabet represents an n -tuple, the same alphabet in regular-type with subscript i represents the i -th entry of the n -tuple unless stated otherwise. For example, $\mathbf{x} = (x_1, \dots, x_n)$. For two n -tuples \mathbf{x} and \mathbf{y} , some well known metrics are defined as follows.

Definition 2.1.4. *Hamming distance between \mathbf{x} and \mathbf{y} is defined as*

$$d_H(\mathbf{x}, \mathbf{y}) = |\{i \in [n] : x_i \neq y_i\}|.$$

Definition 2.1.5. *Minkowski distance of order $p > 0$ between \mathbf{x} and \mathbf{y} is defined as*

$$\ell_p(\mathbf{x}, \mathbf{y}) = \sqrt[p]{\sum_{i \in [n]} |x_i - y_i|^p}.$$

Definition 2.1.6. *Chevyshev distance between \mathbf{x} and \mathbf{y} is defined as*

$$\ell_\infty(\mathbf{x}, \mathbf{y}) = \max_{i \in [n]} |x_i - y_i|.$$

The distance between two permutations is the distance between the tuples representing them. We say two permutation (tuples) \mathbf{x} and \mathbf{y} are d -close to each other under metric δ if $\delta(\mathbf{x}, \mathbf{y}) \leq d$. A metric δ is *right-invariant* if and only if $\delta(\mathbf{xz}, \mathbf{yz}) = \delta(\mathbf{x}, \mathbf{y})$ for all permutations \mathbf{x}, \mathbf{y} and \mathbf{z} . We show that Hamming distance, Minkowski distance of order $p > 0$ and Chebyshev distance are right-invariant.

Lemma 2.1.7. *For n -tuples \mathbf{x}, \mathbf{y} and a permutation $\mathbf{z} \in S_n$, we have $d_H(\mathbf{xz}, \mathbf{yz}) = d_H(\mathbf{x}, \mathbf{y})$, $\ell_p(\mathbf{xz}, \mathbf{yz}) = \ell_p(\mathbf{x}, \mathbf{y})$ and $\ell_\infty(\mathbf{xz}, \mathbf{yz}) = \ell_\infty(\mathbf{x}, \mathbf{y})$.*

Proof. By the definitions, we have

$$\begin{aligned}
 d_H(\mathbf{xz}, \mathbf{yz}) &= |\{i \in [n] : x_{\mathbf{z}(i)} \neq y_{\mathbf{z}(i)}\}| \\
 &= |\{j : x_j \neq y_j \text{ for some } i \in [n] \text{ with } j = \mathbf{z}(i)\}| \\
 &= |\{i \in [n] : x_i \neq y_i\}| \\
 &= d_H(\mathbf{x}, \mathbf{y}), \\
 \ell_p(\mathbf{xz}, \mathbf{yz}) &= \sqrt[p]{\sum_{i \in [n]} |x_{\mathbf{z}(i)} - y_{\mathbf{z}(i)}|^p} \\
 &= \sqrt[p]{\sum_{i \in [n]} |x_i - y_i|^p} \\
 &= \ell_p(\mathbf{x}, \mathbf{y}),
 \end{aligned}$$

and

$$\begin{aligned}
 \ell_\infty(\mathbf{xz}, \mathbf{yz}) &= \max_{i \in [n]} |x_{\mathbf{z}(i)} - y_{\mathbf{z}(i)}| \\
 &= \max_{i \in [n]} |x_i - y_i| \\
 &= \ell_\infty(\mathbf{x}, \mathbf{y}).
 \end{aligned}$$

□

We define the weight function $wt_\delta(\mathbf{x})$ for a right-invariant metric δ and a k -tuple \mathbf{x} as $wt_\delta(\mathbf{x}) = \delta(\mathbf{e}_k, \mathbf{x})$. For any right invariant metric δ , the distance between the closest pairs of elements in a subgroup G of S_n is equal to the minimum weight of non-identity permutation

in G , since

$$\min_{\mathbf{x}, \mathbf{y} \in G, \mathbf{x} \neq \mathbf{y}} \delta(\mathbf{x}, \mathbf{y}) = \min_{\mathbf{x}, \mathbf{y} \in G, \mathbf{x} \neq \mathbf{y}} \delta(\mathbf{x}\mathbf{y}^{-1}, \mathbf{e}_n) = \min_{\mathbf{x} \in G, \mathbf{x} \neq \mathbf{e}_n} wt_\delta(\mathbf{x}).$$

Now, we extend the definition of permutations. A *permutation* on a multiset $\{a_1, a_2, \dots, a_k\}$ is a k -tuple $(a_{f(1)}, a_{f(2)}, \dots, a_{f(k)})$ where f is a bijective function from $[k]$ to $[k]$. In 1965, Slepian [36] considered a code of length n for permutation modulation. It consists of all multiple permutations on the multiset

$$\{\overbrace{\mu_1, \dots, \mu_1}^{\lambda_1}, \dots, \overbrace{\mu_m, \dots, \mu_m}^{\lambda_m}\}$$

where $\mu_1 < \mu_2 < \dots < \mu_m$ and $\lambda_1 + \lambda_2 + \dots + \lambda_m = n$. Here, we consider a special case of Slepian's code. Let n, m and λ be positive integers where $n = m\lambda$. Let S_n^λ be the Slepian's code where $\mu_1 = 1, \mu_2 = 2, \dots, \mu_m = m$ and $\lambda_1 = \lambda_2 = \dots = \lambda_m = \lambda$. In other words, S_n^λ is the set of all multiple permutations on $\{\overbrace{1, \dots, 1}^\lambda, \dots, \overbrace{m, \dots, m}^\lambda\}$. In particular, $S_n = S_n^1$. Every element in S_n^λ is called a *frequency permutation* of length n and frequency λ . The identity frequency permutation \mathbf{e}_n^λ in S_n^λ is $(1, \dots, 1, \dots, m, \dots, m)$.

2.2 Coding Theory

Codes are one of the most important objects which have been studied for decades in both computer science and communication engineering, since they have many applications in theoretical fields and practical uses. These applications of codes include error-detection and error-correction in communication, data storage, data compression, cryptography, computational complexity theory, etc. In this section, we give a brief but necessary introduction to coding theory. A *code* C is a set of sequences over a certain symbol set. An element in a code is called a *codeword* which represents a particular *message*. An *encoding* algorithm for code C computes the corresponding codeword in C for any message. A *decoding* algorithm, which is basically the inverse function of an encoding algorithm, recovers the message from a codeword or even from a corrupted one.

We say a code C is a *block code* if every codeword in C has the same length. One of the most important parameters of block codes is the *minimum distance*. The distance between

two codewords can be measured by a certain metric function δ . In different applications or environments, we may choose different metric functions, such as Hamming distance, Minkowski distance, Chebyshev distance, Kendall's tau distance, and etc. The minimum distance $d_\delta(C)$ of a block code C under some metric δ is defined as the minimum distance between distinct codewords x and y in C , i.e.,

$$d_\delta(C) = \min_{x, y \in C} \delta(x, y).$$

It is well known that the error-detection capability and the error-correction capability of a code heavily depend on its minimum distance.

Theorem 2.2.1. (*Error detection*) $0 < \delta(c, x) < d_\delta(C)$ implies $x \notin C$, for any block code C of length n , any codeword $c \in C$ and any n -tuple x .

Proof. By way of contradiction, assume x is a codeword. Then, we have

$$d_\delta(C) \leq \delta(c, x) < d_\delta(C),$$

a contradiction. □

Theorem 2.2.2. (*Error correction*) Given a block code C of length n , a codeword $c \in C$ and an n -tuple x . If $\delta(c, x) < \frac{d_\delta(C)}{2}$, then c is the codeword closest to x .

Proof. By way of contradiction, assume $c' \neq c$ is the closest codeword to x . Since $\delta(\cdot, \cdot)$ is a metric and $\delta(x, c') < \delta(c, x) < \frac{d_\delta(C)}{2}$, we have

$$d_\delta(C) \leq \delta(c, c') \leq \delta(c, x) + \delta(x, c') < d_\delta(C),$$

a contradiction. □

Therefore, a block code is considered as a more powerful error detecting code or a error correcting code when it has larger minimum distance.

Another important parameter of codes is *information rate*. Codes of higher information rate have less space redundancy. The information rate of a block code is the ratio of its message length to its codeword length. The length of a block code C is the average number

of bits to represent a codeword. The message length is $\log_2 |C|$, i.e., the number of bits sufficient to represent a message. In general, codes of larger minimum distance have lower information rate.

One can find the code of largest minimum distance under constraints on the information rate by enumerating all possible candidates. Although this code is the most powerful one against errors among all codes of the same information rate, it does not mean that this code is suitable in practical use. One of the reasons is that we probably do not have the computation power to enumerate every possible codes under such constraints. In other words, the best one does exist, but we may not know what it explicitly is. Another reason is that we probably do not know any efficient encoding and decoding algorithms for this code. Therefore, we need systematic methods to constructing codes for certain codeword lengths, information rates and minimum distances. Moreover, we need codes with efficient encoding and decoding algorithms. In this thesis, we propose several construction to generate codes with predetermined information rate and minimum distance. Some of them have efficient encoding and decoding algorithms.

A (λ, n, d, δ) -FPA C is a subset of S_n^λ and the minimum distance of C under δ . The codeword length of an FPA $C \subseteq S_n^\lambda$ is $\log_2 |S_n^\lambda|$ bits. The message length of an FPA C is $\log_2 |C|$, where $|C|$ is the cardinality of C . Therefore, the information rate of C is $\frac{\log_2 |C|}{\log_2 |S_n^\lambda|}$. For a fixed length n , an FPA C has a smaller symbol set when it has a higher frequency. Thus, FPAs of frequency more than 1 have a higher information rate than PAs of the same length. This is one of the reasons why FPAs can replace PAs in various applications.

2.3 Computational Complexity

In this section, we give some background knowledge about computational complexity theory. There are two main categories of computational problems. The first kind is the *decision problems*. The answer to a decision problem is either ‘yes’ or ‘no’. For example, “Given a real number r , a block code C and a metric δ , is the minimum distance of C under δ less than r ?” is a decision problem. The second kind is the *optimization problems*. In order to define an optimization problem, we need

- An instance set X .
- The set of *feasible solutions* $f(x)$ for every instance $x \in X$.
- A cost function $c(\cdot, \cdot)$, which maps (x, y) into a real number where $x \in X$ and $y \in f(x)$.
- A goal, which is either minimization or maximization.

The answer to an optimization problem on an instance x can be the best feasible solution $y \in f(x)$ or the value of $c(x, y)$, depending on the description of the problem. For example, “Given a block code C and a metric δ , find the minimum distance of C under δ .” Each feasible solution for this problem consists of two distinct codewords y_1 and y_2 . The cost function is $\delta(y_1, y_2)$, the goal of this problem is minimization. In this case, the answer is a number representing the minimum distance of C under δ . There is another version of this problem: “Given a block code C and a metric δ , find the closest pair of distinct codewords in C under δ .” The answer to this version is the feasible solution with the smallest cost. Every optimization problem has a corresponding decision problem which is to ask if a feasible solution of certain cost exists. In this thesis, we discuss both decision problems and optimization problems related to FPAs in Chapter 6.

The *complexity* of an algorithm is measured by how much resource, including time and space, is consumed. It is often a function of the *size* of the *input* of the algorithm. The input size of a problem instance is the length of its description. E.g., the input size of an instance of the problem, “Given a block code C of length k , find the minimum distance of C under the ℓ_∞ -metric”, is $k|C|$, since each codeword in C has length k . An algorithm is considered to be more efficient if it has lower complexity. In general, we say an algorithm is *time efficient* if and only if its time complexity is bounded by a polynomial in terms of its input size, or just simply say, it runs in polynomial time. In Chapter 5, we consider the input size of the encoding and the decoding algorithms as the lengths of message and received frequency permutations, respectively. Therefore, checking all possible codewords is not efficient, since $|C|$ can be an exponential function of the input length.

The complexity of a problem is defined by the complexity of algorithms solving it. Besides providing an explicit algorithm for a problem, we can also use reduction to obtain complexity results on problems.

Definition 2.3.1. A decision problem A is polynomial-time reducible to another decision problem B , denoted as $A \leq_p B$, if there is a polynomial-time computable function f such that

1. f maps instances in A into instances in B .
2. The answer to an instance x in A is ‘yes’ if and only if the answer to the corresponding instance $f(x)$ in B is also ‘yes’.

$A \leq_p B$ means A is not harder than B when polynomial-time algorithms are considerably efficient. If there is an efficient algorithm (in polynomial time) for B , then we can solve instance x in A efficiently by

1. Compute $f(x)$ in polynomial time.
2. Run the efficient algorithm for B on input $f(x)$.
3. Output the result of the algorithm.

Now, we define the following two best known complexity classes P and NP.

Definition 2.3.2. We say a problem A is in P if there is a deterministic algorithm which always output the answer for every instance of it in polynomial time.

Definition 2.3.3. We say a problem A is in NP if there is a polynomial time verifier v such that

- It takes an instance in A and a witness as input.
- For every ‘yes’-instance x , there is a witness w such that $v(x, w)$ accepts.
- For every ‘no’-instance x' and every witness w' , $v(x', w')$ always rejects.

The problems in P are the problems which can be solved efficiently by polynomial-time algorithms, and every ‘yes’-instance of problems in NP can be efficiently verified with some witness. The P versus NP problem is asking whether P and NP are the same, in other words, are the problems in NP as easy as those in P? This problem remains unsolved today, and many researchers believe $P \neq NP$. Recall that algorithms not in polynomial time are considered as inefficient algorithms. We define the best-known classes representing hard problems as follows.

Definition 2.3.4. We say a problem A is NP-hard if every problem $B \in \text{NP}$, $B \leq_p A$.

Definition 2.3.5. We say a problem A is NP-complete if A is in NP and A is NP-hard.

According to the concept of reduction, NP-hard problems are not easier than every NP problem. If an NP-hard problem can be solved in polynomial time, then *every* NP problem can be solved in polynomial time. Hence, we consider NP-hard problems cannot be solved efficiently unless $P = \text{NP}$. Moreover, NP-complete problems are the hardest problems in NP.

The optimization problem is generally harder than its decision version. Consequently, an optimization problem is NP-hard if its decision version is NP-hard. Unless $P = \text{NP}$, we cannot find the optimal solutions for NP-hard optimization problems in polynomial time. So we often seek approximation algorithms for them. We say that an algorithm A is an r -approximate algorithm for a minimization problem (or for a maximization problem) if A always outputs a feasible solution whose cost is no more than r times (no less than $\frac{1}{r}$ times) of the minimum (maximum) cost on any input. Note that A cannot output an answer whose cost is less (larger) than the minimum (maximum) cost for minimization (maximization) problems, since it is not a feasible solution. Unfortunately, some problems do not allow polynomial time r -approximate algorithm for some r , unless $P = \text{NP}$. We call this kind of complexity results as inapproximable results.

Chapter 3

Explicit Lower and Upper Bounds

Let $F(\lambda, n, d, \ell_\infty)$ be the cardinality of the maximum $(\lambda, n, d, \ell_\infty)$ -FPA and $V(\lambda, n, d, \ell_\infty)$ be the number of elements in S_n^λ being d -close to the identity \mathbf{e}_n^λ under ℓ_∞ -metric. In this chapter, we first give a Gilbert type lower bound and a sphere packing upper bound of $F(\lambda, n, d, \ell_\infty)$ by bounding $V(\lambda, n, d, \ell_\infty)$. Then, we introduce an algorithm to enumerate every element \mathbf{x} such that $\ell_\infty(\mathbf{e}_n^\lambda, \mathbf{x}) \leq d$. Furthermore, we generalize the method in [32] to efficiently compute $V(\lambda, n, d, \ell_\infty)$, and we give two implementations. Finally, we leave some open problems about bounding $F(\lambda, n, d, \ell_\infty)$. We use \mathbf{e} to denote \mathbf{e}_n^λ in this chapter, since λ and n are fixed here.

3.1 Gilbert Type and Sphere Packing Bounds

Under Chebyshev distance, an r -radius ball centered at \mathbf{x} in S_n^λ is defined as

$$B(r, \mathbf{x}) = \{\mathbf{y} \in S_n^\lambda : \ell_\infty(\mathbf{x}, \mathbf{y}) \leq r\}.$$

First, we show that any d -radius ball in S_n^λ under ℓ_∞ -metric has the same cardinality, i.e., $|B(d, \mathbf{x})| = V(\lambda, n, d, \ell_\infty)$ for every $\mathbf{x} \in S_n^\lambda$.

Claim 3.1.1. *For any $\mathbf{x} = (x_1, \dots, x_n) \in S_n^\lambda$, there are exactly $V(\lambda, n, d, \ell_\infty)$ \mathbf{y} 's in S_n^λ such that $\ell_\infty(\mathbf{x}, \mathbf{y}) \leq d$.*

Proof. Since every $i \in [m]$ appears exactly λ times in \mathbf{x} , there exists a permutation $\pi \in S_n$

such that $\mathbf{x} = \mathbf{e}\pi$. Recall that ℓ_∞ -metric is right invariant. As a consequence, we have that $\ell_\infty(\mathbf{e}, \mathbf{z}) = \ell_\infty(\mathbf{x}, \mathbf{z}\pi)$ for any $\mathbf{z} \in S_n^\lambda$. Let $Y = \{\mathbf{w}\pi : \mathbf{w} \in B(d, \mathbf{x})\}$ and $\bar{Y} = S_n^\lambda \setminus Y$. For any $\mathbf{y} \in Y$, we have $\ell_\infty(\mathbf{x}, \mathbf{y}) = \ell_\infty(\mathbf{e}, \mathbf{y}\pi^{-1}) \leq d$, since $\mathbf{y}\pi^{-1} \in B(d, \mathbf{x})$. While for $\mathbf{y}' \in \bar{Y}$, $\ell_\infty(\mathbf{x}, \mathbf{y}') = \ell_\infty(\mathbf{e}, \mathbf{y}'\pi^{-1}) > d$. Therefore, only $|Y| = |B(d, \mathbf{e})| = V(\lambda, n, d, \ell_\infty)$ permutations in S_n^λ are d -close to \mathbf{x} . \square

The first inequality in the following theorem is a Gilbert type lower bound, and the second one is a sphere packing upper bound.

Theorem 3.1.2.

$$\frac{|S_n^\lambda|}{V(\lambda, n, d-1, \ell_\infty)} \leq F(\lambda, n, d, \ell_\infty) \leq \frac{|S_n^\lambda|}{V(\lambda, n, \lfloor \frac{d-1}{2} \rfloor, \ell_\infty)}.$$

Proof. To prove the lower bound, we use the following algorithm to generate a $(\lambda, n, d, \ell_\infty)$ -FPA with size at least $\frac{|S_n^\lambda|}{V(\lambda, n, d-1, \ell_\infty)}$.

1. $C \leftarrow \emptyset$, $D \leftarrow S_n^\lambda$.
2. Add an arbitrary $\mathbf{x} \in D$ to C , then remove all frequency permutations in $B(d-1, \mathbf{x})$ from D .
3. If $D \neq \emptyset$ then repeat step 2, otherwise output C .

For $\mathbf{x} \in C$, we remove all frequency permutations $(d-1)$ -close to \mathbf{x} in step 2. Thus, C has minimum distance at least d . D has initially $|S_n^\lambda|$ elements and each iteration of step 2 removes at most $V(\lambda, n, d-1, \ell_\infty)$, so we conclude

$$F(\lambda, n, d, \ell_\infty) \geq |C| \geq \frac{|S_n^\lambda|}{V(\lambda, n, d-1, \ell_\infty)}.$$

Now we turn to the upper bound. Consider a $(\lambda, n, d, \ell_\infty)$ -FPA C^* with the maximum cardinality. Any two $\lfloor \frac{d-1}{2} \rfloor$ -radius balls centered at distinct frequency permutations in C^* do not have any common elements, since the minimum distance is d . In other words, the $\lfloor \frac{d-1}{2} \rfloor$ -radius balls centered at frequency permutations in C^* are all disjoint. We have

$$F(\lambda, n, d, \ell_\infty) = |C^*| \leq \frac{|S_n^\lambda|}{V(\lambda, n, \lfloor \frac{d-1}{2} \rfloor, \ell_\infty)}.$$

□

It is clear that $|S_n^\lambda| = \frac{n!}{(\lambda!)^{n/\lambda}}$. It is already known that $V(1, n, d, \ell_\infty)$ equals to the permanent of some special matrix [28]. We generalize previous analysis to give asymptotic bounds for Theorem 3.1.2. The permanent of an $n \times n$ matrix $A = (a_{i,j})$ is defined as

$$\text{per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^n a_{i,\pi(i)}.$$

Define a symmetric $n \times n$ matrix $A^{(\lambda,n,d)} = (a_{i,j}^{(\lambda,n,d)})$, where $a_{i,j}^{(\lambda,n,d)} = 1$, if $|\lceil \frac{i}{\lambda} \rceil - \lceil \frac{j}{\lambda} \rceil| \leq d$; else $a_{i,j}^{(\lambda,n,d)} = 0$. Note that a frequency permutation \mathbf{x} is d -close to \mathbf{e} if and only if $a_{i,x_i}^{(\lambda,n,d)} = 1$ for every $i \in [n]$. Now, we consider $A^{(\lambda,n,d)}$, and recall that $n = m\lambda$. Since the λ copies of a symbol are considered identical while computing the distance and the entries indexed from $(k\lambda - \lambda + 1)$ to $k\lambda$ of \mathbf{e} represent the same symbol for every $k \in [m]$. It implies that row $(k\lambda - \lambda + 1)$ through row $k\lambda$ of $A^{(\lambda,n,d)}$ are identical and so are columns indexed from $(k\lambda - \lambda + 1)$ to $k\lambda$ for every $k \in [m]$. Thus, we have $A^{(\lambda,n,d)} = A^{(1,m,d)} \otimes \mathbf{1}_\lambda$ where \otimes is the operator of tensor product and $\mathbf{1}_\lambda$ is a $\lambda \times \lambda$ matrix with all entries equal to 1. For example, take $\lambda = 2$, $m = 5$ and $d = 2$:

$$A^{(1,5,2)} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix},$$

$$\mathbf{1}_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

$$A^{(2,10,2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Let $r_i^{(1,m,d)}$ be the row sum of $A^{(1,m,d)}$'s i -th row. We have:

$$r_i^{(1,m,d)} = \begin{cases} d+i & \text{if } i \leq d, \\ 2d+1 & \text{if } d < i \leq m-d, \\ m-i+1+d & \text{if } i > m-d. \end{cases}$$

Then for $i \in [m]$ and $j \in [\lambda]$, the row sum of the $(i\lambda + j)$ -th row of $A^{(\lambda,n,d)}$ is $\lambda r_i^{(1,m,d)}$, due to $A^{(\lambda,n,d)} = A^{(1,m,d)} \otimes \mathbf{1}_\lambda$. We first calculate $V(\lambda, n, d, \ell_\infty)$ by using $\text{per}(A^{(\lambda,n,d)})$.

Lemma 3.1.3.

$$V(\lambda, n, d, \ell_\infty) = \frac{\text{per}(A^{(\lambda,n,d)})}{(\lambda!)^{n/\lambda}}.$$

Proof.

$$\begin{aligned} & \text{per}(A^{(\lambda,n,d)}) \\ &= |\{\pi \in S_n : \forall i, a_{i,\pi(i)}^{(\lambda,n,d)} = 1\}| \\ &= |\{\pi \in S_n : \max_i |\lceil \frac{i}{\lambda} \rceil - \lceil \frac{\pi(i)}{\lambda} \rceil| \leq d\}| \\ &= (\lambda!)^{n/\lambda} |\{\mathbf{y} \in S_n^\lambda : \max_i |\lceil \frac{i}{\lambda} \rceil - y_i| \leq d\}| \\ &= (\lambda!)^{n/\lambda} |\{\mathbf{y} \in S_n^\lambda : \ell_\infty(\mathbf{e}, \mathbf{y}) \leq d\}| \\ &= (\lambda!)^{n/\lambda} V_\infty(\lambda, n, d, \ell_\infty) \end{aligned}$$

The first equality holds since $A^{(\lambda,n,d)}$ is a $(0, 1)$ -matrix and by the definition of permanent. We can convert $\pi \in S_n$ into $\mathbf{y} = (y_1, \dots, y_n) \in S_n^\lambda$ by setting $y_i = \lceil \frac{\pi(i)}{\lambda} \rceil$, and there are exactly

$(\lambda!)^{n/\lambda}$ such π 's in S_n converted to the same \mathbf{y} . Thus, we know the third equality holds. Therefore, the lemma holds by moving $(\lambda!)^{n/\lambda}$ to the left-hand side of the equation. \square

We still need to estimate $\text{per}(A^{(\lambda,n,d)})$ in order to get asymptotic bounds. Kløve [23] reports some bounds and methods to approximate $\text{per}(A^{(1,n,d)})$. We extend his analysis for $\text{per}(A^{(\lambda,n,d)})$.

Lemma 3.1.4.

$$\text{per}(A^{(\lambda,n,d)}) \leq [(2d\lambda + \lambda)!]^{\frac{n}{2d\lambda + \lambda}}.$$

Proof. It is known (Theorem 11.5 in [39]) that for $(0,1)$ -matrix A , $\text{per}(A) \leq \prod_{i=1}^n (r_i!)^{\frac{1}{r_i}}$ where r_i is the sum of the i -th row. Since the sum of any row of $A^{(\lambda,n,d)}$ is at most $2d\lambda + \lambda$, we have

$$\text{per}(A) \leq \prod_{i=1}^n [(2d\lambda + \lambda)!]^{\frac{1}{2d\lambda + \lambda}} = [(2d\lambda + \lambda)!]^{\frac{n}{2d\lambda + \lambda}}$$

\square

We give $\text{per}(A)^{(\lambda,n,d)}$ a lower bound by using the van der Waerden permanent theorem (see p.104 in [39]): *the permanent of an $n \times n$ doubly stochastic matrix A (i.e., A has nonnegative entries, and every row sum and column sum of A is 1.) is no less than $\frac{n!}{n^n}$.* Unfortunately, $A^{(\lambda,n,d)}$ is not a doubly stochastic matrix, since the row sums and columns sums range from $d\lambda + \lambda$ to $2d\lambda + \lambda$. We estimate the lower bound via a matrix derived from $A^{(\lambda,n,d)}$ as follows.

Lemma 3.1.5.

$$\text{per}(A^{(\lambda,n,d)}) \geq \frac{(2d\lambda + \lambda)^n}{2^{2d\lambda}} \cdot \frac{n!}{n^n}.$$

Proof. Let $\tilde{A} = \frac{1}{2d\lambda + \lambda} A^{(\lambda,n,d)}$, which has the sum of any row or column bounded by 1, but is not a doubly stochastic matrix. Observe that every row sum of \tilde{A} is 1 except the first $d\lambda$ and last $d\lambda$ rows. For $i \in [d]$ and $j \in [\lambda]$, both row $(i\lambda - \lambda + j)$ and row $(n - i\lambda + j)$ sum to $\frac{d+i}{2d+1}$. Now we construct an $n \times n$ matrix B from \tilde{A} with each row sum equal to 1 as follows:

For $i \in [d]$ and $j \in [\lambda]$, add $\frac{1}{2d\lambda + \lambda}$ to

- 1) The first $(d - i + 1)\lambda$ entries of row $(i\lambda - \lambda + j)$.
- 2) The last $(d - i + 1)\lambda$ entries of row $(n - i\lambda + j)$.

The row sums of the first $d\lambda$ and last $d\lambda$ rows of B are now

$$\frac{(d-i+1)\lambda}{2d\lambda+\lambda} + \frac{d+i}{2d+1} = 1.$$

We turn to check the column sums of B . Since \tilde{A} is symmetric and by the definition of B , we know B is symmetric as well. Thus we have that B is doubly stochastic and $\text{per}(B) \geq \frac{n!}{n^n}$. With this inequality, we obtain a bound on $\text{per}(A^{(\lambda,n,d)})$. Observe that the entries of the first $d\lambda$ and last $d\lambda$ rows of B are at most $\frac{2}{2d\lambda+\lambda}$ times of the corresponding entries of $A^{(\lambda,n,d)}$, and the other rows are exactly $\frac{1}{2d\lambda+\lambda}$ times of the corresponding rows of $A^{(\lambda,n,d)}$. We have

$$\text{per}(A^{(\lambda,n,d)}) \geq \frac{(2d\lambda+\lambda)^n}{2^{2d\lambda}} \text{per}(B) \geq \frac{(2d\lambda+\lambda)^n}{2^{2d\lambda}} \frac{n!}{n^n}.$$

□

With Lemma 3.1.4 and Lemma 3.1.5, we have the asymptotic bounds as follows.

Theorem 3.1.6.

$$\frac{n!}{[(2d\lambda-\lambda)!]^{\frac{n}{2d\lambda-\lambda}}} \leq F(\lambda, n, d, \ell_\infty) \leq \frac{2^{2\lambda \cdot \lfloor \frac{d-1}{2} \rfloor} n^n}{(2\lambda \cdot \lfloor \frac{d-1}{2} \rfloor + \lambda)^n}.$$

3.2 Enumerate Permutations in a Ball

In this section, we give a recursive algorithm to enumerate all frequency permutations in $B(d, \mathbf{e})$. For convenience, let x_i denote the i -th entry of \mathbf{x} in the rest of this section. First, we investigate \mathbf{e} closely.

i	\cdots	$k\lambda - \lambda$	$k\lambda - \lambda + 1$	\cdots	$k\lambda$	$k\lambda + 1$	\cdots
$\mathbf{e}(i)$	\cdots	$k - 1$	k	\cdots	k	$k + 1$	\cdots

Observe that symbol k appears at the $(k\lambda - \lambda + 1)$ -th, \dots , $(k\lambda)$ -th positions in \mathbf{e} . Therefore, \mathbf{x} is d -close to \mathbf{e} if and only if $x_{k\lambda-\lambda+1}, \dots, x_{k\lambda} \in [k-d, k+d]$. (Note that we simply ignore the non-positive values when $k \leq d$.) In other words, $\ell_\infty(\mathbf{e}, \mathbf{x}) \leq d$ if and only if symbol k

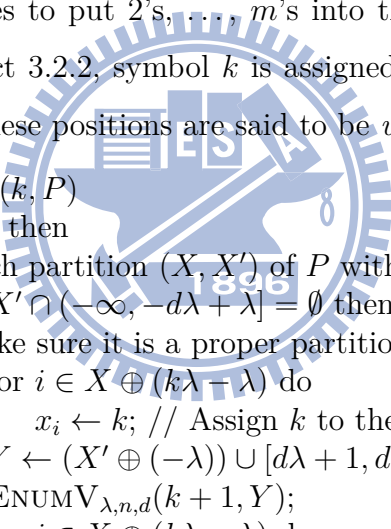
only appear in the $(k\lambda - d\lambda - \lambda + 1)$ -th, \dots , $(k\lambda + d\lambda)$ -th positions of \mathbf{x} . This observation leads us to define the shift operator \oplus .

Definition 3.2.1. For an integer set S and an integer z , define $S \oplus z = \{s + z : s \in S\}$.

Fact 3.2.2. If $\mathbf{x} = (x_1, \dots, x_n)$ is d -close to \mathbf{e} , then $x_i = k$ 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 \mathbf{e} . Note that the set $S = [-d\lambda + 1, d\lambda + \lambda]$ is independent of k . We give a recursive algorithm $\text{ENUMV}_{\lambda,n,d}$ in Figure 3.1. It enumerates all frequency permutations $\mathbf{x} \in S_n^\lambda$ in $B(d, \mathbf{e})$. It is a depth-first-search style algorithm. Basically, it first tries to put 1's into λ proper vacant positions of \mathbf{x} . Then, it tries to put 2's, \dots , m 's into the partial frequency permutations recursively. According to fact 3.2.2, 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 .



```

ENUMVλ,n,d( $k, P$ )
1. if  $k \leq m$  then
2.   for each partition  $(X, X')$  of  $P$  with  $|X| = \lambda$  do
3.     if  $X' \cap (-\infty, -d\lambda + \lambda] = \emptyset$  then
4.       // Make sure it is a proper partition
5.       for  $i \in X \oplus (k\lambda - \lambda)$  do
6.          $x_i \leftarrow k$ ; // Assign  $k$  to the  $i$ -th position
7.        $Y \leftarrow (X' \oplus (-\lambda)) \cup [d\lambda + 1, d\lambda + \lambda]$ ;
8.       ENUMVλ,n,d( $k + 1, Y$ );
9.     for  $i \in X \oplus (k\lambda - \lambda)$  do
10.       $x_i \leftarrow 0$ ; // Reset  $x_i$  to be vacant
11. else
12.   if  $P = [d\lambda + \lambda]$  then output  $(x_1, \dots, x_n)$ ;

```

Figure 3.1: $\text{ENUMV}_{\lambda,n,d}(k, P)$

$\text{ENUMV}_{\lambda,n,d}$ takes an integer k and a subset P of $[-d\lambda + 1, d\lambda + \lambda]$ as its input, and $\text{ENUMV}_{\lambda,n,d}$ uses an $(n + 2d\lambda)$ -dimensional vector \mathbf{x} as a global variable. For convenience, we extend the index set of \mathbf{x} to $[-d\lambda + 1, n + d\lambda]$ and every entry of \mathbf{x} 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 $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ to enumerate all frequency permutations which are d -close to \mathbf{e} . During the enumeration, $\text{ENUMV}_{\lambda,n,d}(k, P)$ assigns symbol k into some λ positions, indexed by $X \oplus (k\lambda - \lambda)$, of a partially assigned frequency permutation, then it recursively invokes $\text{ENUMV}_{\lambda,n,d}(k+1, X' \oplus (-\lambda) \cup [d\lambda + 1, d\lambda + \lambda])$, where X and X' form a partition of P and $|X| = \lambda$. After the recursive call is done, $\text{ENUMV}_{\lambda,n,d}(k, P)$ reset 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 $k = m+1$, $\text{ENUMV}_{\lambda,n,d}(k, P)$ outputs \mathbf{x} if $P = [d\lambda + \lambda]$. Given $n = m\lambda$, k is initialized to 1 and P to $[d\lambda + \lambda]$, we have the following claims.

Claim 3.2.3. *In each of the recursive call of $\text{ENUMV}_{\lambda,n,d}$, 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 $\max(X') \leq d\lambda + \lambda$, we have $\max(Y) = d\lambda + \lambda$. \square

Claim 3.2.4. *In line 6, for each $k \in [m+1]$ and each $i \in Y \oplus ((k+1)\lambda - \lambda)$, we have $x_i = 0$.*

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

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 $x_i = 0$ in these entries. \square

Claim 3.2.5. *In each recursive call of $\text{ENUMV}_{\lambda,n,d}(k, P)$, P must be a subset of $[-d\lambda + 1, d\lambda + \lambda]$ of cardinality $d\lambda + \lambda$. It implies, $|e_i - k| \leq d$ for $i \in P \oplus (k\lambda - \lambda)$.*

Proof. We prove it 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 $\text{ENUMV}_{\lambda,n,d}(k, P)$ invokes $\text{ENUMV}_{\lambda,n,d}(k+1, Y)$. 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 \mathbf{e} has values from $[k - d + 1, k + d + 1]$ in these positions. I.e., $|e_i - (k + 1)| \leq d$ for $i \in Y \oplus (k\lambda)$. Hence, the claim is true. \square

Claim 3.2.6. *At the beginning of the invocation of $\text{ENUMV}_{\lambda,n,d}(k, P)$, $i \in P \oplus (k\lambda - \lambda)$ implies $i > 0$.*

Proof. It is clear for $k = 1$. Observe that $\min(Y) \geq \min(P) - \lambda$. Since

$$(\min(P) - \lambda) \oplus (k\lambda) = \min(P) \oplus (k\lambda - \lambda),$$

the claim holds for $k > 1$. \square

Claim 3.2.7. *For $k \in [m]$, when $\text{ENUMV}_{\lambda,n,d}(k, P)$ invoke $\text{ENUMV}_{\lambda,n,d}(k + 1, Y)$ in line 7, there are exactly λ entries of \mathbf{x} equal i for $i \in [k - 1]$.*

Proof. It is implied by lines 4 and 5. \square

Lemma 3.2.8. *At the beginning of the execution of $\text{ENUMV}_{\lambda,n,d}(k, P)$,*

$$P \oplus (k\lambda - \lambda) = \{i : i > 0 \wedge x_i = 0 \wedge i \in [-d\lambda + 1, d\lambda + \lambda] \oplus (k\lambda - \lambda)\}.$$

Proof. The lemma holds by claims 3.2.4, 3.2.5, and 3.2.6. \square

In order to investigate the behavior of $\text{ENUMV}_{\lambda,n,d}$, we define *partial frequency permutations*. 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., Chebyshev distance between two k -dimensional partial frequency permutations, \mathbf{y} and \mathbf{y}' , is defined as $\ell_\infty(\mathbf{y}, \mathbf{y}') = \max_{i \in [k], y_i \neq *, y'_i \neq *} |y_i - y'_i|$. Now, we begin to prove that $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ only outputs the frequency permutations d -close to \mathbf{e} .

Lemma 3.2.9. *Let τ_k be a partial frequency permutation d -close to \mathbf{e} and with each symbol $1, \dots, k-1$ appearing exactly λ times in τ_k for some $k \in [m+1]$. If the entries of the global variable \mathbf{x} are configured as*

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

then every frequency permutation \mathbf{y} in S_n^λ , generated by $\text{ENUMV}_{\lambda,n,d}(k, P)$, satisfies the following conditions:

1. \mathbf{y} is consistent with τ_k over the entries with symbols $1, \dots, k-1$.
2. \mathbf{y} is d -close to \mathbf{e} .

Proof. We prove it by reverse induction. First, we consider the case $k = m+1$. By claim 3.2.7, every symbol appears exactly λ times in τ_{m+1} and \mathbf{x} . By claim 3.2.4 and claim 3.2.6, we know that all of x_1, \dots, x_n are nonzero if and only if $P = [d\lambda + \lambda]$. There are two possible cases:

- $P = [d\lambda + \lambda]$: By claim 3.2.5, (x_1, \dots, x_n) is the only frequency permutation in S_n^λ satisfying both conditions.
- $P \neq [d\lambda + \lambda]$: Then there is some $i \in [n]$ such that $x_i = 0$. But there is no frequency permutation \mathbf{y} in S_n^λ with $y_i = m+1 > m$. This means \mathbf{x} is not well assigned.

Note that $\text{ENUMV}_{\lambda,n,d}(k, P)$ outputs \mathbf{x} only if $P = [d\lambda + \lambda]$, otherwise there is no output. Hence, the claim is true for $k = m+1$. Assume the claim is true down to $k+1$. For k , by lemma 3.2.8, $P \oplus (k\lambda - \lambda)$ is exactly the set of all positions which are vacant and valid for k . In order to enumerate frequency permutations satisfying the second condition, we must assign k 's into the valid positions. Therefore, we just need to try all possible choices of λ -element subset $X \oplus (k\lambda - \lambda) \subseteq P \oplus (k\lambda - \lambda)$. Line 3 of $\text{ENUMV}_{\lambda,n,d}(k, P)$ ensures that X is properly selected. Then symbol k is assigned to $X \oplus (k\lambda - \lambda)$ and we have a new partial frequency permutation τ_{k+1} where

$$(\tau_{k+1})_i = \begin{cases} x_i & , x_i \neq 0, \\ * & , \text{otherwise.} \end{cases}$$

By induction hypothesis, it is clear that the generated frequency permutations satisfy both conditions. \square

We have the following theorem as an immediate result of lemma 3.2.9.

Theorem 3.2.10. $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ enumerates exactly all the frequency permutations d -close to \mathbf{e} in S_n^λ .

Proof. By lemma 3.2.9, $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ only outputs frequency permutations d -close to \mathbf{e} . The rest is to show that every frequency permutation d -close to \mathbf{e} will be enumerated. Let \mathbf{y} be a frequency permutation d -close to \mathbf{e} in S_n^λ . We define $\mathbf{x}^{<1}, \dots, \mathbf{x}^{<m+1}$ as

$$\mathbf{x}_i^{<k} = \begin{cases} y_i & , y_i < k, \\ 0 & , y_i \geq k. \end{cases}$$

Note that $\mathbf{x}^{<1} = (0, \dots, 0)$, $\mathbf{x}^{<m+1} = \mathbf{y}$, and for $k \in [m]$, $\text{ENUMV}_{\lambda,n,d}(k, P)$ with \mathbf{x} initialized to $\mathbf{x}^{<k}$ must invokes $\text{ENUMV}_{\lambda,n,d}(k+1, Y)$ with \mathbf{x} initialized to $\mathbf{x}^{<k+1}$ in some iteration of the for-loop. Thus, \mathbf{y} will be enumerated eventually. \square

3.3 Compute the Ball Size

The number of elements generated by $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ is $V(\lambda, n, d, \ell_\infty)$, since the algorithm $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$ enumerates $B(d, \mathbf{e})$. However, the enumeration is not efficient, since $V(\lambda, n, d, \ell_\infty)$ is usually a very large number. In this section, we give two efficient implementations to compute $V(\lambda, n, d, \ell_\infty)$. Especially, $V(\lambda, n, d, \ell_\infty)$ can be computed in sublinear time for constant d and λ .

From the algorithm $\text{ENUMV}_{\lambda,n,d}$, we see that whether $\text{ENUMV}_{\lambda,n,d}(k, P)$ recursively invokes $\text{ENUMV}_{\lambda,n,d}(k+1, Y)$ or not depends only on k , P and Y . During the execution of $\text{ENUMV}_{\lambda,n,d}(1, [d\lambda + \lambda])$, $\text{ENUMV}_{\lambda,n,d}(k, P)$ 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 $\text{ENUMV}_{\lambda,n,d}(k, U \cup [d\lambda + 1, d\lambda + \lambda])$ invokes $\text{ENUMV}_{\lambda,n,d}(k + 1, V \cup [d\lambda + 1, d\lambda + \lambda])$.

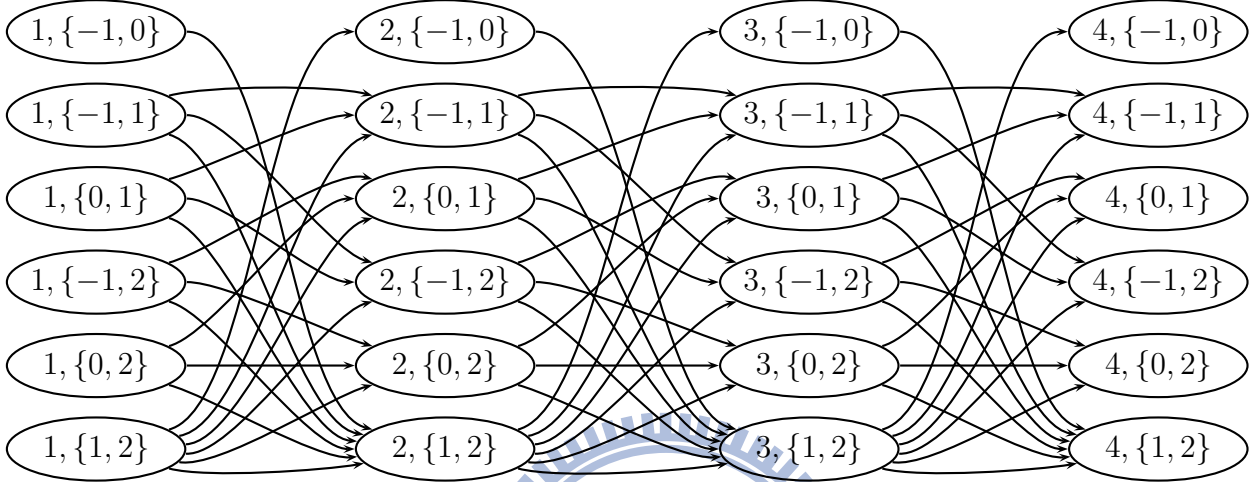


Figure 3.2: Graph $G_{2,6,1}$

For example, figure 3.2 shows the structure of $G_{2,6,1}$.

$V(\lambda, n, d, \ell_\infty)$ equals the number of invocations of $\text{ENUMV}_{\lambda,n,d}(m + 1, [d\lambda + \lambda])$. With this observation, $V(\lambda, n, d, \ell_\infty)$ 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 $\mathcal{O}(|V| + |E|)$, where $|V| = (m + 1) \binom{2d\lambda}{d\lambda}$ and $|E| = \mathcal{O}(|V|^2)$. So $V(\lambda, n, d, \ell_\infty)$ can be calculated in polynomial time with respect to n if λ and d are constants.

The computation actually can be done in $\mathcal{O}(\log n)$ for constant λ and d . We 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 $k \in [m]$ such that

$$\text{ENUMV}_{\lambda,n,d}(k, P \cup [d\lambda + 1, d\lambda + \lambda])$$

invokes

$$\text{ENUMV}_{\lambda,n,d}(k+1, P' \cup [d\lambda+1, d\lambda+\lambda]).$$

Note that $|V_H| = \binom{2d\lambda}{d\lambda}$. Figure 3.3 shows $H_{2,1}$ as an example.

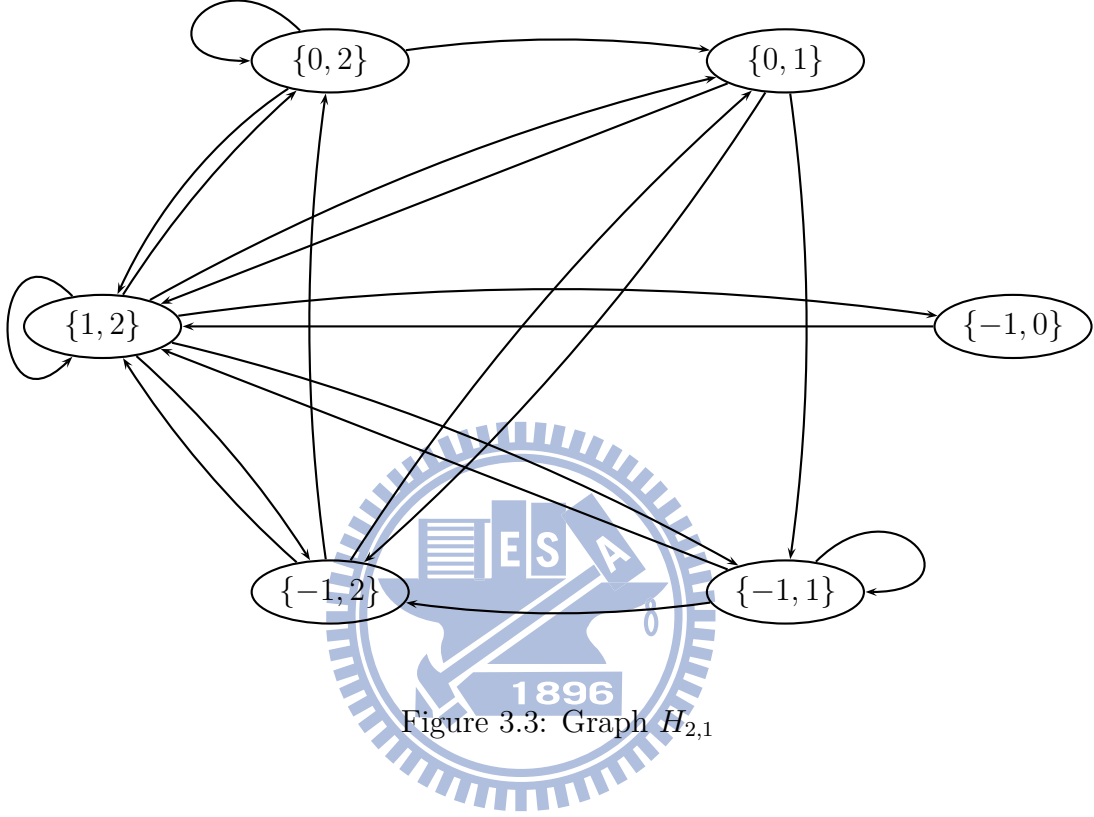


Figure 3.3: Graph $H_{2,1}$

Theorem 3.3.1. $V(\lambda, n, d, \ell_\infty)$ can be computed in $\mathcal{O}(\log n)$ time for constant d and λ .

Proof. Observe that the value of $k \in [m]$ is independent of the invocation of $\text{ENUMV}_{\lambda,n,d}(k+1, P' \cup [d\lambda+1, d\lambda+\lambda])$ by $\text{ENUMV}_{\lambda,n,d}(k, P \cup [d\lambda+1, d\lambda+\lambda])$, 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 $V_H = \{v_1, \dots, v_{|V_H|}\}$, where $v_1 = [d\lambda]$. The number of paths of length m from v_1 to v_1 is the first entry of the first column of the m -th power of A_H , where A_H is the adjacency matrix of $H_{\lambda,d}$. Since m -th power can be computed in $\mathcal{O}(f(\binom{2d\lambda}{d\lambda}) \log m)$, where $\mathcal{O}(f(r))$ is the time cost of multiplying two $r \times r$ matrices. It is well-known that $f(r) = \mathcal{O}(r^{2.376})$ by the Coppersmith-Winograd algorithm. With constants λ and d , $V_{\lambda,n,d}$ can be found in $\mathcal{O}(\log n)$ time. \square

However, the space to store the adjacency matrix A_H , $\Omega\left(\binom{2d\lambda}{d\lambda}^2\right)$, can be too large to execute the $\mathcal{O}(\log n)$ -time algorithm. For example, by setting $d = 3$ and $\lambda = 3$, we need at least $\binom{18}{9}^2 \approx 2.36 \times 10^9$ entries to store A_H^m . This makes the constant factor extraordinarily large in the proof of theorem 3.3.1. Hence, we provide an alternative implementation which runs in $\mathcal{O}\left(\binom{2d\lambda}{d\lambda} \cdot \binom{d\lambda+\lambda}{\lambda} \cdot m\right)$ time and $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$ space. This allows us to compute more efficiently for the cases with smaller m and larger d and λ . For example, $\binom{2d\lambda}{d\lambda} \cdot \binom{d\lambda+\lambda}{\lambda} \cdot m \approx 10^9$ for $d = 3, \lambda = 3$, and $m = 100$. To achieve the $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$ -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 $\mathbf{y} = (y_1, \dots, y_{|V_H|})$ and $\mathbf{y}' = A_H \mathbf{y} = (y'_1, \dots, y'_{|V_H|})$, 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 $A_H \mathbf{y}$ in $\mathcal{O}(|V_H| + S)$ space and $\mathcal{O}(T)$ time for any $|V_H|$ -dimensional vector \mathbf{y} .

Lemma 3.3.2. $|E_H| \leq |V_H| \cdot \binom{d\lambda+\lambda}{\lambda}$ and E_H can be enumerated in $\mathcal{O}(d\lambda)$ space and $\mathcal{O}(|E_H|)$ 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 $\binom{d\lambda+\lambda-r}{\lambda-r} \leq \binom{d\lambda+\lambda}{\lambda}$, 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 $\mathcal{O}(d\lambda + \lambda) = \mathcal{O}(d\lambda)$ space and $\mathcal{O}\left(\binom{d\lambda+\lambda}{\lambda}\right)$ time. Since we can recycle the space, the enumeration of edges in E_H can be done in $\mathcal{O}(d\lambda)$ space and $\mathcal{O}(|E_H|)$ time. \square

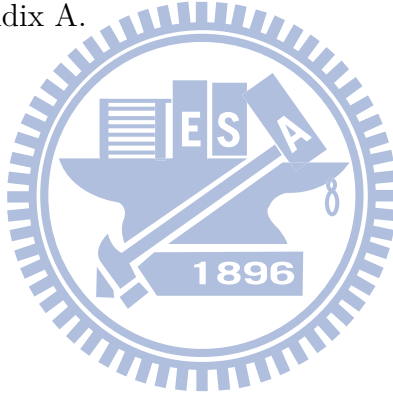
Now, we give the alternative implementation.

Theorem 3.3.3. $V(\lambda, n, d, \ell_\infty)$ can be computed in $\mathcal{O}\left(\binom{2d\lambda}{d\lambda} \cdot \binom{d\lambda+\lambda}{\lambda} \cdot m\right)$ time and $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$ space.

Proof. Let $\mathbf{x} = (1, 0, \dots, 0)^T$. Since $A_H^m \mathbf{x}$ is the first column of A_H^m , $V(\lambda, n, d, \ell_\infty)$ is the first entry of $A_H^m \mathbf{x}$. The alternative evaluates $A_H^1 \mathbf{x}, \dots, A_H^m \mathbf{x}$ iteratively. Instead of storing the whole adjacency matrix, it only uses two $|V_H|$ -dimension vectors \mathbf{y} and \mathbf{y}' for storing $A_H^i \mathbf{x}$ and the intermediate result of $A_H^{i+1} \mathbf{x}$, respectively. Initially, $\mathbf{y} = \mathbf{x}$ and $i = 0$. We compute

$A_H \mathbf{y}$ by the algorithm described in lemma 3.3.2 and using \mathbf{y}' to store the intermediate result. Then, we copy the result of $A_H \mathbf{y}$ back to \mathbf{y} . After m repetitions, we have $\mathbf{y} = A_H^m \mathbf{x}$, and the first entry of \mathbf{y} is $V(\lambda, n, d, \ell_\infty)$. Therefore, the space complexity can be reduced to $\mathcal{O}(|V_H| + d\lambda) = \mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$. The running time is $m \cdot \mathcal{O}(|E_H|) = \mathcal{O}\left(\binom{2d\lambda}{d\lambda} \cdot \binom{d\lambda+\lambda}{\lambda} \cdot m\right)$. \square

A naive approach to evaluate the permanent of an $n \times n$ matrix takes $\mathcal{O}(n!)$ time. In practice, $\Theta((\lambda!)^m V(\lambda, n, d, \ell_\infty))$ time is still required when using a backtracking algorithm. It is clear that both of our methods are much faster. Kløve [24] solved the recurrence of $V(\lambda, n, d, \ell_\infty)$, and gave the value of $V(\lambda, n, d, \ell_\infty)$ for $\lambda \in [10]$, $m \in [20]$, $n = m\lambda$ and $d = 1$. Schwartz [32] gave an algorithm which can be applied to computing $V(1, n, d, \ell_\infty)$. In this thesis, we provide solutions to computing $V(\lambda, n, d, \ell_\infty)$ for $\lambda > 1$ and $d > 1$, which is not contained in their works. We list the values of $V(\lambda, n, d, \ell_\infty)$ for $\lambda > 1$, $m \in [20]$, $n = m\lambda$, $d > 1$ and $d\lambda \leq 10$ in Appendix A.





Chapter 4

Constructions and Related Bounds

In [26], Kløve *et al.* gave several constructions for permutation arrays and they obtained better bounds via observing some properties of the constructions. Here, we start with an explicit construction method which is basically a generalization of one method in [26]. Then, we give several recursive constructions.

4.1 An Explicit Construction

We first give an explicit construction as follows.

Definition 4.1.1. *Given λ , m , and d such that d divides m . We define*

$$C_1(\lambda, m, d) = \{(x_1, \dots, x_{m\lambda}) \in S_{m\lambda}^\lambda : \forall i \in [m\lambda], x_i \equiv i \pmod{d}\}.$$

Theorem 4.1.2. *If $m = ad$, $C_1(\lambda, m, d)$ is a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA of cardinality $\left(\frac{(a\lambda)!}{(\lambda!)^a}\right)^d$.*

Proof. Since $C_1(\lambda, m, d) \subseteq S_{m\lambda}^\lambda$, $C_1(\lambda, m, d)$ has frequency λ and length $m\lambda$. To show the minimum distance, consider two different elements $\mathbf{x}, \mathbf{y} \in C_1(\lambda, m, d)$ and assume their i -th entries are different, i.e., $x_i \neq y_i$. Since $x_i \equiv y_i \pmod{d}$, we have d divides $(x_i - y_i)$ and we have $|x_i - y_i| \geq d$. Hence the minimum distance is at least d .

Now we turn to the cardinality of $C_1(\lambda, m, d)$. For each $j \in [d]$, define a family of index sets

$$X_j = \{(k-1)d + j : k \in [a\lambda]\} = \{j, d+j, \dots, (a\lambda-1)d+j\}.$$

For any $\mathbf{x} \in C_1(\lambda, m, d)$ and any $i \in X_j$, we have $x_i \equiv j \pmod{d}$ and x_i can be any member in $\{j, d+j, \dots, (a-1)d+j\} = \{(k-1)d+j : k \in [a]\}$. Note that each $(k-1)d+j$ appears exactly λ times in \mathbf{x} . Thus for each $j \in [d]$, there are $\frac{(a\lambda)!}{(\lambda!)^a}$ possible combinations for \mathbf{x} at the indices in X_j . Thus, $|C_1(\lambda, m, d)| = \left(\frac{(a\lambda)!}{(\lambda!)^a}\right)^d$ and the theorem holds. \square

Theorem 4.1.2 implies the following lower bound for the cardinality of $(\lambda, n, d, \ell_\infty)$ -FPAs.

Corollary 4.1.3. *If $m = ad$, then $F(\lambda, n, d, \ell_\infty) \geq \left(\frac{(a\lambda)!}{(\lambda!)^a}\right)^d$.*

The information rate of this construction is higher than the lower bound in theorem 3.1.6 under certain parameters. We illustrate some results in Table 4.1.

Table 4.1: Compare C_1 and the bounds in Section 3.1

λ	m	d	Code length (bits)	Information (bits)	Lower bound	Upper bound
10	10	5	306.85438	87.47631	14.76807	140.00000
5	20	5	386.62718	167.24911	110.79509	220.00000
2	50	5	474.76499	255.38692	233.05650	340.19281
1	100	5	524.76499	305.38692	319.55240	436.19281
1	100	10	524.76499	217.91061	226.05207	355.39312
1	100	20	524.76499	138.13781	130.31086	257.59287
1	100	50	524.76499	50.00000	1.41030	150.91463

4.2 Recursive Constructions

We give several recursive constructions for FPAs. These construction could give codes with higher information rate than the construction C_1 in the previous section, however it might require to begin the recursion with better rate than C_1 . The first one is by concatenation. For $\mathbf{x} = (x_1, \dots, x_p)$ and $\mathbf{y} = (y_1, \dots, y_q)$, let $\mathbf{x}|\mathbf{y}$ denote $(x_1, \dots, x_p, y_1, \dots, y_q)$.

Definition 4.2.1. *Given a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA C_λ and a $(\rho, m\rho, d, \ell_\infty)$ -FPA C_ρ . Define*

$$C_\lambda|C_\rho = \{c_\lambda|c_\rho : c_\lambda \in C_\lambda, c_\rho \in C_\rho\}$$

Theorem 4.2.2. *Given a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA C_λ and a $(\rho, m\rho, d, \ell_\infty)$ -FPA C_ρ , $C_\lambda|C_\rho$ is a $(\lambda + \rho, m\lambda + m\rho, d, \ell_\infty)$ -FPA of cardinality $|C_\lambda| \cdot |C_\rho|$.*

Corollary 4.2.3. For integers λ, ρ, m and d ,

$$F(\lambda + \rho, m\lambda + m\rho, d, \ell_\infty) \geq F(\lambda, m\lambda, d, \ell_\infty)F(\rho, m\rho, d, \ell_\infty).$$

The second recursive construction is by interleaving technics.

Definition 4.2.4. Given a $(\lambda, n, d, \ell_\infty)$ -FPA C and a positive integer r . Define

$$C_2(C, r) = \{(r\mathbf{x}^{(1)} - 0 \cdot \mathbf{1})|(r\mathbf{x}^{(2)} - 1 \cdot \mathbf{1})| \cdots |(r\mathbf{x}^{(r)} - (r-1)\mathbf{1}) : \forall i \in [r], \mathbf{x}^{(i)} \in C\}$$

where $\mathbf{1} = \overbrace{(1, \dots, 1)}^n$.

Theorem 4.2.5. Given C and r as above, $C_2(C, r)$ is a $(\lambda, rn, rd, \ell_\infty)$ -FPA of cardinality $|C|^r$.

Corollary 4.2.6. $F(\lambda, rn, rd, \ell_\infty) \geq F(\lambda, n, d, \ell_\infty)^r$.

The rest recursive constructions are by introducing k new symbols. We give three kinds of k -symbol extension constructions. Assume we have $\mathbf{x} = (x_1, \dots, x_{m\lambda}) \in S_{m\lambda}^\lambda$ and $\mathbf{y} = (y_1, \dots, y_{k\lambda}) \in [m+k]^{k\lambda}$ such that for any $i \in [m+k]$, there are at most λ entries in \mathbf{y} equal to i . Our goal is to extend \mathbf{x} into an element in $S_{\lambda(m+k)}^\lambda$. Observe that initially $\mathbf{y}|\mathbf{x}$ may not be a legitimate element in $S_{\lambda(m+k)}^\lambda$. However we can re-assign values to some of \mathbf{x} 's entries such that $\mathbf{y}|\mathbf{x}$ is in $S_{\lambda(m+k)}^\lambda$. To do that we define a total order for the entries in \mathbf{x} , i.e., for $i, j \in [m\lambda]$, we say x_i is larger than x_j if the value of x_i is strictly larger than x_j 's, or when $x_i = x_j$ and $i < j$. Let γ_i be the number of entries in \mathbf{y} equal to i . The extension algorithm $\phi_k(\mathbf{y}, \mathbf{x})$ operates as in Figure 4.1.

$\phi_k(\mathbf{y}, \mathbf{x})$

1. for $i = m+k$ downto 1 do
2. set the largest $\lambda - \gamma_i$ unchanged entries in \mathbf{x} to i and mark them as changed;
3. next i
4. return $\mathbf{y}|\mathbf{x}$;

Figure 4.1: The extension algorithm $\phi_k(\mathbf{y}, \mathbf{x})$

It is easy to check that $\phi_k(\mathbf{y}, \mathbf{x})$ returns a permutation of frequency λ and length $m\lambda + k\lambda$. Since every symbol appears at most λ times in \mathbf{y} , we have $\gamma_j \leq \lambda$ for every $j \in [m+k]$, and

there are $\lambda - \gamma_j + \gamma_j = \lambda$ entries equal to j in $\mathbf{y}|\mathbf{x}$ after the j -th iteration. Moreover, those entries will not be changed afterwards. This shows $\phi_k(\mathbf{y}, \mathbf{x})$ transforms $\mathbf{y}|\mathbf{x}$ into a legitimate element in $S_{\lambda(m+k)}^\lambda$.

For positive integers k, t with $k \leq t$, we consider a *selective function* $f : [k] \rightarrow [t]$, which selects k elements in ascending order from $[t]$, i.e., $f(i)$ is the i -th smallest one among the selected k elements $f(1), \dots, f(k)$. For any $\boldsymbol{\pi} \in S_{k\lambda}^\lambda$ and any t -tuple \mathbf{s} with $1 \leq s_1 < \dots < s_t \leq m+k$, define $\psi_{k,t,\mathbf{s}}(f, \boldsymbol{\pi}) = (s_{f(\pi_1)}, \dots, s_{f(\pi_{k\lambda})})$ and

$$\Psi_{k,t,\mathbf{s}}(C) = \{\psi_{k,t,\mathbf{s}}(f, \boldsymbol{\pi}) : f \text{ is a selective function from } [k] \text{ to } [t] \text{ and } \boldsymbol{\pi} \in C\}$$

for $C \subseteq S_{k\lambda}^\lambda$. We give two distance-preserving constructions by extending frequency permutation \mathbf{x} with $\mathbf{y} \in \Psi_{k,t,\mathbf{s}}(S_{k\lambda}^\lambda)$ and with $\mathbf{y} \in \Psi_{k,t,\mathbf{s}}(C)$ where C is an FPA, respectively.

Definition 4.2.7. Given positive integers k, t with $t \geq k$, a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA C and a t -tuple \mathbf{s} with $1 \leq s_1 < s_2 < \dots < s_t \leq m+k$ and $s_{i+1} - s_i \geq d$ for $i \in [t-1]$. Define

$$C_3(C, k, t, \mathbf{s}) = \{\phi_k(\mathbf{y}, \mathbf{x}) : \mathbf{x} \in C, \mathbf{y} \in \Psi_{k,t,\mathbf{s}}(S_{k\lambda}^\lambda)\}.$$

Definition 4.2.8. Given a positive integer k , a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA C , a $(\lambda, k\lambda, d', \ell_\infty)$ -FPA C' and a k -tuple \mathbf{s} with $1 \leq s_1 < s_2 < \dots < s_k \leq m+k$ and $s_{i+1} - s_i \geq \lceil \frac{d}{d'} \rceil$ for $i \in [k-1]$. Define

$$C_4(C, C', k, \mathbf{s}) = \{\phi_k(\mathbf{y}, \mathbf{x}) : \mathbf{x} \in C, \mathbf{y} \in \Psi_{k,k,\mathbf{s}}(C')\}.$$

Both $C_3(C, k, t, \mathbf{s})$ and $C_4(C, C', k, \mathbf{s})$ are $(\lambda, m\lambda + k\lambda, d, \ell_\infty)$ -FPAs. To prove this, we argue that the distance between any pair of codewords is preserved when they are constructed from distinct \mathbf{y} 's or distinct \mathbf{x} 's. Therefore, we need the following lemma.

Lemma 4.2.9. For two selective functions f and g mapping from $[k]$ to $[t]$ and $\boldsymbol{\pi}, \boldsymbol{\rho} \in S_{k\lambda}^\lambda$, let $\mathbf{y} = \psi_{k,t,\mathbf{s}}(f, \boldsymbol{\pi})$ and $\mathbf{y}' = \psi_{k,t,\mathbf{s}}(g, \boldsymbol{\rho})$. We have $\ell_\infty(\mathbf{y}, \mathbf{y}') \geq d$ when $f \neq g$ or $\boldsymbol{\pi} \neq \boldsymbol{\rho}$. Moreover, $\ell_\infty(\mathbf{y}, \mathbf{y}') \geq d \cdot \ell_\infty(\boldsymbol{\pi}, \boldsymbol{\rho})$ when $f = g$ and $\boldsymbol{\pi} \neq \boldsymbol{\rho}$.

Proof. Let $F = \{f(i) : i \in [k]\}$ and $G = \{g(i) : i \in [k]\}$. If $f \neq g$, there exists i^* such that $i^* \in F - G$. Therefore $f(\pi_j) = i^* \neq g(\rho_j)$ implies $y_j \neq y'_j$. We have $\ell_\infty(\mathbf{y}, \mathbf{y}') \geq |y_j - y'_j| \geq d$.

If $f = g$ and $\pi \neq \rho$, then there exists j^* such that $\pi_{j^*} \neq \rho_{j^*}$. Assume $|\pi_{j^*} - \rho_{j^*}| = d'$, we have $\ell_\infty(\mathbf{y}, \mathbf{y}') \geq |y_{j^*} - y'_{j^*}| \geq |s_{f(\pi_{j^*})} - s_{f(\rho_{j^*})}| \geq d' \min_{i \in [t-1]} (s_{i+1} - s_i) \geq d'd$. \square

Lemma 4.2.10. *If for every $i \in [m+k]$, \mathbf{y} has either λ entries equal to i or no such entry, then for $\mathbf{x}, \mathbf{x}' \in S_{m\lambda}^\lambda$, $\ell_\infty(\phi_k(\mathbf{y}, \mathbf{x}), \phi_k(\mathbf{y}, \mathbf{x}')) \geq \ell_\infty(\mathbf{x}, \mathbf{x}')$.*

Proof. Let γ_i be the number of entries in \mathbf{y} equal to i . By the assumption on \mathbf{y} , we know γ_i is either λ or 0. Thus in every iteration of $\phi_k(\mathbf{y}, \mathbf{z})$ either 0 or λ symbols in \mathbf{z} are changed. Suppose that we run $\phi_k(\mathbf{y}, \mathbf{x})$ and $\phi_k(\mathbf{y}, \mathbf{x}')$ in parallel. According to ϕ_k , there are either λ entries or nothing changed in each iteration. Without loss of generality, let j be the index such that $x_j - x'_j = d$. Note that the algorithm changes only one kind of symbol in each iteration. Therefore x'_j must be changed at least $x_j - x'_j = d$ iterations later after x_j is changed and the magnitude is smaller than x_j 's. This implies $\ell_\infty(\phi_k(\mathbf{y}, \mathbf{x}), \phi_k(\mathbf{y}, \mathbf{x}')) \geq \ell_\infty(\mathbf{x}, \mathbf{x}')$. \square

Now, we prove the following theorems.

Theorem 4.2.11. $C_3(C, k, t, \mathbf{s})$ is a $(\lambda, m\lambda + k\lambda, d, \ell_\infty)$ -FPA of cardinality $\binom{t}{k} \frac{(k\lambda)!}{(\lambda!)^k} |C|$.

Proof. Consider codewords $z = \phi_k(\psi_{k,t,\mathbf{s}}(f, \pi), \mathbf{x})$ and $z' = \phi_k(\psi_{k,t,\mathbf{s}}(g, \rho), \mathbf{x}')$ in $C_3(C, k, t, \mathbf{s})$. If $f \neq g$ or $\pi \neq \rho$, then we have $\ell_\infty(z, z') \geq \ell_\infty(\psi_{k,t,\mathbf{s}}(f, \pi), \psi_{k,t,\mathbf{s}}(g, \rho)) \geq d$ by lemma 4.2.9. If $f = g$ and $\pi = \rho$, then $\ell_\infty(z, z') \geq \ell_\infty(\mathbf{x}, \mathbf{x}') \geq d$ by lemma 4.2.10 and C is a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA. Note that there are $\binom{t}{k}$ selective functions from $[k]$ to $[t]$ and $\frac{(k\lambda)!}{(\lambda!)^k}$ permutations in $S_{k\lambda}^\lambda$, thus the theorem holds. \square

Corollary 4.2.12. $F(\lambda, m\lambda + k\lambda, d, \ell_\infty) \geq \binom{t}{k} \frac{(k\lambda)!}{(\lambda!)^k} F(\lambda, m\lambda, d, \ell_\infty)$ for $k \leq t$ and $td < m + k$.

Theorem 4.2.13. $C_4(C, C', k, \mathbf{s})$ is a $(\lambda, m\lambda + k\lambda, d, \ell_\infty)$ -FPA of cardinality $|C'| \cdot |C|$.

Proof. Consider codewords $z = \phi_k(\psi_{k,k,\mathbf{s}}(f, \pi), \mathbf{x})$ and $z' = \phi_k(\psi_{k,k,\mathbf{s}}(g, \rho), \mathbf{x}')$ in $C_4(C, C', k, \mathbf{s})$. Since there is only one selective function from $[k]$ to $[k]$, we have $f = g$. If $\pi \neq \rho$, then we have $\ell_\infty(z, z') \geq d' \cdot \frac{d}{d'} = d$ by lemma 4.2.9 and C' is a $(\lambda, k\lambda, d', \ell_\infty)$ -FPA. If $\pi = \rho$, then $\ell_\infty(z, z') \geq \ell_\infty(\mathbf{x}, \mathbf{x}') \geq d$ by lemma 4.2.10 and C is a $(\lambda, m\lambda, d, \ell_\infty)$ -FPA. Hence the theorem holds. \square

Corollary 4.2.14. $F(\lambda, m\lambda + k\lambda, d, \ell_\infty) \geq F(\lambda, k\lambda, d', \ell_\infty) \cdot F(\lambda, m\lambda, d, \ell_\infty)$ for $k \cdot \lceil \frac{d}{d'} \rceil < m + k$.

Finally, we provide another symbol extension construction with the distance decreasing property. We relax the constraints on \mathbf{y} by allowing some symbols appearing less than λ times in \mathbf{y} .

Lemma 4.2.15. *If for every $i \in [m+k]$, \mathbf{y} has at most λ entries equal to i , then*

$$\ell_\infty(\phi_k(\mathbf{y}, \mathbf{x}), \phi_k(\mathbf{y}, \mathbf{x}')) \geq \ell_\infty(\mathbf{x}, \mathbf{x}') - 1,$$

for any $\mathbf{x}, \mathbf{x}' \in S_{m\lambda}^\lambda$.

Proof. Consider the j -th entries of \mathbf{x} and \mathbf{x}' , and without loss of generality, let $x_j - x'_j = d > 0$. Assume x_j and x'_j is the α -th and the β -th smallest entry in \mathbf{x} and \mathbf{x}' respectively. α and β must in the form $x_j\lambda - p$ and $x'_j\lambda - q$ for some $p, q \in \{0, \lambda - 1\}$. Since $x_j - x'_j = d$, we have $\alpha - \beta \geq d\lambda - \lambda + 1$.

Suppose that we run $\phi_k(\mathbf{y}, \mathbf{x})$ and $\phi_k(\mathbf{y}, \mathbf{x}')$ in parallel. According to ϕ_k , there are at most λ entries changed in each iteration. The iteration difference between the iterations when x'_j and x_j are respectively changed is at least $\lfloor \frac{\alpha - \beta}{\lambda} \rfloor \geq \lfloor \frac{d\lambda - \lambda + 1}{\lambda} \rfloor = d - 1$. The corresponding entries in the output have difference at least $d - 1$, and thus we conclude $\ell_\infty(\phi_k(\mathbf{y}, \mathbf{x}), \phi_k(\mathbf{y}, \mathbf{x}')) \geq \ell_\infty(\mathbf{x}, \mathbf{x}') - 1$. \square

Lemma 4.2.15 shows that the distance decreases at most 1 after applying $\phi_k(\mathbf{y}, \cdot)$. From this point of view, we can trade minimum distance for larger code size. For integer λ, k, t and t -tuple \mathbf{s} , let $Q(\lambda, k, t, \mathbf{s})$ be the set of vectors of length $k\lambda$ consisting of symbols in $\{s_1, \dots, s_t\}$ such that no symbol appears more than λ times. Note that for any simple set \mathbf{s} , $Q(\lambda, k, t, \mathbf{s})$ has the same cardinality, thus we define $q(\lambda, k, t) = |Q(\lambda, k, t, \mathbf{s})|$.

Definition 4.2.16. *Given positive integers k, t with $t \geq k$, a $(\lambda, m\lambda, d + 1, \ell_\infty)$ -FPA C and a t -tuple \mathbf{s} , $1 \leq s_1 < s_2 < \dots < s_t \leq m + k$ and $s_{i+1} - s_i \geq d$ for $i \in [t - 1]$. Define*

$$C_5(C, k, t, \mathbf{s}) = \{\phi_k(\mathbf{y}, \mathbf{x}) : \mathbf{x} \in C, \mathbf{y} \in Q(\lambda, k, t, \mathbf{s})\}.$$

Theorem 4.2.17. *For $d \geq 1$, $C_5(C, k, t, \mathbf{s})$ is a $(\lambda, m\lambda + k\lambda, d, \ell_\infty)$ -FPA of cardinality $q(\lambda, k, t)|C|$.*

Proof. Consider codewords $\mathbf{z} = \phi_k(\mathbf{y}, \mathbf{x})$ and $\mathbf{z}' = \phi_k(\mathbf{y}', \mathbf{x}')$ where $\mathbf{y}, \mathbf{y}' \in Q(\lambda, k, t, \mathbf{s})$ and \mathbf{x}, \mathbf{x}' in C . We know that $\ell_\infty(\mathbf{z}, \mathbf{z}') \geq d$ when $\mathbf{y} \neq \mathbf{y}'$, since for some $i \in [k\lambda]$, $|y_i - y'_i| \geq \min_{j \in [t-1]} s_{j+1} - s_j \geq d$. By lemma 4.2.15, if $\mathbf{y} = \mathbf{y}'$, then $\ell_\infty(\mathbf{z}, \mathbf{z}') \geq \ell_\infty(\mathbf{x}, \mathbf{x}') \geq d+1-1 = d > 0$. These facts imply $\mathbf{z} \neq \mathbf{z}'$ if $\mathbf{y} \neq \mathbf{y}'$ or $\mathbf{x} \neq \mathbf{x}'$. By the construction of C_5 , it is clear that the cardinality is $q(\lambda, k, t)|C|$. \square

Corollary 4.2.18. $F(\lambda, m\lambda + k\lambda, d, \ell_\infty) \geq q(k, t)F(\lambda, m\lambda, d+1, \ell_\infty)$ for $k \leq t$ and $td < m+k$.

Let $C^{(\lambda, d)} = \{\boldsymbol{\pi} \in S_{(d+1)\lambda}^\lambda : \pi_1, \dots, \pi_\lambda \in \{1, d+1\} \text{ and } \pi_{\lambda+1} \leq \dots \leq \pi_{(d+1)\lambda}\}$. The code constructed by the simple encoding algorithm in the next chapter can be also obtained by the algorithm in Figure 4.2. However, the minimum distance of the output is preserved as d .

1. $C^1 = C^{(\lambda, d)}$;
2. for $i = 2$ to k do
3. $C^i = C_5(C^{i-1}, 1, 2, (1, d+i))$;
4. next
5. Output C^k ;

Figure 4.2: Extension-based construction for code in Chapter 5



Chapter 5

Codes with Efficient Encoding and Decoding

In this chapter, we give a family of FPAs defined by an encoding algorithm. They come with efficient encoding and decoding algorithms. The idea of this construction is based on the previous work[28] by Lin *et al.* We generalize their algorithm for constructing FPAs. In addition, we give the first local decoding algorithm for FPAs under Chebyshev distance. At last, we propose a list decoding algorithm for this code, too.

5.1 Encoding Algorithm

We give an encoding algorithm $\text{ENC}_{n,k}^\lambda$ (see Figure 5.1) which convert k -bit message into a permutation in S_n^λ where $n \geq k + \lambda$.

The encoding algorithm $\text{ENC}_{n,k}^\lambda$ maps binary vectors from Z_2^k to S_n^λ . For examples, the output of $\text{ENC}_{10,4}^2(0, 1, 0, 0)$ is

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

and $\text{ENC}_{10,4}^2(0, 1, 1, 1)$ outputs

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

It is clear that $\text{ENC}_{n,k}^\lambda$ runs in $O(n)$ time while encoding any k -bit message. Next we inves-

Algorithm $\text{Enc}_{n,k}^\lambda$
Input: $\mathbf{y} = (y_1, \dots, y_k) \in Z_2^k$
Output: $\mathbf{x} = (x_1, \dots, x_n) \in S_n^\lambda$

1. $\max \leftarrow n; \min \leftarrow 1;$
2. **for** $i \leftarrow 1$ **to** k **do**
3. **if** $y_i = 1$
4. **then** $\{x_i \leftarrow \lceil \frac{\max}{\lambda} \rceil; \max \leftarrow \max - 1;\}$
5. **else** $\{x_i \leftarrow \lceil \frac{\min}{\lambda} \rceil; \min \leftarrow \min + 1;\}$
6. **for** $i \leftarrow k + 1$ **to** n **do**
7. $x_i \leftarrow \lceil \frac{\min}{\lambda} \rceil; \min \leftarrow \min + 1;$
8. **Output** $\mathbf{x};$

Figure 5.1: $\text{ENC}_{n,k}^\lambda$ encodes messages in Z_2^k with S_n^λ .

tigate the properties of the code obtained by $\text{ENC}_{n,k}^\lambda$. Let $C_{n,k}^\lambda$ be the image of $\text{ENC}_{n,k}^\lambda$.

Theorem 5.1.1. $C_{n,k}^\lambda$ is a $(\lambda, n, \lfloor \frac{n-k}{\lambda} \rfloor, \ell_\infty)$ -FPA with cardinality 2^k .

Proof. Consider two messages $\mathbf{p} = (p_1, \dots, p_k)$ and $\mathbf{p}' = (p'_1, \dots, p'_k) \in Z_2^k$. Let $\mathbf{x} = \text{ENC}_{n,k}^\lambda(\mathbf{p})$ and $\mathbf{x}' = \text{ENC}_{n,k}^\lambda(\mathbf{p}')$ be the outputs of $\text{ENC}_{n,k}^\lambda$ on inputs \mathbf{p} and \mathbf{p}' , respectively. Let r be the smallest index such that $p_r \neq p'_r$. Without loss of generality, we assume $p_r = 1, p'_r = 0$ and there are exactly z zeroes among p_1, \dots, p_{r-1} . Consequently, x_r is set to $\lceil \frac{\max}{\lambda} \rceil = \lceil \frac{n-r+1+z}{\lambda} \rceil$ and x'_r is set to $\lceil \frac{\min}{\lambda} \rceil = \lceil \frac{1+z}{\lambda} \rceil$ by $\text{ENC}_{n,k}^\lambda$. The distance between \mathbf{x} and \mathbf{x}' is:

$$\begin{aligned}
 \ell_\infty(\mathbf{x}, \mathbf{x}') &\geq \left\lceil \frac{n-r+1+z}{\lambda} \right\rceil - \left\lceil \frac{1+z}{\lambda} \right\rceil \\
 &> \frac{n-r+1+z}{\lambda} - \frac{1+z}{\lambda} - 1 \\
 &= \frac{n-r}{\lambda} - 1 \\
 &\geq \frac{n-k}{\lambda} - 1, \text{ since } r \leq k.
 \end{aligned}$$

The first inequality holds by the fact of ceiling function: $\alpha \leq \lceil \alpha \rceil < \alpha + 1$, for any real number α . Note that the distance has integer value only here. If $\frac{n-k}{\lambda}$ is integer then the distance is at least $\lfloor \frac{n-k}{\lambda} \rfloor$; else it is at least $\lceil \frac{n-k}{\lambda} - 1 \rceil$, which is $\lfloor \frac{n-k}{\lambda} \rfloor$ exactly, i.e., the distance between any two codewords in $C_{n,k}^\lambda$ is at least $\lfloor \frac{n-k}{\lambda} \rfloor$. Since every message is encoded into a

distinct codeword, we have $|C_{n,k}^\lambda| = 2^k$. □

Since $C_{n,k}^\lambda$ is a $(\lambda, n, \lfloor \frac{n-k}{\lambda} \rfloor, \ell_\infty)$ -FPA, we let $d = \lfloor \frac{n-k}{\lambda} \rfloor$ in the rest of this chapter for convenience.

5.2 Unique Decoding Algorithm

Unique decoding algorithms for classic error correcting codes are usually much more complicated than their encoding algorithms. While, our proposed decoding algorithm $\text{UNIQUE}_{n,k}^\lambda$ (see Figure 5.2) remains simple.

Algorithm $\text{UNIQUE}_{n,k}^\lambda$
Input: $\mathbf{x} = (x_1, \dots, x_n) \in S_n^\lambda$
Output: $\mathbf{y} = (y_1, \dots, y_k) \in Z_2^k$

1. $max \leftarrow n; min \leftarrow 1;$
2. **for** $i \leftarrow 1$ **to** k **do**
3. **if** $|x_i - \lceil \frac{max}{\lambda} \rceil| < |x_i - \lceil \frac{min}{\lambda} \rceil|$
4. **then** $\{y_i \leftarrow 1; max \leftarrow max - 1;\}$
5. **else** $\{y_i \leftarrow 0; min \leftarrow min + 1;\}$
6. **Output** $\mathbf{y};$

Figure 5.2: $\text{UNIQUE}_{n,k}^\lambda$ decodes words in S_n^λ to messages in Z_2^k .

The running time of $\text{UNIQUE}_{n,k}^\lambda$ is clearly $O(k)$, even faster than the encoding algorithm. We show its correctness as follows.

Theorem 5.2.1. *Given a frequency permutation $\mathbf{x} = (x_1, \dots, x_n)$ which is $\frac{d-1}{2}$ -close to $\text{ENC}_{n,k}^\lambda(\mathbf{y})$ for some $\mathbf{y} \in Z_2^k$, algorithm $\text{UNIQUE}_{n,k}^\lambda$ outputs \mathbf{y} correctly.*

Proof. By contradiction, assume $\text{UNIQUE}_{n,k}^\lambda$ outputs $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_k) \neq \mathbf{y}$. Let r be the smallest index such that $y_r \neq \hat{y}_r$ and z be the number of zeroes in y_1, \dots, y_{r-1} . At the beginning of the r -th iteration, $max = n - r + 1 + z$ and $min = 1 + z$ because for every $i < r$, $y_i = \hat{y}_i$. Without loss of generality, assume $1 = y_r \neq \hat{y}_r = 0$. Let x'_r be the r -th entry of $\text{ENC}_{n,k}^\lambda(\mathbf{y})$. Note that x'_r is set to $\lceil \frac{max}{\lambda} \rceil = \lceil \frac{n-r+1+z}{\lambda} \rceil$ by $\text{ENC}_{n,k}^\lambda$. While \hat{y}_r is decoded to

0 by $\text{UNIQUE}_{n,k}^\lambda$, we have $|x_r - \lceil \frac{max}{\lambda} \rceil| \geq |x_r - \lceil \frac{min}{\lambda} \rceil|$. Thus,

$$\begin{aligned} \ell_\infty(\mathbf{x}, \text{ENC}_{n,k}^\lambda(\mathbf{y})) &\geq |x_r - x'_r| = |x_r - \lceil \frac{max}{\lambda} \rceil| \\ &\geq \frac{1}{2} (|x_r - \lceil \frac{max}{\lambda} \rceil| + |x_r - \lceil \frac{min}{\lambda} \rceil|) \\ &\geq \frac{1}{2} (\lceil \frac{max}{\lambda} \rceil - \lceil \frac{min}{\lambda} \rceil) \\ &= \frac{1}{2} (\lceil \frac{n-r+1+z}{\lambda} \rceil - \lceil \frac{1+z}{\lambda} \rceil) \geq \frac{d}{2}. \end{aligned}$$

The last inequality is true, since we know $\lceil \frac{n-r+1+z}{\lambda} \rceil - \lceil \frac{1+z}{\lambda} \rceil \geq \lfloor \frac{n-k}{\lambda} \rfloor = d$ from the proof of Theorem 5.1.1. This contradicts that \mathbf{x} is $\frac{d-1}{2}$ -close to $\text{ENC}_{n,k}^\lambda(\mathbf{y})$. \square

5.3 Local Decoding Algorithm

A local decoding algorithm computes a specific message bit without reading the whole received string. Here, we show a probabilistic algorithm $\text{LOCAL}_{n,k}^\lambda$, see Figure 5.3, which can perform local decoding. $\text{LOCAL}_{n,k}^\lambda$ allows us to decode the i -th message bit more efficiently than $\text{UNIQUE}_{n,k}^\lambda$, but it may give a wrong result with certain probability. To illustrate this fact, we consider two permutations $\mathbf{x} = (1, 4, 1, 2, 2, 3, 3, 4, 5, 5)$, $\mathbf{y} = (1, 5, 1, 2, 2, 3, 3, 4, 4, 5) = \text{ENC}_{10,4}^2(0, 1, 0, 0)$ and the result of $\text{LOCAL}_{10,4}^2(2, \mathbf{x})$. Since

$$\ell_\infty(\mathbf{x}, \mathbf{y}) = 1 \leq \frac{\lfloor \frac{10-4}{2} \rfloor}{2},$$

$\text{UNIQUE}_{10,4}^2(\mathbf{x})$ outputs $(0, 1, 0, 0)$. $\text{LOCAL}_{10,4}^2(2, \mathbf{x})$ should output 1, but when j is picked as 9 or 10 at line 3, $\text{LOCAL}_{10,4}^2(2, \mathbf{x})$ outputs 0.

We discuss its efficiency and error probability in this section. We prove that it reads at most $\lambda + 1$ entries of the received word in Lemma 5.3.1, hence its running time is $O(\lambda)$. It has a chance to output wrongly, but we show that the error probability is small in Theorem 5.3.2. Furthermore, $\text{LOCAL}_{n,k}^\lambda$ always outputs the correct message bit when it was given a codeword as input, see Corollary 5.3.3.

Lemma 5.3.1. *Given a frequency permutation $\mathbf{x} = (x_1, \dots, x_n) \in S_{n,k}^\lambda$ and an index $i \in [k]$, $\text{LOCAL}_{n,k}^\lambda$ terminates within λ iterations.*

Algorithm $\text{LOCAL}_{n,k}^\lambda$
Input: $i \in [n], (x_1, \dots, x_n) \in S_n^\lambda$
Output: m_i , the i -th message bit
1. $J \leftarrow \{i + 1, \dots, n\}$;
2. **do**
3. Uniformly and randomly pick $j \in J$;
4. **if** $x_i > x_j$ **then** output 1;
5. **if** $x_i < x_j$ **then** output 0;
6. $J \leftarrow J - \{j\}$;
7. **loop**;

Figure 5.3: $\text{LOCAL}_{n,k}^\lambda$ decodes one bit by reading at most $\lambda + 1$ symbols.

Proof. By contradiction, assume $\text{LOCAL}_{n,k}^\lambda$ does not output before the end of the λ -th iteration. For $r \leq \lambda$, let j_r be the index picked in the r -th iteration. For every $r \leq \lambda$, we have $x_i = x_{j_r}$, otherwise $\text{LOCAL}_{n,k}^\lambda$ outputs at the r -th iteration. Therefore, there are at least $\lambda + 1$ entries of \mathbf{x} equal to x_i . It implies $\mathbf{x} \notin S_{n,k}^\lambda$, a contradiction. There is some $x_{j_r} \neq x_i$, and $\text{LOCAL}_{n,k}^\lambda$ outputs in the r -th iteration. \square

Theorem 5.3.2. *Given a permutation $\mathbf{x} = (x_1, \dots, x_n)$ δ -close to a codeword $\text{ENC}_{n,k}^\lambda(\mathbf{y}) = (x'_1, \dots, x'_n) \in S_{n,k}^\lambda$ for some \mathbf{y} and an index $i \in [k]$, $\text{LOCAL}_{n,k}^\lambda$ outputs y_i with probability at least $1 - \frac{2\delta+1}{d}$ at its first iteration.*

Proof. Without loss of generality, we assume $y_i = 0$, $x'_i = t$ and let u be the maximum number among x'_{i+1}, \dots, x'_n , i.e., at the beginning of the i -th iteration $\min = t$ and $\max = u$ while encoding. Assume there are γ numbers equal to t among x'_1, \dots, x'_{i-1} , and there are γ' numbers equal to u among x'_{i+1}, \dots, x'_n . According to the encoding algorithm, we have

$$\{x'_{i+1}, \dots, x'_n\} = \{\overbrace{t, \dots, t}^{\lambda-\gamma-1}, \overbrace{t+1, \dots, t+1}^{\lambda}, \dots, \overbrace{u, \dots, u}^{\gamma'}\}.$$

Since $\ell_\infty(\mathbf{x}, \text{ENC}_{n,k}^\lambda(\mathbf{y})) \leq \delta$, we have $|x_j - x'_j| \leq \delta$ and $|x_i - x'_i| \leq \delta$. The probability that

$\text{LOCAL}_{n,k}^\lambda$ does not output m_i at the first iteration is:

$$\begin{aligned}\Pr[x_i \geq x_j] &\leq \Pr[x'_i + \delta \geq x_j] \\ &\leq \Pr[x'_i + \delta \geq x'_j - \delta] \\ &= \Pr[x'_i + 2\delta \geq x'_j].\end{aligned}$$

There are at most $2\delta\lambda + \lambda - \gamma - 1$ possible choices of x'_j such that x'_j is less than or equal to $x'_i + 2\delta$. Thus,

$$\Pr[x_i \geq x_j] \leq \frac{(2\delta + 1)\lambda - \gamma - 1}{n - i} \leq \frac{2\delta\lambda + \lambda}{d\lambda} = \frac{2\delta + 1}{d}.$$

Therefore, the probability that $\text{LOCAL}_{n,k}^\lambda$ outputs y_i correctly at the first iteration is at least $1 - \frac{2\delta+1}{d}$. \square

Corollary 5.3.3. *Given a codeword $\mathbf{x} = \text{ENC}_{n,k}^\lambda(\mathbf{y})$ for some message $\mathbf{y} \in \{0,1\}^k$ and an index $i \in [k]$, $\text{LOCAL}_{n,k}^\lambda$ outputs y_i correctly.*

Proof. By Lemma 5.3.1, there exists $r \leq \lambda$ such that $\text{LOCAL}_{n,k}^\lambda$ terminates at the r -th iteration. Let j be the index picked at the r -th iteration, we have $x_j \neq x_i$, where $j > i$. Note that \mathbf{x} is a codeword, i.e., $x_i < x_j$ implies $y_i = 0$ and $x_i > x_j$ implies $y_i = 1$. Hence, $\text{LOCAL}_{n,k}^\lambda$ outputs y_i correctly. \square

5.4 List Decoding Algorithm

In this section, we define list decoding algorithms first.

Definition 5.4.1. *On input \mathbf{x} , an (ϵ, L) -list decoding algorithm always outputs a list of size at most L which contains all ϵ -close codewords to \mathbf{x} .*

We say a code C is (ϵ, L) -list-decodable if there is an (ϵ, L) -list decoding algorithm for C . We propose a simple $(\lfloor \frac{m-k'-1}{2} \rfloor, 2^{k-k'\lambda})$ -list decoding algorithm $\text{LIST}_{n,k,k'}^\lambda$ for $C_{n,k}^\lambda$ where $k'\lambda \leq k$, see Figure 5.4. $\text{LIST}_{n,k,k'}^\lambda$ is based on $\text{UNIQUE}_{n,k'\lambda}^\lambda$ which decodes the first $k'\lambda$ message bits \mathbf{y} from \mathbf{x} . Then, $\text{LIST}_{n,k,k'}^\lambda$ outputs all codewords encoded from messages starting with \mathbf{y} .

Theorem 5.4.2. *$\text{LIST}_{n,k,k'}^\lambda$ is a $(\lfloor \frac{m-k'-1}{2} \rfloor, 2^{k-k'\lambda})$ -list decoding algorithm for $C_{n,k}^\lambda$.*

Algorithm List $_{n,k,k'}^\lambda$

Input: $\mathbf{x} = (x_1, \dots, x_n) \in S_n^\lambda$

Output: $X \subseteq S_n^\lambda$

1. $\mathbf{y} \leftarrow \text{UNIQUE}_{n,k'\lambda}^\lambda(\mathbf{x})$; // Note that $\mathbf{y} \in Z_2^{k'\lambda}$
2. $X \leftarrow \{\text{ENC}_{n,k}^\lambda(\mathbf{y}') : \mathbf{y}' \in Z_2^k \text{ and } \forall i \in [k'\lambda], y'_i = y_i\}$;
3. Output X ;

Figure 5.4: LIST $_{n,k}^\lambda$ gives a list of candidates of the closest codewords.

Proof. Let \mathbf{x} be the input. The number of k -tuples in Z_2^k , such that their first $k'\lambda$ bits are $y_1, \dots, y_{k'\lambda}$, is at most $2^{k-k'\lambda}$, since the first $k'\lambda$ entries are fixed. The rest is to prove that all codewords $\lfloor \frac{m-k'-1}{2} \rfloor$ -close to \mathbf{x} are in the output X . Contrarily, assume there is an $\mathbf{x}^* = \text{ENC}_{n,k}^\lambda(\mathbf{y}^*) \notin X$ such that $\ell_\infty(\mathbf{x}, \mathbf{x}^*) \leq \lfloor \frac{m-k'-1}{2} \rfloor$. Hence, there exists $r \in [k'\lambda]$ such that $y_r^* \neq y_r$. Consider $\mathbf{x}' = \text{ENC}_{n,k'\lambda}^\lambda(y_1, \dots, y_{k'\lambda})$. It is easy to verify that $\mathbf{x}' = \text{ENC}_{n,k}^\lambda(\mathbf{y}')$ where $\mathbf{y}' = (y_1, \dots, y_{k'\lambda}, \overbrace{0, \dots, 0}^{k-k'\lambda})$. Assume there are exactly z zeroes among y_1, \dots, y_{r-1} , and let $r^* = \min_{y_r^* \neq y_r} r$. By an argument similar to the one in the proof of 5.1.1, we have

$$|x_{r^*}^* - x_{r^*}'| = \left\lceil \frac{n - r^* + 1 + z}{\lambda} \right\rceil - \left\lceil \frac{1 + z}{\lambda} \right\rceil > \frac{n - k'\lambda}{\lambda} - 1 = m - k' - 1.$$

However, \mathbf{x}^* is $\lfloor \frac{m-k'-1}{2} \rfloor$ -close to \mathbf{x} . This implies

$$\begin{aligned} |x_{r^*}' - x_{r^*}^*| &\geq |x_{r^*}^* - x_{r^*}'| - |x_{r^*}^* - x_{r^*}^*| \\ &> m - k' - 1 - \left\lfloor \frac{m - k' - 1}{2} \right\rfloor \\ &\geq \left\lfloor \frac{m - k' - 1}{2} \right\rfloor \\ &\geq \ell_\infty(\mathbf{x}, \mathbf{x}^*) \\ &\geq |x_{r^*}^* - x_{r^*}^*|. \end{aligned}$$

In other words, x_{r^*} is closer to $x_{r^*}^*$ than x_{r^*}' . Note that $\mathbf{x}' = \text{ENC}_{n,k'\lambda}^\lambda(y_1, \dots, y_{k'\lambda})$. Thus, $\text{UNIQUE}_{n,k'\lambda}^\lambda(\mathbf{x})$ should choose $y_{r^*}^*$ as the r^* -th bit of its output. We have $y_{r^*}^* = y_{r^*}$, a contradiction. \square

The running time of LIST $_{n,k,k'}^\lambda(\mathbf{x})$ is $\mathcal{O}(n2^{k-k'\lambda})$, since we need encode all k -tuples starting

with $\text{UNIQUE}_{n,k'\lambda}^\lambda(\mathbf{x})$. When $k - k'\lambda = \mathcal{O}(\log n)$, $\text{LIST}_{n,k,k'}^\lambda(\mathbf{x})$ is a polynomial-time algorithm.

5.5 Private Information Retrieval

Consider an investment company stores some data of future contracts on remote storage service providers. When the company retrieves their data, the service providers know which data are sent. However, the company does not want to leak the information about the queried contracts, since its profit may be affected by the disclosure. A *private information retrieval* (PIR) scheme allows users to query data from the storage servers without disclosing all information about which data is retrieved. It has been shown that PIR can be constructed from locally decodable codes, see Trevisan's survey [38]. In this section, we give a PIR based on our local decoding algorithm $\text{LOCAL}_{n,k}^\lambda$.

A PIR consists of q independent servers s_1, \dots, s_q , which cannot exchange information with the others. All servers know a codeword $\mathbf{x} = (x_1, \dots, x_n)$ representing a k -bit message $\mathbf{y} = (y_1, \dots, y_k)$, and a user wants to know one bit y_i of \mathbf{y} via query a symbol from each server. We say a PIR protocol has *recovery* r if the user can obtain the message bit with probability r . Let $\mathcal{P}_{s,i}(j)$ be the probability that the x_j is queried from server s when the user tries to retrieve y_i . A PIR has *privacy* p if

$$p = \max_{s \in \{s_1, \dots, s_q\}, i, i' \in [k]} \sum_{j \in [n]} \frac{1}{2} |\mathcal{P}_{s,i}(j) - \mathcal{P}_{s,i'}(j)|.$$

A (q, r, p) -PIR is a q -server PIR with recovery r and privacy p . A (q, r, p) -PIR has perfect recovery if $r = 1$ and perfect privacy if $p = 0$. With our FPA $C_{n,k}^\lambda$ in Chapter 5, we construct a PIR with perfect recovery. For a message \mathbf{y} , we put $\mathbf{x} = \text{ENC}_{n,k}^\lambda(\mathbf{y})$ on all $\lambda + 1$ servers. Assume the user wants to retrieve y_i . The scheme is simple:

1. Generate a permutation $\pi \in S_{\lambda+1}$ uniformly at random.
2. Sample λ random distinct indices i_1, \dots, i_λ uniformly from $[i + 1, n]$.
3. Query $x_i, x_{i_1}, \dots, x_{i_\lambda}$ from $s_{\pi(1)}, s_{\pi(2)}, \dots, s_{\pi(\lambda+1)}$, respectively.
4. If $x_{i_j} > x_i$ for some $j \in [\lambda]$, then $y_i = 0$, otherwise $y_i = 1$.

Theorem 5.5.1. *Our PIR scheme is a $(\lambda + 1, 1, \frac{1}{\lambda+1} - \frac{2\lambda}{(\lambda+1)(n-1)})$ -PIR.*

Proof. The perfect recovery is guaranteed by the proof of Corollary 5.3.3, since there are exactly λ copies of each symbol in $[m]$. The probability distributions $\mathcal{P}_{s_1,i}, \dots, \mathcal{P}_{s_q,i}$ are identical for a fixed i , since π is chosen uniformly at random. Consider when we retrieve y_i . We do not query x_j for $j < i$, so $\mathcal{P}_{s_1,i}(j) = 0$ for $j < i$. We have to read x_i , hence $\mathcal{P}_{s_1,i}(i) = \frac{1}{\lambda+1}$. We query λ entries among x_{i+1}, \dots, x_n , hence $\mathcal{P}_{s_1,i}(j) = \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i}$ for $j > i$. Therefore, we have the privacy

$$\begin{aligned}
p &= \max_{s \in \{s_1, \dots, s_q\}, i, i' \in [k]} \sum_{j \in [n]} \frac{1}{2} |\mathcal{P}_{s,i}(j) - \mathcal{P}_{s,i'}(j)| \\
&= \max_{i, i' \in [k]} \sum_{j \in [n]} \frac{1}{2} |\mathcal{P}_{s_1,i}(j) - \mathcal{P}_{s_1,i'}(j)| \\
&= \frac{1}{2} \max_{i, i' \in [k], i < i'} \left(\left| \frac{1}{\lambda+1} - 0 \right| + \sum_{j=i+1}^{i'-1} \left| \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i} - 0 \right| \right. \\
&\quad \left. + \left| \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i} - \frac{1}{\lambda+1} \right| + \sum_{j=i'+1}^n \left| \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i} - \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i'} \right| \right) \\
&= \frac{1}{2} \max_{i, i' \in [k], i \leq i'} \left(\frac{1}{\lambda+1} + \left(\frac{1}{\lambda+1} - \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i} \right) + \sum_{j=i+1}^{i'-1} \frac{\lambda}{\lambda+1} \cdot \frac{1}{n-i} + \sum_{j=i'+1}^n \frac{\lambda}{\lambda+1} \cdot \left(\frac{1}{n-i'} - \frac{1}{n-i} \right) \right) \\
&= \frac{1}{\lambda+1} \left(1 + \frac{1}{2} \max_{i, i' \in [k], i \leq i'} \left(\frac{(n-i')\lambda}{n-i'} - \frac{(n-i'+1 - (i'-1-i))\lambda}{n-i} \right) \right) \\
&= \frac{1}{\lambda+1} \left(1 + \frac{1}{2} \max_{i \in [k]} \left(\lambda - \frac{(n+i+2)\lambda}{n-i} \right) \right) \\
&= \frac{1}{\lambda+1} \left(1 + \max_{i \in [k]} \left(-\frac{(i+1)\lambda}{n-i} \right) \right) \\
&= \frac{1}{\lambda+1} \left(1 - \frac{2\lambda}{n-1} \right) = \frac{1}{\lambda+1} - \frac{2\lambda}{(\lambda+1)(n-1)}.
\end{aligned}$$

For $i, i' \in [k]$ and $j \in [n]$, we have that $|\mathcal{P}_{s_1,i}(j) - \mathcal{P}_{s_1,i'}(j)| = |\mathcal{P}_{s_1,i'}(j) - \mathcal{P}_{s_1,i}(j)|$ and $|\mathcal{P}_{s_1,i}(j) - \mathcal{P}_{s_1,i'}(j)| \geq 0 = |\mathcal{P}_{s_1,i}(j) - \mathcal{P}_{s_1,i}(j)|$, so there is a maximum achieved by a selection $i < i'$. Therefore, the third equality holds. We obtain the forth equality by removing the absolute-value bars and reordering terms. The rest part involves only simple calculations. \square

The communication complexity of a PIR scheme is the number of bits exchanged between the user and the servers. Yekhanin[45] proposed a family of locally decodable codes and a

$(3, 1, 0)$ -PIR with perfect privacy and perfect recovery. The communication complexity of Yekhanin's $(3, 1, 0)$ -PIR is $\mathcal{O}\left(n^{10^{-7}}\right)$, and it can be lowered to $\mathcal{O}\left(n^{\frac{1}{\log \log n}}\right)$ if there are infinitely many Mersenne primes. Although our PIR has imperfect privacy, the communication complexity of our PIR scheme has $\mathcal{O}(\lambda \log n)$, which is much lower than Yekhanin's result when λ is a constant.



Chapter 6

Complexity Issues

6.1 Complexity Problems Related to FPAs

The minimum distance of an FPA plays a main role in many applications. In Chapter 4, we give several construction methods for generating FPAs of certain minimum distance. Here, we study the problem of another direction. We investigate the complexity of computing the minimum distance for an arbitrary FPA. In Chapter 5, we propose efficient decoders for a family of FPAs. Conversely, we analyze the complexity of decoding a general FPA in this chapter.

In Chapter 2, we say an algorithm is time efficient if and only if it runs in polynomial time with respect to its input size. When we describe an FPA $C \subseteq S_n^\lambda$ by writing down all codewords in C , we need $\Theta(n|C|)$ symbols in $[m]$ to represent $|C|$. This allows us to compute the minimum distance of $|C|$ under Hamming distance, Minkowski distance and Chebyshev distance efficiently by enumerating all possible pairs of distinct codewords in C , since computing the distance between two n -tuples is in $\mathcal{O}(n)$ and $n \binom{|C|}{2} = \mathcal{O}(n|C|^2) = \mathcal{O}(\text{poly}(n|C|))$. Also, for the decoding problem, “Given an n -tuple \mathbf{x} in $[m]^n$ and an FPA $C \subseteq S_n^\lambda$, find the codeword closest in C under metric δ ”, can be done efficiently in $\mathcal{O}(n|C|)$ when δ is Hamming distance, Minkowski distance or Chebyshev distance by evaluating the distance between every codeword in C and \mathbf{x} .

However, some families of FPAs have much shorter descriptions than general FPAs. One of those families is the *subgroup permutation arrays* (SPAs). A SPA permutation array of

length n is a subgroup of S_n . We can define a SPA by giving its generator set $\{g_1, \dots, g_k\}$. We only need kn symbols in $[n]$ to denote a generator set of k generators, but the cardinality of the corresponding subgroup can be an exponential of kn . For example, we can represent every permutation in S_n by a sequence of swapping adjacent pairs, i.e., $\{(\overline{i, i+1}) : i \in [n-1]\}$ generates S_n which has cardinality $n!$, and it just has input size $\Theta(n^2)$. Now, we define the shortest distance in subgroup permutation array problem (SDSPA_δ) and the closest distance in subgroup permutation array problem (CDSPA_δ).

Definition 6.1.1. (SDSPA_δ) Given a generator set $\{g_1, \dots, g_k\}$ for a subgroup G of S_n and a positive integer B , determine if there exists distinct permutations $\mathbf{x}, \mathbf{y} \in H$ such that $\delta(\mathbf{x}, \mathbf{y}) \leq B$.

Definition 6.1.2. (CDSPA_δ) Given a generator set $\{g_1, \dots, g_k\}$ for a subgroup G of S_n , a permutation $\mathbf{y} \in S_n$ and a positive integer B , determine if there exists a permutation $\mathbf{x} \in H$ such that $\delta(\mathbf{x}, \mathbf{y}) \leq B$.

CDSPA_δ is the decision problem of maximum likelihood decoding problem. Buchheim *et al.* proved that CDSPA_δ , which is called *minimum subgroup distance problem* in their work, is NP-hard for Hamming distance, Chebyshev distance, Cayley distance, Minkowski distance of order $p \in \mathbb{N}$, Lee's distance, Kendall's tau, and Ulam's distance, see Theorem 2, Theorem 3, and Theorem 4 in [6].

Recall that the minimum distance of $G \subseteq S_n$ under a right invariant metric is equal to the minimum weight of non-identity permutation in G . To prove SDSPA_δ is NP-hard under right invariant metric δ , we give a reduction from an NP-hard problem to the problem of computing the minimum weight of non-identity permutation in G under metric δ . Now, we define the minimum weight problem of subgroup permutation code under a right-invariant metric δ , and we call it MINWSPA_δ for short.

Definition 6.1.3. (MINWSPA_δ) Given a generator set $\{g_1, \dots, g_k\}$ for a subgroup G of S_n and a positive integer B , determine if there exists a permutation $\mathbf{x} \in H$ such that $0 < wt_\delta(\mathbf{x}) \leq B$.

6.2 Minimum Distance of Subgroup Codes

Cameron and Wu [7] gave reductions from the minimum weight problems for binary linear block codes to MINWSPA_δ under various metrics such as Hamming distance, Cayley distance, movement distance, Minkowski distance of order $p \in \mathbb{N}$, Kendall's tau distance, Lee and Ulam metrics (see Theorem 5 in [7]). Since Vardy [40] showed that the minimum weight problems for binary linear block codes is NP-hard, we have that MINWSPA_δ under every metric previously mentioned is NP-hard, too. Moreover, SDSPA_δ is also NP-hard if δ is right-invariant and in the above list.

NAESAT is a classical NP-complete problem (e.g., see Papadimitriou's book, page 187 [30]). Cameron and Wu [7] gave a reduction from NAESAT to $\text{MINWSPA}_{\ell_\infty}$ (see Theorem 18). But their construction actually failed, which is shown in Section 6.3. In the rest of this section, we give a reduction from Not-All-Equal-SAT (NAESAT) to $\text{MINWSPA}_{\ell_\infty}$. We give the formal definition of Not-All-Equal-SAT problem as follows.

Definition 6.2.1. (*NAESAT*) *Given a boolean formula ϕ in conjunctive normal form, which consists of m exact-3-literal clauses c_1, \dots, c_m over n variables x_1, \dots, x_n , decide whether there exists an assignment σ such that for every clause c , not all literals in c are assigned to the same truth value.*

If a formula ϕ has such an assignment σ , then ϕ is satisfiable, and we say σ is a satisfying assignment. Otherwise, ϕ is unsatisfiable.

We will first sketch the idea of our reduction. For every positive integer α , we give a mapping function f_α from NAESAT instances into subgroup permutation codes. More specifically, $f_\alpha(\phi)$ is a subgroup permutation code over $[N]$ for a n -variable- m -clause formula ϕ where $N = (6m + 2n)(2\alpha + 6)$. Satisfiable formulas are mapped into codes of minimum weight $\alpha + 5$, and unsatisfiable ones are mapped into codes of minimum weight $2\alpha + 5$, see Figure 6.1.

In the rest of this section, we use f, \mathbf{e} and wt to denote f_α, \mathbf{e}_N and wt_{ℓ_∞} for short, respectively. For an n -variable- m -clause E3CNF-formula (exact-3-literal-conjunctive-normal-

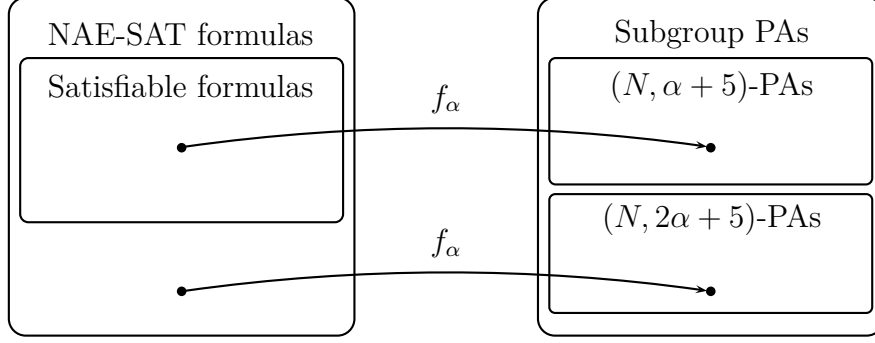


Figure 6.1: The sketch of reduction

form-formula) ϕ , we map ϕ into a subgroup $f(\phi) \subseteq S_N$ where $f(\phi)$ has $2n$ generators

$$g_1(\phi), \dots, g_n(\phi), g'_1(\phi), \dots, g'_n(\phi).$$

The generators are designed to be self-inverse and commutative. I.e.,

- for every generator x , $xx = e$,
- for every pair of generators x and y , $xy = yx$.

Therefore, every permutation $\pi \in f(\phi)$ can be expressed in a normal form as

$$\pi = \left(\prod_{i \in [n]} g_i(\phi)^{z_i(\pi)} \right) \left(\prod_{i \in [n]} g'_i(\phi)^{z'_i(\pi)} \right)$$

where $z_i(\pi), z'_i(\pi) \in \{0, 1\}$ for $i \in [n]$. With this observation, we divide $f(\phi)$ into four categories.

1. $\mathcal{C}_0(\phi) = \{\pi : \forall i \in [n], z_i(\pi) = z'_i(\pi) = 0\} = \{e\}$.
2. $\mathcal{C}_1(\phi) = \{\pi : \forall i \in [n], z_i(\pi) + z'_i(\pi) = 1\}$.
3. $\mathcal{C}_2(\phi) = \{\pi : \exists i \in [n], z_i(\pi) + z'_i(\pi) = 2\}$.
4. $\mathcal{C}_3(\phi)$: permutations not in the above three categories.

There is a bijective mapping μ from the set of truth assignments on the variables x_1, \dots, x_n of ϕ to $\mathcal{C}_1(\phi)$. An assignment σ sets x_i to be true if and only if $z_i(\pi) = 1$ where $\pi = \mu(\sigma)$.

For example, permutation $\pi = g_1(\phi)g'_2(\phi)g'_3(\phi) \in f(\phi)$ is mapped from the assignment σ with $\sigma(x_1) = T, \sigma(x_2) = F, \sigma(x_3) = F$ for some trivariate formula ϕ . We partition $\mathcal{C}_1(\phi)$ into two sets $\mathcal{C}_1^s(\phi)$ and $\mathcal{C}_1^u(\phi)$ where $\mathcal{C}_1^s(\phi)$ is exactly the image of the set of all assignments satisfying ϕ and $\mathcal{C}_1^u(\phi) = \mathcal{C}_1(\phi) \setminus \mathcal{C}_1^s(\phi)$. In other words, $\mathcal{C}_1^s(\phi) = \{\mu(\sigma) : \sigma \text{ satisfies } \phi\}$ and $\mathcal{C}_1^u(\phi) = \{\mu(\sigma) : \sigma \text{ does not satisfy } \phi\}$. We consider $\mathcal{C}_1^s(\phi)$ and $\mathcal{C}_1^u(\phi)$ to be different categories in the following.

The main goal of our reduction is to ensure that the permutations in the same category have the same weight. Moreover, the weight of permutations in $\mathcal{C}_1^s(\phi)$ must be different from the weight of permutations in the other categories. We list the weights of all categories in our reduction as follows.

- $wt(\pi) = 0$ for $\pi = e$.
- $wt(\pi) = \alpha + 5$ for $\pi \in \mathcal{C}_1^s(\phi)$.
- $wt(\pi) = 2\alpha + 5$ for $\pi \in \mathcal{C}_1^u(\phi) \cup \mathcal{C}_2(\phi) \cup \mathcal{C}_3(\phi)$.

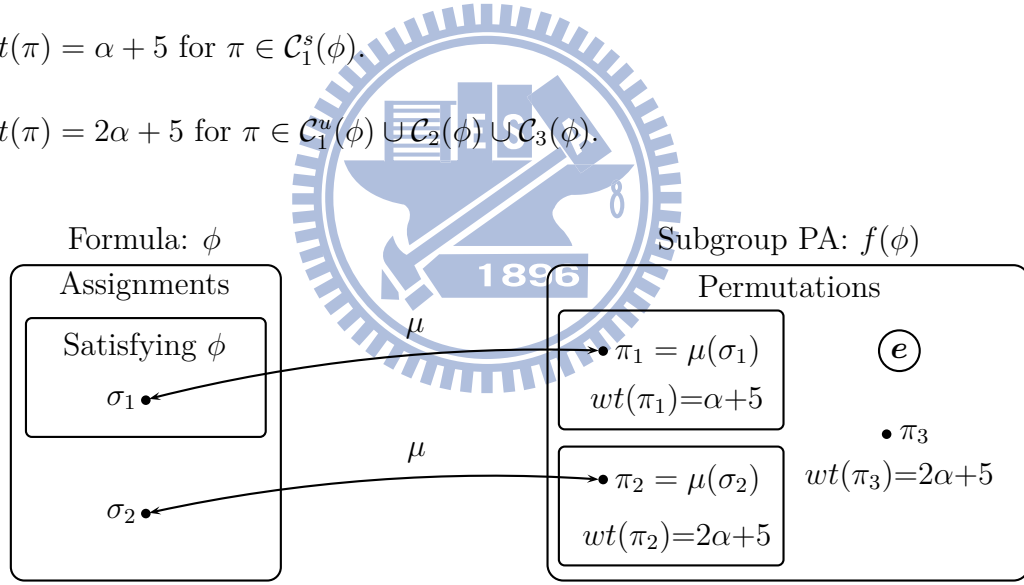


Figure 6.2: Satisfiable ϕ is mapped into $(N, \alpha + 5)$ -PA $f(\phi)$.

A satisfiable formula ϕ is mapped to $f(\phi)$ of minimum weight $\alpha + 5$, since $\mathcal{C}_1^s(\phi)$ is non-empty, see Figure 6.2. On the other hand, there does not exist an assignment satisfying any unsatisfiable ϕ' , so $\mathcal{C}_1^s(\phi')$ is empty for ϕ' . Hence, $f(\phi')$ has minimum weight $2\alpha + 5$, see Figure 6.3. This immediately implies Theorem 6.2.2.

Theorem 6.2.2. *Given an NAESAT instance ϕ , if ϕ is satisfiable then $f(\phi)$ has minimum weight $\alpha + 5$, otherwise $f(\phi)$ has minimum weight $2\alpha + 5$.*

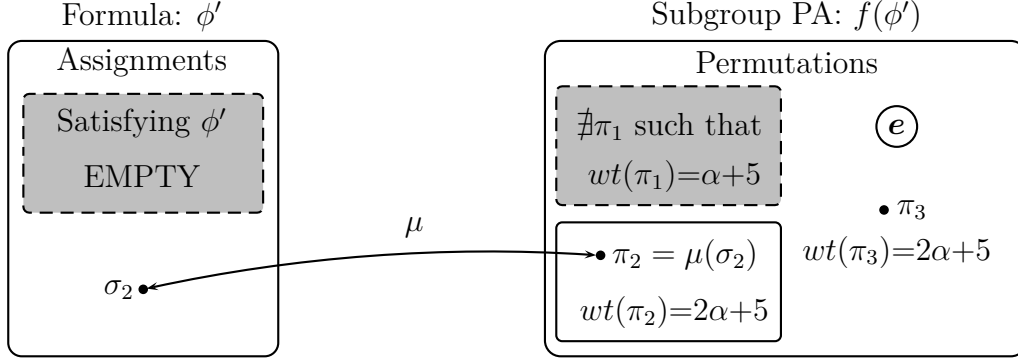


Figure 6.3: Unsatisfiable ϕ' is mapped into $(N, 2\alpha + 5)$ -PA $f(\phi')$.

Recall that we set $N = (6m + 2n)(2\alpha + 6)$. For convenience, we define $\beta = 2\alpha + 6$, and we have $N = 6m\beta + 2n\beta$. To construct the corresponding generator set from an NAESAT instance ϕ , we define three kinds of permutation gadgets for clauses, variables and the truth assignment over $[N]$, respectively. There is no intersection between any two different kinds of gadgets, i.e., every position can only be permuted exactly by one kind of gadgets.

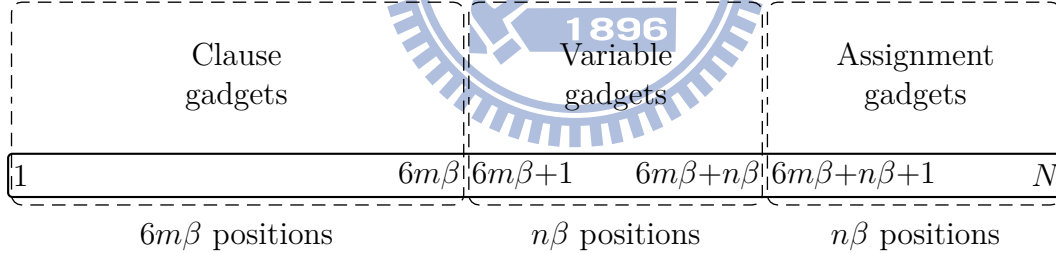


Figure 6.4: No position is permuted by 2 kinds of gadgets.

Every generator of $f(\phi)$ is the product of a clause gadget, a variable gadget and an assignment gadget. Define

$$g_i(\phi) = g_{i|c}(\phi)g_{i|v}(\phi)g_{i|a}(\phi),$$

where $g_{i|c}(\phi)$, $g_{i|v}(\phi)$ and $g_{i|a}(\phi)$ are the clause gadget, the variable gadget and the assignment gadget of $g_i(\phi)$, respectively. Similarly, we define

$$g'_i(\phi) = g'_{i|c}(\phi)g'_{i|v}(\phi)g'_{i|a}(\phi).$$

For $\pi \in f(\phi)$, we define

$$\pi_c = \left(\prod_{i \in [n]} g_{i|c}(\phi)^{z_i(\pi)} \right) \left(\prod_{i \in [n]} g'_{i|c}(\phi)^{z'_i(\pi)} \right),$$

$$\pi_v = \left(\prod_{i \in [n]} g_{i|v}(\phi)^{z_i(\pi)} \right) \left(\prod_{i \in [n]} g'_{i|v}(\phi)^{z'_i(\pi)} \right),$$

and

$$\pi_a = \left(\prod_{i \in [n]} g_{i|a}(\phi)^{z_i(\pi)} \right) \left(\prod_{i \in [n]} g'_{i|a}(\phi)^{z'_i(\pi)} \right).$$

Since the positions permuted by π_c , π_v , and π_a are all disjoint, we have $\pi = \pi_c \pi_v \pi_a$ and $wt(\pi) = \max(wt(\pi_c), wt(\pi_v), wt(\pi_a))$. In the rest of this section, we show how to construct the gadgets such that the weight distribution of $f(\phi)$ is as in table 6.1.

Table 6.1: Weight contribution of different gadgets

	$wt(\pi_c)$	$wt(\pi_v)$	$wt(\pi_a)$	$wt(\pi)$
$\pi = \mathbf{e}$	0	0	0	0
$\pi \in \mathcal{C}_1^s(\phi)$	$\alpha + 5$	$\leq \alpha + 5$	0	$\alpha + 5$
$\pi \in \mathcal{C}_1^u(\phi)$	$2\alpha + 5$	$\leq \alpha + 5$	0	$2\alpha + 5$
$\pi \in \mathcal{C}_2(\phi)$	$\leq 2\alpha + 5$	$2\alpha + 5$	$\leq 2\alpha + 5$	$2\alpha + 5$
$\pi \in \mathcal{C}_3(\phi)$	$\leq 2\alpha + 5$	$\leq \alpha + 5$	$2\alpha + 5$	$2\alpha + 5$

Before we go through the detail of our reduction, we introduce several tools. Klein four-group is the building block of our proofs. It is defined as $K_4 = \{\mathbf{e}, \kappa_1, \kappa_2, \kappa_3\}$, where $\kappa_1 = \overline{(1,2)}\overline{(3,4)}$, $\kappa_2 = \overline{(1,3)}\overline{(2,4)}$, and $\kappa_3 = \overline{(1,4)}\overline{(2,3)}$. Its operation is shown in Table 6.2. It is clear that K_4 is commutative and $wt(\kappa_u) = u$ for $u \in [3]$. Note that for $u \in [3]$, $\kappa_u \kappa_u = \mathbf{e}$, and for distinct $u, v, w \in \{1, 2, 3\}$, $\kappa_u = \kappa_v \kappa_w$ and $\kappa_u \kappa_v \kappa_w = \mathbf{e}$.

Then, we define the stretch operation a_t on a permutation π where $\max\{i : \pi(i) \neq i\} \leq \frac{N}{t}$ as $a_t(\pi) = \rho_t \pi \rho_t^{-1}$ where $\rho_t \in S_N$ is a permutation such that $\rho_t(x) = tx$ for $t \in [\lceil \frac{n}{t} \rceil]$. For example,

$$a_3 \left(\overline{(1,2)}\overline{(3,4)} \right) = \overline{(\rho_3(1), \rho_3(2))} \overline{(\rho_3(3), \rho_3(4))} = \overline{(3,6)}\overline{(9,12)}$$

when $N \geq 4 \cdot 3 = 12$. The stretch operations do not change the cycle structure, but it

Table 6.2: Operation of Klein four-group.

\circ	\mathbf{e}	κ_1	κ_2	κ_3
\mathbf{e}	\mathbf{e}	κ_1	κ_2	κ_3
κ_1	κ_1	\mathbf{e}	κ_3	κ_2
κ_2	κ_2	κ_3	\mathbf{e}	κ_1
κ_3	κ_3	κ_2	κ_1	\mathbf{e}

amplifies the weight to t times.

The *symbol shift* operation $s_b(\cdot)$ on a permutation π as $s_b(\pi) = \tau_b \pi \tau_b^{-1}$ where $\tau_b \in S_N$ and $\tau_b(x) \equiv x + b \pmod{N}$. It is a special relabeling operation. For example

$$s_2\left(\overline{(1,2)(3,4)}\right) = \overline{(\tau_2(1), \tau_2(2))(\tau_2(3), \tau_2(4))} = \overline{(3,4)(5,6)}$$

when $N > 6$. For two permutations

$$x = \overline{(x_1, x_2, \dots, x_p)(y_1)(y_2) \cdots (y_q)}$$

and

$$y = \overline{(x_1)(x_2) \cdots (x_p)(y_1, y_2, \dots, y_q)},$$

we have

$$xy = \overline{(x_1, x_2, \dots, x_p)(y_1, y_2, \dots, y_q)}$$

when $\{x_1, x_2, \dots, x_p\}$ and $\{y_1, y_2, \dots, y_q\}$ are disjoint. Hence, we have

$$wt(xy) = \max\{wt(x), wt(y)\}$$

under such configuration. With this observation, we apply a sequence of stretch operations, symbol shift operations, relabeling operations and compositions on elements in K_4 and swapping pairs (permutations in the form $\overline{(p, q)}$ for $p, q \in [N]$) to construct more complex gadgets of higher weights while preserving some cycle structures.

The clause gadgets permute $1, \dots, 6m\beta$, which are derived from the work by Cameron

and Wu[7]. The main idea of the clause gadget is to assure that all literals are not assigned to the same value. For convenience, we express the following permutations, which are the basic construction blocks, with the shift and stretch operations over elements of Klein four-group and swapping pairs. Let $\kappa'_i = a_2(\kappa_i)s_{-1}(a_2(\kappa_i))$ for $i \in [3]$ and $\chi = \overline{(1,2)}\overline{(3,4)}\overline{(5,6)}\overline{(7,8)}$. We list κ'_1, κ'_2 and κ'_3 explicitly as follows:

$$\begin{aligned}\kappa'_1 &= \overline{(1,3)}\overline{(5,7)}\overline{(2,4)}\overline{(6,8)}, \\ \kappa'_2 &= \overline{(1,5)}\overline{(3,7)}\overline{(2,6)}\overline{(4,8)}, \\ \kappa'_3 &= \overline{(1,7)}\overline{(3,5)}\overline{(2,8)}\overline{(4,6)}.\end{aligned}$$

These basic blocks inherit some good properties from Klein four-group.

Proposition 6.2.3. $\kappa'_1, \kappa'_2, \kappa'_3$ and χ are self-inverse and commutative. Moreover, $\kappa'_u = \kappa'_v \kappa'_w$ for $\{u, v, w\} = \{1, 2, 3\}$.

Proof. $\kappa'_1, \kappa'_2, \kappa'_3$ and χ are self-inverse, because they only swap pairs of positions. Since κ'_1, κ'_2 and κ'_3 only swap positions of the same parity, they preserve the properties of the Klein four-group. They are commutative, and $\kappa'_u = \kappa'_v \kappa'_w$ for $\{u, v, w\} = \{1, 2, 3\}$.

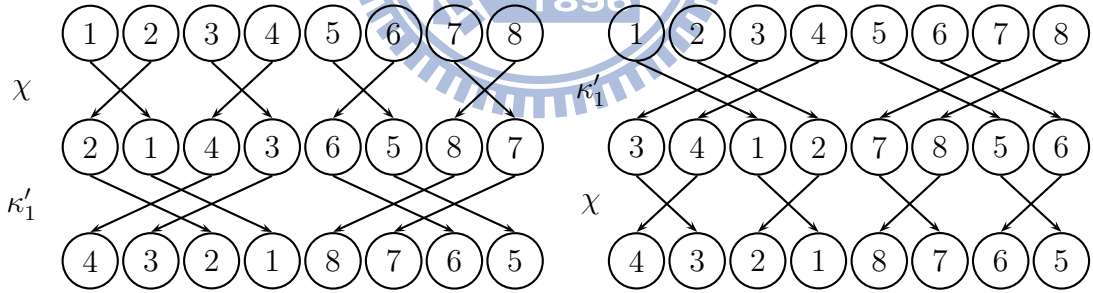


Figure 6.5: $\kappa'_1 \chi = \chi \kappa'_1$

The rest part is to show that $\kappa'_i \chi = \chi \kappa'_i$ for $i \in [3]$. Figure 6.5 shows $\kappa'_1 \chi = \chi \kappa'_1$. It is also easy to verify $\kappa'_2 \chi = \chi \kappa'_2$ and $\kappa'_3 \chi = \chi \kappa'_3$. These four permutations are commutative. \square

From the result of Proposition 6.2.3, we can derive $K'_8\{e, \kappa'_1, \kappa'_2, \kappa'_3, \chi, \chi\kappa'_1, \chi\kappa'_2, \chi\kappa'_3\}$ is a group with composition operation, see Table 6.3. We list the weights of elements in K'_8 in Table 6.4.

Table 6.3: Operation of basic construction blocks and their products.

\circ	e	κ'_1	κ'_2	κ'_3	χ	$\chi\kappa'_1$	$\chi\kappa'_2$	$\chi\kappa'_3$
e	e	κ'_1	κ'_2	κ'_3	χ	$\chi\kappa'_1$	$\chi\kappa'_2$	$\chi\kappa'_3$
κ'_1	κ'_1	e	κ'_3	κ'_2	$\chi\kappa'_1$	χ	$\chi\kappa'_3$	$\chi\kappa'_2$
κ'_2	κ'_2	κ'_3	e	κ'_1	$\chi\kappa'_2$	$\chi\kappa'_3$	χ	$\chi\kappa'_1$
κ'_3	κ'_3	κ'_2	κ'_1	e	$\chi\kappa'_3$	$\chi\kappa'_2$	$\chi\kappa'_1$	χ
χ	χ	$\chi\kappa'_1$	$\chi\kappa'_2$	$\chi\kappa'_3$	e	κ'_1	κ'_2	κ'_3
$\chi\kappa'_1$	$\chi\kappa'_1$	χ	$\chi\kappa'_3$	$\chi\kappa'_2$	κ'_1	e	κ'_3	κ'_2
$\chi\kappa'_2$	$\chi\kappa'_2$	$\chi\kappa'_3$	χ	$\chi\kappa'_1$	κ'_2	κ'_3	e	κ'_1
$\chi\kappa'_3$	$\chi\kappa'_3$	$\chi\kappa'_2$	$\chi\kappa'_1$	χ	κ'_3	κ'_2	κ'_1	e

Table 6.4: Weights of the basic construction blocks

$wt(e)$	$wt(\kappa'_1)$	$wt(\kappa'_2)$	$wt(\kappa'_3)$	$wt(\chi)$	$wt(\chi\kappa'_1)$	$wt(\chi\kappa'_2)$	$wt(\chi\kappa'_3)$
0	2	4	6	1	3	5	7

However, the weights of the basic construction blocks do not fit our reduction. To adjust the weight without destroying the properties in Proposition 6.2.3, we relabel the symbols in gadgets with some permutation $q \in S_N$ such that

$$q(x) = \begin{cases} 1 & , x = 1. \\ \alpha + x - 1 & , 1 < x < 8. \\ 2\alpha + 6 & , x = 8. \end{cases}$$

Then, we define the modified basic blocks $\kappa''_1, \kappa''_2, \kappa''_3$ and χ' as follows,

$$\begin{aligned} \kappa''_1 &= q\kappa'_1q^{-1} \\ &= \overline{(q(1), q(3))} \overline{(q(5), q(7))} \overline{(q(2), q(4))} \overline{(q(6), q(8))} \\ &= \overline{(1, \alpha + 2)} \overline{(\alpha + 4, \alpha + 6)} \overline{(\alpha + 1, \alpha + 3)} \overline{(\alpha + 5, 2\alpha + 6)}, \\ \kappa''_2 &= q\kappa'_2q^{-1} \\ &= \overline{(q(1), q(5))} \overline{(q(3), q(7))} \overline{(q(2), q(6))} \overline{(q(4), q(8))} \\ &= \overline{(1, \alpha + 4)} \overline{(\alpha + 2, \alpha + 6)} \overline{(\alpha + 1, \alpha + 5)} \overline{(\alpha + 3, 2\alpha + 6)}, \end{aligned}$$

$$\begin{aligned}
\kappa_3'' &= q\kappa_3'q^{-1} \\
&= \overline{(q(1), q(7))} \overline{(q(3), q(5))} \overline{(q(2), q(8))} \overline{(q(4), q(6))} \\
&= \overline{(1, \alpha + 6)} \overline{(\alpha + 2, \alpha + 4)} \overline{(\alpha + 1, 2\alpha + 6)} \overline{(\alpha + 3, \alpha + 5)}, \\
\chi' &= q\chi q^{-1} \\
&= \overline{(q(1), q(2))} \overline{(q(3), q(4))} \overline{(q(5), q(6))} \overline{(q(7), q(8))} \\
&= \overline{(1, \alpha + 1)} \overline{(\alpha + 2, \alpha + 3)} \overline{(\alpha + 4, \alpha + 5)} \overline{(\alpha + 6, 2\alpha + 6)}.
\end{aligned}$$

And we define $K_8'' = \{e, \kappa_1'', \kappa_2'', \kappa_3'', \chi', \chi'\kappa_1'', \chi'\kappa_2'', \chi'\kappa_3''\}$.

Since this modification is simply done by relabeling, it is clear that the cycle structures are preserved. Hence, the algebraic structures of K_8' and K_8'' are isomorphic. The weight of these blocks and their products are as in Table 6.5. We will use them to manipulate the weight of clause gadgets.

Table 6.5: Weights of modified basic blocks

$wt(e)$	$wt(\kappa_1'')$	$wt(\kappa_2'')$	$wt(\kappa_3'')$	$wt(\chi')$	$wt(\chi'\kappa_1'')$	$wt(\chi'\kappa_2'')$	$wt(\chi'\kappa_3'')$
0	$\alpha + 1$	$\alpha + 3$	$\alpha + 5$	α	$\alpha + 2$	$\alpha + 4$	$2\alpha + 5$

Now, we define the second layer construction blocks

$$\begin{aligned}
h_1 &= \kappa_1'' s_\beta(\kappa_2'') s_{2\beta}(\kappa_3''), \\
h_2 &= \kappa_2'' s_\beta(\kappa_3'') s_{2\beta}(\kappa_1''), \\
h_3 &= \kappa_3'' s_\beta(\kappa_1'') s_{2\beta}(\kappa_2''), \\
h^* &= \chi' \kappa_3'' s_\beta(\chi' \kappa_3'') s_{2\beta}(\chi' \kappa_3'').
\end{aligned}$$

Since $\beta = 2\alpha + 6$, all of the above blocks permute $[1, \beta]$, $[\beta + 1, 2\beta]$ and $[2\beta + 1, 3\beta]$ separately, see Figure 6.6. The second layer blocks inherit the self-inverse property, commutativity, and the weight from the modified basic blocks.

Proposition 6.2.4. h_1, h_2, h_3 and h^* are self-inverse and commutative. Moreover,

- $wt(h_k) = wt(\kappa_3'') = \alpha + 5$ for $k \in [3]$.

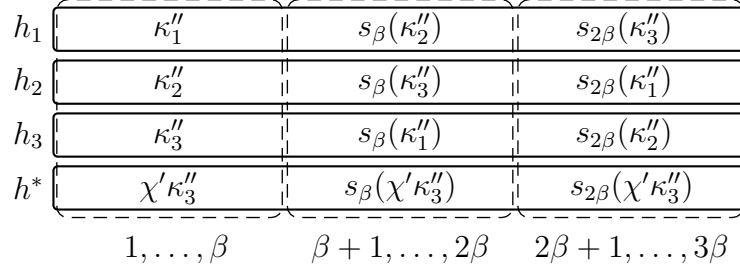


Figure 6.6: Second layer blocks

- $wt(h^*) = wt(\chi \kappa''_3) = 2\alpha + 5$.
- $wt(h^* h_k) = wt(\chi' \kappa''_k) = \alpha + 4$ for $k \in [3]$.

Proof. Since no position is permuted by two of $\kappa''_1, s_\beta(\kappa''_2)$ and $s_{2\beta}(\kappa''_3)$, we have

$$wt(h_1) = \max(wt(\kappa''_1), wt(s_\beta(\kappa''_2)), wt(s_{2\beta}(\kappa''_3))) = \alpha + 5.$$

Similarly, $wt(h_2) = wt(h_3) = wt(\kappa''_3) = \alpha + 5$, and it is clear that $wt(h^*) = wt(\chi \kappa''_3) = 2\alpha + 5$. Since $\kappa''_1, \kappa''_2, \kappa''_3$ and χ' inherits the multiplicative structure mentioned in Proposition 6.2.3 from $\kappa'_1, \kappa'_2, \kappa'_3$ and χ , we have

$$h^* h_1 = \chi' \kappa''_2 s_\beta(\chi' \kappa''_1) s_{2\beta}(\chi'),$$

$$h^* h_2 = \chi' \kappa''_1 s_\beta(\chi') s_{2\beta}(\chi' \kappa''_2),$$

and

$$h^* h_3 = \chi' s_\beta(\chi' \kappa''_2) s_{2\beta}(\chi' \kappa''_1).$$

Therefore, $wt(h^* h_1), wt(h^* h_2)$ and $wt(h^* h_3)$ all equal

$$\max(wt(\chi'), wt(\chi' \kappa''_1), wt(\chi' \kappa''_2)) = \alpha + 4.$$

□

By using the second layer blocks, the components corresponding to the k -th literal of the

j -th clause assigned true and false are defined respectively as

$$h_{j,k,T} = s_{6\beta(j-1)}(h^*h_k),$$

$$h_{j,k,F} = s_{6\beta(j-1)+3\beta}(h^*h_k),$$

where $k \in [3]$ and $j \in [m]$. For every $j \in [m]$ and $t \in \{T, F\}$, by Proposition 6.2.4, we have

- For every $k \in [3]$, $h_{j,k,t}$ has weight $\alpha + 4$.
- For distinct $k, k' \in [3]$, $h_{j,k,t}h_{j,k',t}$ has weight $\alpha + 5$.
- $h_{j,1,t}h_{j,2,t}h_{j,3,t}$ has weight $2\alpha + 5$.

Let

$$P = \{(i, j, k) : x_i \text{ is the } k\text{-th literal in } c_j\},$$

$$Q = \{(i, j, k) : \bar{x}_i \text{ is the } k\text{-th literal in } c_j\}.$$

Now, we define the clause gadgets of $g_1(\phi), \dots, g_n(\phi), g'_1(\phi), \dots, g'_n(\phi)$ as follows.

$$g_{i|c}(\phi) = \left(\prod_{(i,j,k) \in P} h_{j,k,T} \right) \left(\prod_{(i,j,k) \in Q} h_{j,k,F} \right),$$

$$g'_{i|c}(\phi) = \left(\prod_{(i,j,k) \in P} h_{j,k,F} \right) \left(\prod_{(i,j,k) \in Q} h_{j,k,T} \right).$$

Lemma 6.2.5. *About the clause gadgets, the following statements are true.*

1. For $\pi \in f(\phi)$, $wt(\pi_c) \leq 2\alpha + 5$.
2. For $\pi \in \mathcal{C}_1^u(\phi)$, $wt(\pi_c) = 2\alpha + 5$.
3. For $\pi \in \mathcal{C}_1^s(\phi)$, $wt(\pi_c) = \alpha + 5$.

Proof. Since $h_{j,k,t}$ and $h_{j',k',t'}$ permute disjoint positions for $j \neq j'$ or $t \neq t'$, we have that

$$wt(\pi_c) \leq \max_{u,v,w \in \{0,1\}} \{(h^*h_1)^u (h^*h_2)^v (h^*h_3)^w\} = 2\alpha + 5$$

for $\pi \in f(\phi)$. The first statement is true.

Now, consider permutation $\pi = \mu(\sigma)$ where σ is an assignment for formula ϕ . From the definition of π_c , we have

$$\pi_c = \left(\prod_{i \in [n]} g_{i|c}(\phi)^{z_i(\pi)} \right) \left(\prod_{i \in [n]} g'_{i|c}(\phi)^{z'_i(\pi)} \right).$$

If the k -th literal in the j -th clause c_j is x_i and $\sigma(x_i) = \text{T}$, then π_c has a factor $g_{i|c}(\phi)$ and $g_{i|c}(\phi)$ has a factor $h_{j,k,T}$, since $(i, j, k) \in P$. Similarly, if the k -th literal in the j -th clause c_j is \bar{x}_i and $\sigma(x_i) = \text{F}$, then π_c has a factor $g'_{i|c}(\phi)$ and $g'_{i|c}(\phi)$ has a factor $h_{j,k,T}$, since $(i, j, k) \in Q$. Thus, we conclude that if the k -th literal in the j -th clause c_j is assigned to be true, then $h_{j,k,T}$ is a factor of π_c , but not for $h_{j,k,F}$. As a consequence of applying similar arguments, we know that if σ makes the k -th literal in c_j false, then $h_{j,k,F}$ is a factor of π_c , but $h_{j,k,T}$ is not.

If all literals in c_j are assigned to be true, then $h_{j,1,T}h_{j,2,T}h_{j,3,T} = s_{6\beta(j-1)}(h^*)$ is a factor of π_c and $wt(\pi_c) \geq 2\alpha + 5$. Similarly, if σ assigns all literals in c_j to be false, then

$$wt(\pi_c) \geq wt(s_{6\beta(j-1)+3\beta}(h_{j,1,F}h_{j,2,F}h_{j,3,F})) = wt(s_{6\beta(j-1)+3\beta}(h^*)) = 2\alpha + 5.$$

For every assignment σ unsatisfying ϕ , there exists a clause in which all literals are assigned to the same truth value. Hence, every $\pi \in f(\phi)$ has weight at least $2\alpha + 5$. The second statement follows by this observation and the first statement.

If the k -th literal in c_j is the only literal assigned to be true by σ , then π_c has a factor $h_{j,k,T}h_{j,v,F}h_{j,w,F}$ for $\{k, v, w\} = \{1, 2, 3\}$. In this case, the weight contributed from symbol $6\beta j - 6\beta + 1$ to $6\beta j$ is $\max(wt(h_{j,k,T}), wt(h_{j,v,F}h_{j,w,F})) = \alpha + 5$. Similarly, if σ only assigns the k -th literal in c_j to be false, then the weight contributed from symbol $6\beta j - 6\beta + 1$ to $6\beta j$ is $\max(wt(h_{j,k,F}), wt(h_{j,v,T}h_{j,w,T})) = \alpha + 5$ for $\{k, v, w\} = \{1, 2, 3\}$. Note that every satisfying σ assigns either one or two literals in each clause to be true. Hence, for $\pi \in \mathcal{C}_1^s(\phi)$, we have $wt(\pi_c) = \alpha + 5$. The third statement is true. \square

The variable gadgets assure that no variable is assigned both true and false values. They permute elements in $[6m\beta + 1, 6m\beta + n\beta]$. Let q' be a permutation in S_N such that $q'(1) =$

$1, q'(2) = \alpha + 5, q'(3) = \alpha + 6$ and $q'(4) = 2\alpha + 6$. By relabeling κ_1 and κ_2 by q' , we define

$$v_T = q' \kappa_1 q'^{-1} = \overline{(q'(1), q'(2))} \overline{(q'(3), q'(4))} = \overline{(1, \alpha + 1)} \overline{(\alpha + 6, 2\alpha + 6)}$$

and

$$v_F = q' \kappa_2 q'^{-1} = \overline{(q'(1), q'(3))} \overline{(q'(2), q'(4))} = \overline{(1, \alpha + 6)} \overline{(\alpha + 1, 2\alpha + 6)}.$$

Note that

$$v_T v_F = q' \kappa_1 q'^{-1} q' \kappa_2 q'^{-1} = q' \kappa_3 q'^{-1} = \overline{(1, 2\alpha + 6)} \overline{(\alpha + 1, \alpha + 6)}.$$

The weights of v_T , v_F , and $v_T v_F$ are α , $\alpha + 5$, and $2\alpha + 5$, respectively. The variable gadgets are defined as

$$g_{i|v} = s_{b_1 + \beta i}(v_T), \text{ and}$$

$$g'_{i|v} = s_{b_1 + \beta i}(v_F),$$

where $b_1 = 6m\beta - \beta$. Since they are made of Klein-four group K_4 by relabeling, they are self-inverse and commutative.

Lemma 6.2.6. *For $\pi \in f(\phi)$, $wt(\pi_v) = 2\alpha + 5$ if $\pi \in \mathcal{C}_2(\phi)$, otherwise $wt(\pi_v) \leq \alpha + 5$.*

Proof. Assume $\pi \in \mathcal{C}_2(\phi)$. There exists i such that $z_i(\pi) + z'_i(\pi) = 2$. We have $z_i(\pi) = z'_i(\pi) = 1$, and this implies that $g_{i|v}$ and $g'_{i|v}$ are both factors of π_v . Hence, $wt(\pi_v) \geq wt(v_T v_F) = 2\alpha + 5$. Because it is clear that the maximum weight of the variable gadgets is $2\alpha + 5$, we know that $wt(\pi_v) = 2\alpha + 5$ for $\pi \in \mathcal{C}_2(\phi)$.

For $\pi \in f(\phi) \setminus \mathcal{C}_2(\phi)$ and $i \in [n]$, we have $z_i(\pi) + z'_i(\pi) < 2$. Thus, at most one of $g_{i|v}$ and $g'_{i|v}$ is a factor of π_v . We have $wt(\pi_v) \leq \max(0, wt(v_T), wt(v_F)) = \alpha + 5$, and the lemma is true. \square

The assignment gadgets assure that if x_i is assigned a value, then x_{i+1} and x_{i-1} are both assigned, where $i \pm 1 \in \mathbb{Z}_n$ and $x_0 \equiv x_n$. They permute elements $6m\beta + n\beta + 1, \dots, N$. We use the following permutation to give a chain reaction, i.e., if there is any missing gadget, then the weight will deviate significantly. Let

$$u_i = s_{b_2 + \beta i - \beta} \left(\overline{(1, 2\alpha + 6)} \right) s_{b_2 + \beta i} \left(\overline{(1, 2\alpha + 6)} \right),$$

for $i < n$ and

$$u_n = s_{b_2+\beta n-\beta} \left(\overline{(1, 2\alpha+6)} \right) s_{b_2} \left(\overline{(1, 2\alpha+6)} \right)$$

where $b_2 = 6m\beta + n\beta$. For convenience, we also use u_0 as the alias of u_n . The assignment gadgets are defined as:

$$g_{i|a} = g'_{i|a} = u_i,$$

for $i \in [n]$ and they are clearly self-inverse and commutative.

Lemma 6.2.7. *About the assignment gadgets, the following statements are true.*

1. For $\pi \in f(\phi)$, $wt(\pi_a) \leq 2\alpha + 5$.
2. For $\pi \in \mathcal{C}_1^s(\phi) \cup \mathcal{C}_1^u(\phi)$, $wt(\pi_a) = 0$.
3. For $\pi \in \mathcal{C}_3(\phi)$, $wt(\pi_a) = 2\alpha + 5$.

Proof. Since $\beta = 2\alpha + 6$ and the assignment gadgets swap pairs $(v, v + 2\alpha + 5)$ only for integers $v \equiv 1 \pmod{\beta}$, the first statement is clearly true. For $\pi \in \mathcal{C}_1^s(\phi) \cup \mathcal{C}_1^u(\phi)$, we have $z_i(\pi) + z'_i(\pi) = 1$. Thus,

$$\begin{aligned} \pi_a &= \prod_{i \in [n]} g_{i|a}^{z_i(\pi)} g'_{i|a}^{z'_i(\pi)} \\ &= \prod_{i \in [n]} u_i \\ &= \left(\prod_{i \in [n]} \overline{(b_2 + \beta i - \beta + 1, b_2 + \beta i)} \right)^2 \\ &= \mathbf{e}. \end{aligned}$$

All pairs are swapped twice by π_a , i.e., every element permuted by assignment gadgets π_a remains in its original position, so we have $wt(\pi_a) = 0$.

For $\pi \in \mathcal{C}_3(\phi)$, we have $z_i(\pi) + z'_i(\pi) < 2$ for every $i \in [n]$. Moreover, there exists $i_0, i_1 \in [n]$ such that $z_{i_0}(\pi) + z'_{i_0}(\pi) = 0$ and $z_{i_1}(\pi) + z'_{i_1}(\pi) = 1$, because $\pi \neq \mathbf{e}$. Now recall that for $i < n$

$$u_i = s_{b_2+\beta i-\beta} \left(\overline{(1, 2\alpha+6)} \right) s_{b_2+\beta i} \left(\overline{(1, 2\alpha+6)} \right),$$

and

$$u_n = s_{b_2+\beta n-\beta} \left(\overline{(1, 2\alpha+6)} \right) s_{b_2} \left(\overline{(1, 2\alpha+6)} \right).$$

Without loss of generality, we can assume that $i_0 = i_1 - 1$. As a consequence, $g_{i_0|a}$, $g'_{i_0|a}$, $g_{i_1|a}$ and $g'_{i_1|a}$ are the only gadgets permuting $b_2 + \beta i_1 - \beta + 1$ and $b_2 + \beta i_1$. Since $z_{i_0}(\pi) + z'_{i_0}(\pi) + z_{i_1}(\pi) + z'_{i_1}(\pi) = 0 + 1 = 1$, exactly one of $g_{i_0|a}$, $g'_{i_0|a}$, $g_{i_1|a}$ and $g'_{i_1|a}$ can be a factor of π_a . Therefore, $b_2 + \beta i_1 - \beta + 1$ and $b_2 + \beta i_1$ must be swapped by π_a . We have $wt(\pi_a) = 2\alpha + 5$ in this case. We conclude that all statements are true. \square

With Lemmas 6.2.5, 6.2.6 and 6.2.7, we know that the gadgets contribute weights as in table 6.1. Since the construction can be done in polynomial time of $|\phi|$ for constant α , this shows Theorem 6.2.2 is true. Since NAESAT is an NP-complete problem, we have the following corollary as an immediate result of Theorem 6.2.2.

Corollary 6.2.8. *MINWSPA $_{\ell_\infty}$ is NP-complete.*

Proof. MINWSPA $_{\ell_\infty}$ is in NP, since we can finish computing the weight of any permutation π and verifying if π is in the subgroup by Schreier-Sims algorithm [33] in polynomial time. Set $\alpha = 1$. By Theorem 6.2.2, ϕ is satisfiable if and only if the corresponding subgroup $f(\phi)$ has minimum weight at most $\alpha + 5 = 6$. Hence, we can conclude MINWSPA $_{\ell_\infty}$ is NP-complete. \square

A feasible solution for minimizing the weight of a subgroup permutation code C is a permutation in C , and the cost of the permutation equals its weight. Note that an approximation algorithm A cannot output an answer whose cost is less than the minimum cost, since it is not a feasible solution. According to this fact, we derive the following inapproximable result.

Theorem 6.2.9. *For any constant $\epsilon > 0$, there does not exist a polynomial-time $(2 - \epsilon)$ -approximate algorithm for finding the minimum weight of a subgroup permutation code C unless $P = NP$.*

Proof. Assume A is a polynomial-time $(2 - \epsilon)$ -approximate algorithm. We can construct a polynomial-time algorithm to solve NAESAT by using A .

1. Set $\alpha > \frac{5}{\epsilon}$.
2. For formula ϕ , construct the subgroup $H = f_\alpha(\phi)$ and run $A(H)$.
3. If $A(H)$ outputs a number no more than $(2 - \epsilon)(\alpha + 5)$, then accept ϕ , otherwise reject.

For a satisfiable formula ϕ , the minimum weight of H is $\alpha + 5$ and $A(H) \leq (2 - \epsilon)(\alpha + 5)$. Hence, ϕ would be accept by the above algorithm. For unsatisfiable ϕ , the minimum weight of H is $2\alpha + 5$. Since $\alpha\epsilon > 5$ and an approximate algorithm cannot give an answer less than the minimum solution, we have

$$A(H) \geq 2\alpha + 5 > 2\alpha + 10 - \alpha\epsilon - 5\epsilon = (2 - \epsilon)(\alpha + 5).$$

This implies that NAESAT is in P if such A exists. Since NAESAT is a NP-complete problem, the theorem is true. □

6.3 Cameron-Wu's Reduction

The reduction in Cameron and Wu's work [7] uses only two kinds of gadgets. Their variable gadget v_i for the i -th variable is $(2i - 1, 2i)$ and their clause gadget $h_{j,k}$ for the k -th literal in the j -th clause is defined as $s_{2n+24(j-1)}(h_k)$ where $s_b(\cdot)$, h_1 , h_2 and h_3 are the same as in Section 6.2 with $\alpha = 1$. The generators are defined as

$$g_i = v_i \left(\prod_{(i,j,k) \in P} s_{2n+24(j-1)}(h_k) \right),$$

$$g'_i = v_i \left(\prod_{(i,j,k) \in Q} s_{2n+24(j-1)}(h_k) \right),$$

where P, Q are also defined as

$$P = \{(i, j, k) : x_i \text{ is the } k\text{-th literal in } c_j\},$$

$$Q = \{(i, j, k) : \bar{x}_i \text{ is the } k\text{-th literal in } c_j\}.$$

In addition, they construct another generator

$$g^* = \prod_{j \in [m]} s_{2n+24(j-1)}(h^*)$$

where $h^* = \overline{(1, 8)}\overline{(2, 7)}\overline{(3, 6)}\overline{(4, 5)}\overline{(9, 16)}\overline{(10, 15)}\overline{(11, 14)}\overline{(12, 13)}\overline{(17, 24)}\overline{(18, 23)}\overline{(19, 22)}\overline{(20, 21)}$, which is identical to ours by setting $\alpha = 1$. However, their construction does not work in the following instance. Let $\phi = (x_1 \vee x_2 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2)$. By their construction, the corresponding subgroup G is generated by

$$\begin{aligned} g_1 &= v_1 h_{1,1} = \overline{(1, 2)} s_4(h_1), \\ g'_1 &= v_1 h_{2,1} = \overline{(1, 2)} s_{28}(h_1), \\ g_2 &= v_2 h_{1,2} h_{1,3} h_{2,2} h_{2,3} \\ &= \overline{(3, 4)} s_4(h_2) s_4(h_3) s_{28}(h_2) s_{28}(h_3) \\ &= \overline{(3, 4)} s_4(h_1) s_{28}(h_1), \\ g'_2 &= v_2 = \overline{(3, 4)}, \\ g^* &= s_4(h^*) s_{28}(h^*). \end{aligned}$$

Note that ϕ is an unsatisfiable formula for NAESAT. According to their proof of Theorem 18, [7], elements of G should not have weight 5, since ϕ is unsatisfiable. But $g^* g_1 g'_1 = s_4(h^* h_1) s_{28}(h^* h_1)$ has weight 5. Therefore, we need to design the gadgets more carefully to prove that $\text{MINWSPA}_{\ell_\infty}$ is NP-complete.



Chapter 7

Conclusion and Future Works

7.1 Conclusion

In this thesis, we characterize what are good FPAs for different purposes. We prove lower and upper bounds on the cardinality of FPAs, and we provide an efficient algorithm for computing the ball size under Chebyshev distance, which is a combinatorial problem highly related to estimate the cardinality of FPAs. In addition, we give a few constructions for FPAs of certain good properties. We give an efficient encoding and three different types of decoding algorithms for a family of FPAs. However, we also find that there are several negative results. We show that some problems related to the design of good FPAs are NP-hard.

7.2 Future Works

There are many future works and improvement to be done. We list some of them as follows.

1. More accurate estimation on the ball size. Our estimation makes use of inequalities on permanent. However, they are not tight.
2. Tighter lower and upper bounds for the cardinality of FPAs. Although a good estimation on the ball size would help, bounds may not have any relation to the ball size. For example, see Section 7.3.
3. Better construction methods for FPAs. Our explicit construction in Chapter 4 just

beats the Gilbert-Varshamov type lower bound closely in some case. Besides, the code family in Chapter 5 has even lower information rate than the Gilbert-Varshamov type bound in Chapter 3. It is interesting to seek a family of FPAs with both higher information rate and efficient decoders.

4. Finding new locally decodable FPAs, list decodable FPAs and related bounds. We have shown there is a family of FPAs which have both a local decoding algorithm and a list decoding algorithm. We wonder whether there is another family exhibiting these properties. On the other hand, is there any limitation of these kinds of FPAs?
5. Approximation algorithms for SDSPA_δ and CDSPA_δ . From previous results [6] and [7], we know both problems are NP-hard under various metrics. But, if some algorithm can approximate SDSPA_δ or CDSPA_δ within certain factor, then it might help us construct FPAs of certain information rate or efficient decoders.
6. Complexity results on other related problems of FPAs. Besides the minimum distance and the decoding problem, there are still other interesting problems around error correcting codes. For instance, the covering radius problem, see Section 7.5.
7. FPA-based PIR schemes with better privacy. The space overhead and communication complexity of our PIR scheme are small, however, its privacy is distant from perfectness. It is possible to derive a better PIR from a new locally decodable FPA. Also, we can construct PIR with a new local decoding algorithm on a known FPA.
8. New applications for FPAs. In chapter 19 of Arora and Barak's book [2], they showed an application of locally decodable codes, hardness amplification. Since we give a local decoder for the codes constructed in Chapter 5, we believe there is a similar result. Another possibility is to find applications in steganography (see Section 7.4), since some schemes are based on linear block codes or PAs.

7.3 Code-Anticode Type Bounds

In this thesis, We mainly investigate on the lower and upper bounds which are related to the size of balls. However, there are some other types of bounds in coding theory. For example, the code-anticode upper bound is a generalization of the sphere-packing bound.

An anticode A of maximum distance d is a set such that for $x, y \in A$, the distance between x and y are at most d . For any linear code $C \subseteq \mathbb{F}_q^n$ of minimum distance d and anticode $A \subseteq \mathbb{F}_q^n$ of maximum distance $d - 1$, it is well-known that $|C| \cdot |A| \leq q^n$. Note that a ball of radius r is a special anticode of maximum distance $2r$, since the distance between any two elements is at most $2r$ due to triangle inequality. Hence, the sphere-packing upper bound is only a special case of code-anticode type upper bound. In general, if one constructs a larger anticode of certain maximum distance $d - 1$, then the code-anticode type upper bound becomes tighter.

Tamo and Schwartz [37] gave some upper bounds for permutation arrays under ℓ_∞ -metric by constructing anticodes. In their derivation, they showed the correctness of the code-anticode argument $|C| \cdot |A| \leq |S_n|$ by exploiting the use of inverse permutations, since S_n is a group with composition operation. However, their methods cannot be directly applied to FPAs, because we do not have a group operator for S_n^λ where $\lambda > 1$. We aim to generalize the code-anticode type methods for bounding the cardinality of FPAs, and then we can derive tighter bounds by constructing better anticodes.

Conjecture 7.3.1. *There exists a code-anticode type bound for FPAs.*

7.4 Steganography

Steganography is the knowledge about hiding some information to a file in some way such that only the sender and the receiver can learn the hidden information while others cannot sense the existence of the hidden information. The sender modifies a cover object, such as a picture, a video clip or an audio file, in a certain manner to hide the information.

Many multimedia formats have potential to hide some extra information in themselves. For image formats, the most well-known steganography method is embedding the information

on the least significant bits of the pixels in the cover object. This method creates some distortion on the cover image. However, there is another approach exploiting the nature of permutation to embed information without generating any visible distortion. For example, we can change the color order in the palette of a GIF (see Kwan's website[27]) or a PNG (see Stevenson's work[34]) picture without changing how it looks. For another instance, Inoue and Matsumoto [19] showed how to embed message by reordering the MIDI file. These works exploit the order of elements, thus we may use a permutation code for representing the information.

Conjecture 7.4.1. *Some steganography scheme makes use of FPAs.*

7.5 Covering Radius

The covering radius of an FPA $C \subseteq S_n^\lambda$ is defined as the minimum radius r such that for every frequency permutation $\mathbf{x} \in S_n^\lambda$, the ball of radius r centered at \mathbf{x} contains at least one frequency permutation in C . In coding theory, this quantity r has an important role. The maximum likelihood decoding algorithm, which outputs a codeword closest to the received string, recovers at most r errors from a received string while it can recover at least $\frac{d-1}{2}$ errors where d is the minimum distance. There are some other applications related to the covering radius of codes. For example, the maximum distortion of matrix embedding scheme [18] proposed by Fridrich and Soukal for steganography is exactly the cover radius of the corresponding linear code.

The decision version problem for computing the covering radius of a code is defined as follows.

Definition 7.5.1. *Given a code C and a metric δ , determine whether the covering radius of C under δ is less than or equal to some value b .*

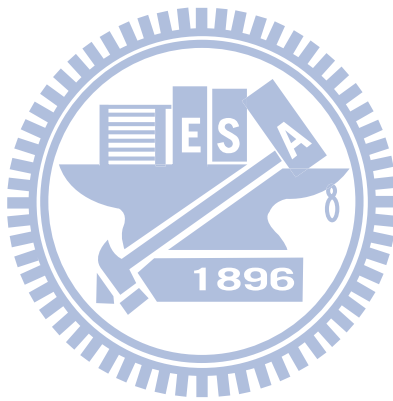
Similar to what we mentioned in Chapter 6, when the problem instance is of size $\mathcal{O}(\log |C|)$, the naive algorithm for covering radius runs inefficiently in exponential time. Moreover, the decision problem for the covering radius problem of a binary linear code under Hamming distance is Π_2^P -complete, see McLoughlin's work [29]. McLoughlin actually gave a

polynomial-time reduction from the AE qualified 3-Dimensional matching problem. But we cannot directly apply this reduction to the covering radius problem of a permutation code, since permutation codes have some different nature from the linear codes.

Definition 7.5.2. ($CRSPA_\delta$) *Given a code C and a metric δ , determine whether the covering radius of C under δ is less than or equal to some value b .*

Conjecture 7.5.3. $CRSPA_{\ell_\infty}$ is Π_2^P -complete.





Bibliography

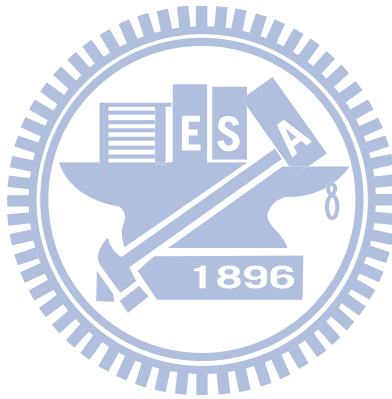
- [1] S. Arora, L. Babai, J. Stern, Z. Sweedyk, “The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations,” *Journal of Computer and System Science*, vol. 54, 1997, pp. 317–331.
- [2] S. Arora, B. Barak, *Computational Complexity*, Cambridge University Press, 2009.
- [3] A. A. Babaev, “Procedures of Encoding and Decoding of Permutations,” *Cybernetics and Systems Analysis*, vol. 20, pp. 861–863, 1984.
- [4] E. R. Berlekamp, R. J. McEliece, H. C.A. van Tilborg, “On the Inherent Intractibility of Certain Coding Problems,” *IEEE Transactions on Information Theory*, pp. 384–386, 1978.
- [5] I. Blake, “Permutation Codes for Discrete Channels,” *IEEE Transactions on Information Theory*, vol. 20, pp. 138–140, 1974.
- [6] C. Buchheim, P. J. Cameron, T. Wu, “On the Subgroup Distance Problem,” *Discrete Mathematics*, vol. 309, pp. 962–968, 2009.
- [7] P. J. Cameron, T. Wu, “The Complexity of the Weight Problem for Permutation and Matrix Groups,” *Discrete Mathematics*, vol. 310, pp. 408–416, 2010.
- [8] P. Cappelletti, C. Golla, P. Olivo, E. Zanon, *Flash Memories*, Kluwer Academic Publishers, 1999.
- [9] J.-C. Chang, R.-J. Chen, T. Kløve, S.-C. Tsai, “Distance-Preserving Mappings from Binary Vectors to Permutations,” *IEEE Transactions on Information Theory*, vol. 49, pp. 1054–1059, 2003.

- [10] J.-C. Chang, “Distance-Increasing Mappings from Binary Vectors to Permutations,” *IEEE Transactions on Information Theory*, vol. IT-51, pp. 359–363, 2005.
- [11] J.-C. Chang, “Distance-Increasing Mappings from Binary Vectors to Permutations that Increase Hamming Distances by at Least Two,” *IEEE Transactions on Information Theory*, vol. 52, pp. 1683–1689, 2006.
- [12] C. J. Colbourn, T. Kløve, “Permutation Arrays for Powerline Communication and Mutually Orthogonal Latin Squares,” *IEEE Transactions on Information Theory*, vol. 50, pp. 1289–1291, 2004.
- [13] D. R. de la Torre, C. J. Colbourn, and A. C. H. Ling, “An Application of Permutation Arrays to Block Cipher,” *Congressus Numerantium*, vol. 145, pp. 5–7, 2000.
- [14] M. Deza, S. A. Vanstone, “Bounds on Permutation Arrays,” *Journal of Statistical Planning and Inference*, vol. 2, pp. 19–209, 1978.
- [15] I. Dinur, “Approximating SVP_∞ to within almost Polynomial Factors is NP-hard,” *Combinatorica*, vol. 23, pp. 205–243, 2003.
- [16] K. Efremenko, “3-Query Locally Decodable Codes of Subexponential Length,” in *Proceedings of ACM Symposium on Theory of Computing*, pp. 39–44, 2009.
- [17] S. Huczynska, G. L. Mullen, “Frequency Permutation Arrays,” *Journal of Combinatorial Designs*, vol. 14, pp. 463–478, 2006.
- [18] J. Fridrich and D. Soukal, “Matrix Embedding for Large Payloads,” *IEEE Transactions on Information Forensics and Security*, vol. 1, pp. 390–395, 2006.
- [19] D. Inoue, T. Matsumoto, “A scheme of Standard MIDI Files steganography and its evaluation,” *Security and Watermarking of Multimedia Contents IV*, pp. 194–205, 2002.
- [20] A. Jiang, R. Mateescu, M. Schwartz, J. Bruck, “Rank Modulation for Flash Memories,” in *Proceedings of IEEE International Symposium on Information Theory*, pp. 1731–1735, 2008.

- [21] A. Jiang, M. Schwartz and J. Bruck, "Error-Correcting Codes for Rank Modulation," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 1736–1740, 2008.
- [22] S. Khot, "Hardness of Approximating the Shortest Vector Problem in Lattices," *Journal of the ACM*, Vol. 52, pp. 789–808, 2005.
- [23] T. Kløve, "Spheres of Permutations under the Infinity Norm - Permutations with Limited Displacement," *Reports in Informatics, Dept. of Informatics, Univ. Bergen*, Report no. 376, 2008.
- [24] T. Kløve, "Frequency Permutation Arrays within Distance one," *Reports in Informatics, Dept. of Informatics, Univ. Bergen*, Report no. 382, 2009.
- [25] T. Kløve, "Lower Bounds on the Size of Spheres of Permutations under the Chebychev Distance," *Designs, Codes and Cryptography*, vol. 59, pp. 183–191, 2011.
- [26] T. Kløve, T.-T. Lin, S.-C. Tsai, W.-G. Tzeng, "Permutation Arrays Under the Chebyshev Distance," *IEEE Transactions on Information Theory*, vol. 56, pp. 2611–2617, 2010.
- [27] M. Kwan, The GIF Shuffle,
<http://www.darkside.com.au/gifshuffle/>
- [28] T.-T. Lin, S.-C. Tsai, W.-G. Tzeng, "Efficient Encoding and Decoding with Permutation Arrays," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 211–214, 2008.
- [29] A. M. McLoughlin, "The Complexity of Computing the Covering Radius of a Code," *IEEE Transactions on Information Theory*, vol. 30, pp. 800–804, 1984.
- [30] C. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Co, 1995.
- [31] K. W. Shum, "Permutation Coding and MFSK Modulation for Frequency Selective Channel," *IEEE Personal, Indoor and Mobile Radio Communications*, vol. 13, pp. 2063–2066, Sept. 2002.

- [32] M. Schwartz, “Efficiently Computing the Permanent and Hafnian of some Banded Toeplitz Matrices,” *Linear Algebra and its Applications*, vol. 430, pp. 1364–1374, 2009
- [33] C. Sims, “Computational Methods in the Study of Permutation Groups”, *Computational Problems in Abstract Algebra*, pp. 169–183, 1970.
- [34] D. E. Stevenson, “PNG Palette Permuter,” in *Proceedings of the 11th annual SIGCSE, Conference on Innovation and Technology in Computer Science Education*, pp. 143–147, 2006.
- [35] T. G. Swart, H. C. Ferreira, “Decoding Distance-Preserving Permutation Codes for Power-Line Communications,” in *Proceedings of IEEE AFRICON*, pp. 1–7, 2007.
- [36] D. Slepian, “Permutation Modulation,” *Proceedings of the IEEE*, vol. 53, pp. 228–236, Mar. 1965.
- [37] I. Tamo, M. Schwartz, “Correcting Limited-Magnitude Errors in the Rank-Modulation Scheme,” *IEEE Transactions on Information Theory*, vol. 56, pp. 2551–2560, Jun. 2010.
- [38] L. Trevisan, “Some Applications of Coding Theory in Computational Complexity,” *Quaderni di Matematica*, vol. 13, pp. 347–424, 2004.
- [39] J. H. van Lint, R. M. Wilson, *A Course in Combinatorics 2nd ed.*, Cambridge University Press, 2001.
- [40] A. Vardy, “The Intractability of Computing the Minimum Distance of a Code,” *IEEE Transactions on Information Theory*, vol. 43, pp. 1757–1766, 1997.
- [41] A. J. H. Vinck, J. Häring, “Coding and Modulation for Power-Line Communications,” in *Proceedings of International Symposium on Power Line Communications*, pp. 265–272, Apr. 2000.
- [42] A. J. H. Vinck, J. Häring, T. Wadayama, “Coded M-FSK for Power Line Communications,” in *Proceedings of IEEE International Symposium on Information Theory*, p. 137, 2000.

- [43] A. J. H. Vinck, “Coded Modulation for Powerline Communications,” *AEU International Journal of Electronics and Communications*, vol. 54, pp. 45–49, 2000.
- [44] Z. Wang, A. A. Jiang, J. Bruck, “On the Capacity of Bounded Rank Modulation for Flash Memories,” in *Proceedings of IEEE International Symposium on Information Theory*, pp. 1234–1238, 2009.
- [45] S. Yekhanin, “Towards 3-query Locally Decodable Codes of Subexponential Length,” *Journal of the ACM*, vol. 55, pp. 1–16, 2008.





Appendix A

Tables of Ball Size

We list the tables of ball size which are not given in previous results.

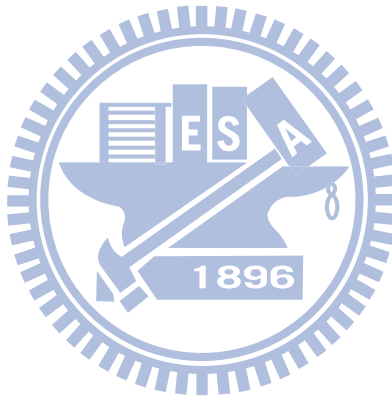


Table A.1: The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 2$

n	$V(2, n, 2, \ell_\infty)$
2	1
4	6
6	90
8	786
10	6139
12	54073
14	477228
16	4113864
18	35579076
20	308945881
22	2679325561
24	23222971098
26	201351085146
28	1745886520422
30	15137227297027
32	131243141767393
34	1137923361184848
36	9866167034815440
38	85542686564024352
40	741681846818742097

Table A.2: The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 3$

n	$V(2, n, 3, \ell_\infty)$
2	1
4	6
6	90
8	2520
10	45450
12	669666
14	9747523
16	154700569
18	2502207156
20	40043708244
22	632349938520
24	9986116318524
26	158192179607364
28	2509767675626581
30	39796612230719845
32	630688880128338378
34	9994168619297530758
36	158396161513685960664
38	2510580301930785916566
40	39792149406721332018414

Table A.3: The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 4$

n	$V(2, n, 4, \ell_\infty)$
2	1
4	6
6	90
8	2520
10	113400
12	3540600
14	88610850
16	2044242426
18	47806940971
20	1196081134201
22	30647443460124
24	784921116539484
26	19899840884886720
28	500019936693729120
30	12551808236761063440
32	315694279415609776404
34	7955400980632212027852
36	200622722060793477132937
38	5057787000067792980984649
40	127452627155747602225756890

Table A.4: The table of ball size under ℓ_∞ -metric for $\lambda = 2, m \in [20], d = 5$

n	$V(2, n, 5, \ell_\infty)$
2	1
4	6
6	90
8	2520
10	113400
12	7484400
14	361859400
16	14091630840
18	489147860970
20	16420511188146
22	563209318269379
24	20416518083009593
26	758713036253909844
28	28351365170599079604
30	1054143198114097909680
32	38864351069181445164480
34	1423417411123883479886400
36	52064892889568503574209920
38	1906534315066176639758670480
40	69931615009402042606373019804

Table A.5: The table of ball size under ℓ_∞ -metric for $\lambda = 3, m \in [20], d = 2$

n	$V(3, n, 2, \ell_\infty)$
3	1
6	20
9	1680
12	61340
15	1886431
18	69496201
21	2568223000
24	91712960320
27	3290467596440
30	118724053748417
33	4276273204804217
36	153904262366842444
39	5541519231941145440
42	199545071017172522244
45	7184755645113714298863
48	258691998154725997048673
51	9314545233907934721851472
54	335381528796576643131475840
57	12075785123501322139824319056
60	434802491356562053648077727185

Table A.6: The table of ball size under ℓ_∞ -metric for $\lambda = 3, m \in [20], d = 3$

n	$V(3, n, 3, \ell_\infty)$
3	1
6	20
9	1680
12	369600
15	41480880
18	3422150780
21	276888204387
24	25512718688405
27	2418264595619240
30	225661997838758560
33	20649533952628896000
36	1889648253594082624960
39	173699198403114756474600
42	16001577154624484682748453
45	1472965856766989578006355117
48	135481185586476496195656612044
51	12459839493182349378716705969200
54	1146141579672729885487800599057600
57	105440511941055519854115528116882480
60	9699923367172090411762252385134967844

Table A.7: The table of ball size under ℓ_∞ -metric for $\lambda = 4, m \in [20], d = 2$

n	$V(4, n, 2, \ell_\infty)$
4	1
8	70
12	34650
16	5562130
20	708212251
24	114774147001
28	18679465660540
32	2906167849870600
36	454904037056013460
40	71729455730285511001
44	11285129375761977675001
48	1773699532985462649188410
52	278931562239767189408085850
56	43869015908453746845566145990
60	6898693708786029238293860809251
64	1084865341390442288732669957148001
68	170605963060816377946936433265175680
72	26829411396875692269491197638918648400
76	4219165662049303123773116859323196816720
80	663502408038018748448058464247159216890001

Table A.8: The table of ball size under ℓ_∞ -metric for $\lambda = 5, m \in [20], d = 2$

n	$V(5, n, 2, \ell_\infty)$
5	1
10	252
15	756756
20	549676764
25	298227062281
30	218838390759073
35	161446400503248672
40	112632613848657302400
45	79169699996993643966432
50	56151546386557366024202177
55	39717291593245217794329362081
60	28058660061656964336359435570604
65	19835819533825566529982592591911412
70	14024417724324420598672399947721245804
75	9914206081036463014882722168252570938889
80	7008596284293402975749309111124669929079521
85	4954676885097638926007640423100194180529855296
90	3502659589845301193905028251874353899223998638208
95	2476160267409321946445662150301548547825713614803904
100	1750492069977099993617695861204414333904857504132837761



Appendix B

Program Codes

B.1 Computing the Ball Size in Python 3

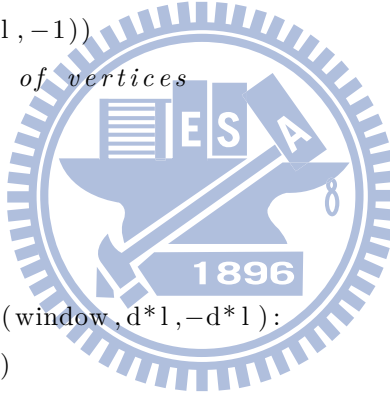
The following is a Python 3 program computing the ball size in $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\binom{d\lambda}{\lambda}m\right)$ time and $\mathcal{O}\left(\binom{2d\lambda}{d\lambda}\right)$ space.

```
1 class FixedSizeSubsets:
2     # Generate s-subsets of u which contain all elements at most m in u
3     def __init__(self, u, s, m):
4         self._univ=[x for x in u if x>m]
5         self._cand=[x for x in u if x<=m]
6         self._size=s
7     def __iter__(self):
8         return self.next()
9     def next(self):
10        len_cand=len(self._cand)
11        len_univ=len(self._univ)
12        if len_cand==self._size:
13            yield self._cand
14        elif len_cand>self._size:
15            pass
16        elif len_univ+len_cand>=self._size:
17            elem = self._univ.pop()
18            # generate the subsets not containing elem
19            for v in self.next():
```

```

20         yield v
21         self._cand.append(elem)
22         # generate the subsets containing elem
23         for v in self.next():
24             yield v
25             self._cand.pop()
26             self._univ.append(elem)
27
28 if __name__ == '__main__':
29     # input lambda m d
30     l=int(input('lambda = '))
31     m=int(input('m = '))
32     d=int(input('d = '))
33     # window:  $\{-d*l+1, \dots, d*l\}$ 
34     window=list(range(d*l,-d*l,-1))
35     # cnt: counter for number of vertices
36     cnt=0
37     # V: vertex set
38     V=[]
39     # Enumerate vertices
40     for v in FixedSizeSubsets(window,d*l,-d*l):
41         V.append(frozenset(v))
42         cnt+=1
43
44     #  $x=(1,0,\dots,0)$ : a vector indexed by vertices
45     x=dict(zip(V,[1]+[0]*(cnt-1)))
46     # s: the start and the end
47     s=frozenset(range(d*l,0,-1))
48
49     # repeat  $x=Ax$  for m times
50     for i in range(m):
51         # y stores the intermediate result of  $Ax$ 
52         y=dict(zip(V,[0]*cnt))
53         for v in V:
54             P=[t-1 for t in v]+list(range(d*l,d*l-1,-1))
55             # enumerate edges

```



```

56         for Q in FixedSizeSubsets(P,l,-d*l):
57             u=frozenset ( set (P)-set (Q))
58             y [u]+=x [v]
59         # copy the result to x
60         x=y
61         # output the result
62         print ("V({},{},{})={}" . format (l , l*(i+1),d,x [s]))

```

