The Query Complexity of Mastermind with ℓ_p Distances

Manuel Fernández V
Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
manuelf@andrew.cmu.edu

David P. Woodruff
Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
dwoodruf@cs.cmu.edu

Taisuke Yasuda
Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
yasuda.taisuke1@gmail.com

Abstract

Consider a variant of the Mastermind game in which queries are ℓ_p distances, rather than the usual Hamming distance. That is, a codemaker chooses a hidden vector $\mathbf{y} \in \{-k, -k+1, \ldots, k-1, k\}^n$ and answers to queries of the form $\|\mathbf{y} - \mathbf{x}\|_p$ where $\mathbf{x} \in \{-k, -k+1, \ldots, k-1, k\}^n$. The goal is to minimize the number of queries made in order to correctly guess $\mathbf{y}$.

In this work, we show an upper bound of $O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right)$ queries for any real $1 \le p < \infty$ and $O(n)$ queries for $p = \infty$. To prove this result, we in fact develop a nonadaptive polynomial time algorithm that works for a natural class of separable distance measures, i.e., coordinate-wise sums of functions of the absolute value. We also show matching lower bounds up to constant factors, even for adaptive algorithms for the approximation version of the problem, in which the problem is to output $\mathbf{y}'$ such that $\|\mathbf{y}' - \mathbf{y}\|_p \le R$ for any $R \le k^{1-\varepsilon} n^{1/p}$ for constant $\varepsilon > 0$. Thus, essentially any approximation of this problem is as hard as finding the hidden vector exactly, up to constant factors. Finally, we show that for the noisy version of the problem, i.e., the setting when the codemaker answers queries with any $q = (1 \pm \varepsilon)\|\mathbf{y} - \mathbf{x}\|_p$, there is no query efficient algorithm.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Theory of computation → Design and analysis of algorithms

Keywords and phrases Mastermind, Query Complexity, ℓ_p Distance

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2019.1

Category APPROX

Funding *David P. Woodruff*: Part of this work was done while visiting Google as well as the Simons Institute for the Theory of Computing. D. Woodruff would also like to thank partial support from the Office of Naval Research (ONR) grant N00014-18-1-2562.

Acknowledgements We thank Flavio Chierichetti and Ravi Kumar for helpful discussions, as well as the anonymous reviewers for helpful feedback.

1 Introduction

Mastermind is a game played between two players, the *codemaker* and the *codebreaker*. In the 1970 original 4-position 6-color version of the game, the codemaker chooses 4 colored pegs, each taking one of 6 colors, and the codebreaker tries to guess the codemaker's 4 pegs by making queries to the codemaker by taking a guess at the sequence of the codemaker's 4 colored pegs. These guesses are answered by two numbers, the number of pegs guessed that are in the right position and the right color, indicated by black pegs, and the additional number of pegs of the right color but in the wrong position, indicated by white pegs.

© Manuel Fernández V, David P. Woodruff, and Taisuke Yasuda;
licensed under Creative Commons License CC-BY

Ever since, this game and its generalizations and variants have been studied by many computer scientists. The original version was completely characterized by [7], who showed upper and lower bounds of 5 queries for deterministic strategies. The n-position k-color generalization of the game was studied in [4], which sparked a line of research that lead to progressive improvement in upper and lower bounds for this problem, both in the original version of the game as well as in related variants of the game [2]. As these variants are not the focus of this work, we refer the reader to the expositions of [5, 2] for more details on this literature.

Note that in the variant that the codebreaker only receives the black peg answers, the problem can be phrased as guessing a hidden vector based on Hamming distance queries. One can then consider many variants of the Mastermind game in which the codebreaker guesses the codemaker's hidden vector based on other distance queries. For instance, motivated by the theory of black-box complexity, [1] recently studied the variant where the distance is the length of the longest common prefix with respect to an unknown permutation. In recreational mathematics, the ℓ_1 distance case has been studied under the name of "digit-distance" [6]. In this work, we study the case of ℓ_p distance queries. That is, the codemaker chooses a hidden vector $\mathbf{y} \in \{-k, -k+1, \dots, k-1, k\}^n$ and answers to queries of the form $\|\mathbf{y}-\mathbf{x}\|_p$ where $\mathbf{x} \in \{-k, -k+1, \dots, k-1, k\}^n$.

1.1 Our contributions

On the algorithmic side, we present Theorem 9, in which we develop a very general nonadaptive algorithm that works for any separable distance measure, i.e., the distance between $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ is given by $f(\mathbf{x}-\mathbf{y})$ where $f(\mathbf{x}) = \sum_{i=1}^n g_i(|x_i|)$. When we apply this to the case of $g_i(x) = x^p$ *for any constant real* $1 \le p < \infty$, i.e., when f is the (p-th power of the) ℓ_p norm, we obtain a polynomial time algorithm making $O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right)$ queries. For $p = \infty$, we give a simple algorithm achieving $O(n)$ queries. We also give lower bounds for any adaptive algorithm that match our upper bounds up to constant factors, *for any constant integer* $1 \le p < \infty$ (Theorem 11) and for $p = \infty$ (Theorem 12). In fact, our lower bounds are for a weaker problem, the problem of outputting an approximation $\mathbf{y}'$ such that its distance from the true hidden vector $\mathbf{y}$ is at most $\|\mathbf{y}'-\mathbf{y}\|_p \le R$, whenever the approximation radius satisfies $R \le k^{1-\varepsilon}n^{1/p}$ (where we think of $n^{1/p} = 1$ when $p = \infty$) for constant $\varepsilon > 0$. Thus, approximation for this problem is hard, in the sense that finding the point exactly is optimal up to constant factors, even when the approximation radius is as large as $k^{1-\varepsilon}n^{1/p}$.

Our main algorithmic technique for obtaining Theorem 9 is a judicious application of a generalization of the Fourier-based detecting matrix construction of [3]. Our lower bounds are simply obtained by counting the number of lattice points in an ℓ_p ball.

Finally, we consider a noisy version of the above problem, where the codemaker is allowed to answer queries with any answer that is within $(1 \pm \varepsilon)\|\mathbf{y}-\mathbf{x}\|_p$. For this variant, we show that any algorithm must make $\Omega(\exp(\varepsilon^2\Theta(k^p n)))$ queries in Theorem 13. That is, there is no query efficient algorithm for this problem.

2 Preliminaries

2.1 Notation

▶ **Definition 1** (ℓ_p norm). *Let* $1 \le p \le \infty$. *Then, we endow* $\mathbb{R}^n$ *with the* ℓ_p *norm* $\|\cdot\|_p$, *given by*

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^n |x_i|^p\right)^{1/p} \tag{1}$$

if $p < \infty$ and

$$\|\mathbf{x}\|_\infty := \max_{i=1}^{n} |x_i| \tag{2}$$

if $p = \infty$.

▶ **Definition 2** (Weight of binary vector). *Let $a \in \{0,1\}^\nu$. Then,* $\mathrm{wt}(a)$ *is the number of* 1*s in a.*

▶ **Definition 3** (Even-odd decomposition). *Let $h : \mathbb{R} \to \mathbb{R}$ be any function. Then, the* even-odd decomposition of h *is given by*

$$\begin{aligned} h_{\text{even}}(x) &:= \frac{h(x) + h(-x)}{2} \\ h_{\text{odd}}(x) &:= \frac{h(x) - h(-x)}{2}. \end{aligned} \tag{3}$$

It is easy to see that $h = h_{\text{even}} + h_{\text{odd}}$ and that $h_{\text{even}}(-x) = h_{\text{even}}(x)$ and $h_{\text{odd}}(-x) = -h_{\text{odd}}(x)$ for all $x \in \mathbb{R}$.

2.2 Bshouty's detecting matrix

We very briefly review the construction of the detecting matrix of [3], as we build off of this result for our algorithms.

▶ **Definition 4** (Detecting matrix [3]). *A $(d_1, d_2, \dots, d_n)$-detecting matrix is a $\{0,1\}$-matrix such that for every $\mathbf{u}, \mathbf{v} \in \prod_{i=1}^n \{0, 1, \dots, d_i - 1\}$ with $\mathbf{u} \neq \mathbf{v}$, we have $M\mathbf{u} \neq M\mathbf{v}$.*

The theorem we use is the following:

▶ **Theorem 5** (Bshouty detecting matrix, Theorem 4/Corollary 5 of [3]). *Let $1 < d_1 \le d_2 \le \dots \le d_n$ where $d_1 + d_2 + \dots + d_n = d$. There is a $(d_1, d_2, \dots, d_n)$-detecting matrix M of size $s \times n$ where*

$$s(\log s - 4) \le 2n \log \frac{d}{n}. \tag{4}$$

Furthermore, for $\mathbf{u} \in \prod_{i=1}^n \{0, 1, \dots, d_i - 1\}$, there is a polynomial time algorithm for recovering $\mathbf{u}$ given $M\mathbf{u}$.

We will only sketch the main idea behind the construction of the matrix and the decoding algorithm, and refer the reader to [3] for the proof of the bounds and the correctness.

2.2.1 Fourier representation [3]

We consider the Fourier basis on real-valued functions defined on the Boolean hypercube $\{-1,+1\}^\nu$, i.e., the basis

$$\mathcal{B} := \left\{ \chi_a(x) := \prod_{a_i = 1} x_i \,\middle|\, a \in \{0,1\}^\nu \right\} \subseteq \{f : \{-1,+1\}^\nu \to \mathbb{R}\}. \tag{5}$$

It is known that $\mathcal{B}$ is an orthonormal basis, and thus any $f : \{-1,+1\}^\nu \to \mathbb{R}$ can be uniquely represented as

$$f(x) = \sum_{a \in \{0,1\}^s} \hat{f}(a) \chi_a(x) \tag{6}$$

where $\hat{f}(a)$ is the Fourier coefficient of χ_a given by

$$\hat{f}(a) = \frac{1}{2^\nu} \sum_{x \in \{-1,+1\}^\nu} f(x)\chi_a(x). \tag{7}$$

Using the fast Fourier transform, all the coefficients $\hat{f}(a)$ can be found from the values of $f(x), x \in \{-1,+1\}^\nu$, and ordered according to lexicographic order of $a \in \{0,1\}^\nu$ in time $O(\nu 2^\nu)$.

2.2.2 Detecting matrix construction

The overall idea is as follows. We choose s as in equation (4) and $\nu := \log_2 s$. Then, we view column vectors in $\mathbb{R}^s$ with $s = 2^\nu$ rows as enumerations of the values of functions $f : \{-1,+1\}^\nu \to \mathbb{R}$. That is, for $x \in \{-1,+1\}^\nu$, the xth row of the column vector representing f is $f(x)$. We then view our detecting matrix $M \in \{0,1\}^{s\times n}$ as a family of n $\{0,1\}$-valued functions defined on $\{-1,+1\}^\nu$ and Mu as a linear combination of functions from this family, where the coefficients of the linear combination are specified by the unknown vector $\mathbf{u} \in \prod_{i=1}^n \{0,1,\dots,d_i-1\}$. The n functions of M have a special structure in the Fourier basis, so that there is an efficient iterative algorithm for recovering the coordinates of u in batches from the Fourier coefficients of the function Mu.

We iteratively construct columns of M as follows. For each $a \in \{0,1\}^\nu$, we will choose ℓ_a more columns to construct, so that in the end, we have $\sum_{a\in\{0,1\}^\nu} \ell_a = n$ columns.

Suppose that columns 1 through r have already been constructed. Let $a \in \{0,1\}^\nu$ and choose an integer ℓ_a such that

$$\begin{aligned} d_{r+1}d_{r+2}\dots d_{r+\ell_a} &\le 2^{\mathrm{wt}(a)} \\ d_{r+1}d_{r+2}\dots d_{r+\ell_a}d_{r+\ell_a+1} &> 2^{\mathrm{wt}(a)-1}. \end{aligned} \tag{8}$$

We then construct ℓ_a more columns of M so that the ith new function $g_{a,i}$ has Fourier coefficient of χ_a as

$$\hat{g}_{a,i}(a) = d_{r+1}d_{r+2}\dots d_{r+i}/2^{\mathrm{wt}(a)} \tag{9}$$

and the Fourier coefficient of χ_b for any $b > a$ (in the usual ordering on the Boolean hypercube) as

$$\hat{g}_{a,i}(b) = 0. \tag{10}$$

The way we choose the column functions $g_{a,i}$ to have these properties is described in [3].

2.2.3 Decoding algorithm

We now show how to efficiently decode $M\mathbf{u}$. Essentially, we will decode ℓ_a of the entries of $\mathbf{u}$ at a time, subtract them off, and recurse.

Note that column vector $M\mathbf{u}$ is the enumeration of the values of a linear combination f of the $g_{a,i}$ functions from above, where the row corresponding to $x \in \{-1,+1\}^\nu$ is $f(x)$. Then, using the fast Fourier transform, we find all the Fourier coefficients $\hat{f}(z)$ for $z \in \{0,1\}^\nu$ and search for a maximal $a \in \{0,1\}^\nu$ such that $\hat{f}(a) \neq 0$. For such an a, one can prove that its Fourier coefficient in f is

$$\hat{f}(a) = \frac{1}{2^{\mathrm{wt}(a)}}(\lambda_{r+1} + \lambda_{r+2}d_{r+1} + \lambda_{r+3}d_{r+1}d_{r+2} + \dots + \lambda_{r+\ell_a+1}d_{r+1}d_{r+2}\dots d_{r+\ell_a}) \tag{11}$$

where r is the number of columns in M before the columns corresponding to a, and $\lambda_{r+i} = u_{r+i}$ (for sake of matching the notation in [3]). Since $\lambda_{r+i} \in \{0, 1, \ldots, d_{r+i} - 1\}$ for all $i \in [\ell_a]$, we can recover all of the λ_{r+i}. Then, these coefficients can be subtracted off and we can recurse on the remaining entries of $\mathbf{u}$.

In our Theorem 9, we will modify the above algorithm to allow for non-integer values for the λ_{r+i}, as long as they are bounded and well-separated (to be made precise later).

3 Algorithms

We now describe our upper bounds. As a warm up, we start with algorithms for ℓ_1, ℓ_2, and ℓ_∞. These will introduce some tricks that we exploit in our coordinate-wise sums algorithm. Then, we combine these tricks along with a modification of the Bshouty detecting matrix algorithm described above to obtain Theorem 9.

3.1 Algorithms for ℓ_1, ℓ_2, and ℓ_∞

Our algorithms will be based around the idea of applying the Bshouty detecting matrix M to the hidden vector $\mathbf{y}$. This can be most straightforwardly applied in the case of ℓ_2, by expanding squared distances (equation (12)).

▶ **Theorem 6** (Algorithm for ℓ_2 queries). *Let $\mathbf{y} \in \{-k, -k+1, \ldots, k-1, k\}^n$ be an unknown vector, and suppose that we receive answers to s queries of the form $\|\mathbf{x} - \mathbf{y}\|_2$. Then, there is a polynomial time algorithm that recovers $\mathbf{y}$ in $s = O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right)$ queries.*

Proof. By first making the query with the $\mathbf{0}$ vector, we may find the norm $\|\mathbf{y}\|_2$ of the unknown vector. Now suppose we query for $\|\mathbf{x} - \mathbf{y}\|_2$. Note then that

$$\langle \mathbf{x}, \mathbf{y} \rangle = \frac{\|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - \|\mathbf{x} - \mathbf{y}\|_2^2}{2} \tag{12}$$

so we can compute the inner product between $\mathbf{x}$ and $\mathbf{y}$. Thus by taking n queries to be the n standard basis vectors $\mathbf{x} = \mathbf{e}_i$ for $i \in [n]$, we can always recover $\mathbf{y}$ in $n+1$ queries. To obtain $s = O\left(\frac{n \log k}{\log n}\right)$ for $k \leq n$, we can take our query vectors $\mathbf{x}$ to be the rows of the detecting matrix of Theorem 4/Corollary 5 of [3] and recover $\mathbf{y}$ by using the decoding algorithm as described in the proof. We thus conclude as desired. ◀

As shown above, if we can simulate computing inner products with binary vectors in $O(1)$ queries each, then we get an $O(n)$ algorithm by querying with the standard basis vectors or $O\left(\frac{n \log k}{\log n}\right)$ by using [3]. For ℓ_1, we take a similar approach. This time, the way we extract the inner product is quite different from the case of ℓ_2. This technique turns out to be much more flexible, and will allow us to generalize the result to coordinate-wise sums.

▶ **Theorem 7** (Algorithm for ℓ_1 queries). *Let $\mathbf{y} \in \{-k, -k+1, \ldots, k-1, k\}^n$ be an unknown vector, and suppose that we receive answers to s queries of the form $\|\mathbf{x} - \mathbf{y}\|_1$. Then, there is a polynomial time algorithm that recovers $\mathbf{y}$ in $s = O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right)$ queries.*

Proof. We will just show how to compute inner products in $O(1)$ queries, since the rest follows as in the ℓ_2 case. Let $\boldsymbol{\tau} \in \{0, 1\}^n$ be any binary vector and consider the sign vector $\boldsymbol{\sigma} \in \{\pm 1\}^n$ with $\sigma_i = (-1)^{\tau_i + 1}$. Then for $\sigma_i \in \{\pm 1\}$ and $-k \leq y_i \leq k$, we have that

$$|k\sigma_i - y_i| = \left|k\sigma_i - \sigma_i^2 y_i\right| = |k - \sigma_i y_i| = k - \sigma_i y_i. \tag{13}$$

Thus,

$$\|k\boldsymbol{\sigma} - \mathbf{y}\|_1 = \sum_{i=1}^{n} |k\sigma_i - y_i| = \sum_{i=1}^{n} k - \sigma_i y_i = kn - \boldsymbol{\sigma} \cdot \mathbf{y} \tag{14}$$

so we may compute the quantity $\boldsymbol{\sigma} \cdot \mathbf{y} = kn - \|k\boldsymbol{\sigma} - \mathbf{y}\|_1$. We may then compute the desired inner product with binary vectors as $\boldsymbol{\tau} \cdot \mathbf{y} = (\boldsymbol{\sigma} \cdot \mathbf{y} + \mathbf{1}_n \cdot \mathbf{y})/2$. ◀

To conclude the section, we show an $O(n)$ algorithm for ℓ_∞ queries. This turns out to be optimal, as we show later.

▶ **Theorem 8** (Algorithm for ℓ_∞ queries). *Let $\mathbf{y} \in \{-k, -k+1, \ldots, k-1, k\}^n$ be an unknown vector, and suppose that we receive answers to s queries of the form $\|\mathbf{x} - \mathbf{y}\|_\infty$. Then, there is a polynomial time algorithm that recovers $\mathbf{y}$ in $s = O(n)$ queries.*

Proof. For each $i \in [n]$, we make the query $q_i^+ = \|k\mathbf{e}_i - \mathbf{y}\|_\infty$ and $q_i^- = \|-k\mathbf{e}_i - \mathbf{y}\|_\infty$. Note that $y_i = 0$ if and only if these two are both equal to k. If $y_i > 0$, then $q_i^- = k + y_i > k$ and if $y_i < 0$, then $q_i^+ = k - y_i > k$. Thus, with these two queries, we can determine y_i. Thus, we recover $\mathbf{y}$ in $O(n)$ queries. ◀

3.2 Algorithm for coordinate-wise sums

In the previous section, we obtained polynomial time algorithms with tight query complexity for ℓ_1 and ℓ_2 by simulating inner product computations between $\mathbf{y}$ and binary vectors. We now generalize these ideas to an algorithm for any query given by sums along the coordinates. This in particular includes all (p-th powers of) ℓ_p norms, even for p not an integer.

▶ **Theorem 9** (Algorithm for coordinate-wise sums). *Let $\mathbf{y} \in \{-k, -k+1, \ldots, k-1, k\}^n$ be an unknown vector, and suppose that we receive answers to s queries of the form $f(\mathbf{y} - \mathbf{x})$, where $f(\mathbf{x}) = \sum_{i=1}^{n} g_i(|x_i|)$. For each $i \in [n]$, define the function $h_i(x) = g_i(k - x)$ and consider the even-odd decomposition $h_i = (h_i)_{\mathrm{even}} + (h_i)_{\mathrm{odd}}$ (see Definition 3). Also consider the following quantities:*

$$\begin{aligned}
M_i^{\min} &:= \min_{x \in \{-k, -k+1, \ldots, k-1, k\}} (h_i)_{\mathrm{odd}}(x) \\
M_i^{\max} &:= \max_{x \in \{-k, -k+1, \ldots, k-1, k\}} (h_i)_{\mathrm{odd}}(x) \\
\Delta_i &:= \min_{\substack{x_1, x_2 \in \{-k, -k+1, \ldots, k-1, k\} \\ x_1 \neq x_2}} |(h_i)_{\mathrm{odd}}(x_1) - (h_i)_{\mathrm{odd}}(x_2)| \\
\Delta &:= \min_{i=1}^{n} \Delta_i \\
d_i &:= \left\lceil \frac{M_i^{\max} - M_i^{\min}}{\Delta} \right\rceil + 1
\end{aligned} \tag{15}$$

If $\Delta > 0$, then there is a polynomial time algorithm that recovers $\mathbf{y}$ with $s = O\left(\min\left\{n, \frac{\log \prod_{i=1}^{n} d_i}{\log n}\right\}\right)$ queries.

Proof. Let $\mathbf{h}_{\mathrm{even}}$ and $\mathbf{h}_{\mathrm{odd}}$ be the functions that apply $(h_i)_{\mathrm{even}}$ and $(h_i)_{\mathrm{odd}}$ on the ith coordinate, respectively. We will show that we can recover $\mathbf{h}_{\mathrm{odd}}(\mathbf{y})$ in $O\left(\min\left\{n, \frac{\log \prod_{i=1}^{n} d_i}{\log n}\right\}\right)$ queries. Note that since $\min_{i=1}^{n} \Delta_i > 0$, $(h_i)_{\mathrm{odd}}$ is injective for each i and thus we can recover $\mathbf{y}$ from $\mathbf{h}_{\mathrm{odd}}(\mathbf{y})$ in polynomial time using a lookup table for the values of $(h_i)_{\mathrm{odd}}$.

3.2.1 Inner products with binary vectors

We first show that we can compute the inner product between $\mathbf{h}_{\text{odd}}(\mathbf{y})$ and any binary vector $\boldsymbol{\tau} \in \{0,1\}^n$. To do this, consider the sign vector $\boldsymbol{\sigma} \in \{\pm 1\}^n$ with $\sigma_i = (-1)^{\tau_i+1}$. Note that for $\sigma_i \in \{\pm 1\}$ and $-k \leq y_i \leq k$, we have $|k\sigma_i - y_i| = |k - \sigma_i y_i| = k - \sigma_i y_i$. Then, by querying vectors of the form $\mathbf{x} = k\boldsymbol{\sigma}$, we obtain

$$f(k\boldsymbol{\sigma} - \mathbf{y}) = \sum_{i=1}^n g_i(k - \sigma_i y_i) = \sum_{i=1}^n h_i(\sigma_i y_i). \tag{16}$$

Then using the even/oddness of $(h_i)_{\text{even}}/(h_i)_{\text{odd}}$, we have

$$\sum_{i=1}^n h_i(\sigma_i y_i) = \left(\sum_{i=1}^n (h_i)_{\text{even}}(y_i)\right) + \left(\sum_{i=1}^n \sigma_i (h_i)_{\text{odd}}(y_i)\right) = \mathbf{1}_n \cdot \mathbf{h}_{\text{even}}(\mathbf{y}) + \boldsymbol{\sigma} \cdot \mathbf{h}_{\text{odd}}(\mathbf{y}). \tag{17}$$

Note also that by querying for $k\mathbf{1}_n$ and $-k\mathbf{1}_n$, we also obtain

$$\begin{aligned} \frac{f(k\mathbf{1}_n - \mathbf{y}) + f(-k\mathbf{1}_n - \mathbf{y})}{2} &= \sum_{i=1}^n (h_i)_{\text{even}}(y_i) = \mathbf{1}_n \cdot \mathbf{h}_{\text{even}}(\mathbf{y}) \\ \frac{f(k\mathbf{1}_n - \mathbf{y}) - f(-k\mathbf{1}_n - \mathbf{y})}{2} &= \sum_{i=1}^n (h_i)_{\text{odd}}(y_i) = \mathbf{1}_n \cdot \mathbf{h}_{\text{odd}}(\mathbf{y}). \end{aligned} \tag{18}$$

Using these, we may compute $\boldsymbol{\tau} \cdot \mathbf{h}_{\text{odd}}(\mathbf{y}) = \frac{1}{2}(\boldsymbol{\sigma} + \mathbf{1}_n) \cdot \mathbf{h}_{\text{odd}}(\mathbf{y})$ and thus we are able to compute dot products of arbitrary binary vectors with $\mathbf{h}_{\text{odd}}(\mathbf{y})$. At this point, we can obtain $O(n)$ queries just by taking the binary vectors to be the standard basis vectors, so we focus on obtaining an algorithm making at most $O\left(\frac{\log \prod_{i=1}^n d_i}{\log n}\right)$ queries.

3.2.2 Modification of the Bshouty detecting matrix decoding [3]

Recall the detecting matrix of [3] for integer vectors in $\prod_{i=1}^n \{0, 1, \ldots, d_i - 1\}$ for $d_i \in \mathbb{N}$ for $i \in [n]$. If $\mathbf{h}_{\text{odd}}(\mathbf{y})$ took integer values, then we could just directly use this theorem to conclude with the desired query complexity. However, this is not true of $\mathbf{h}_{\text{odd}}(\mathbf{y})$, and so we need to show how to modify the [3] construction to handle our setting.

We first shift and scale our vector $\mathbf{h}_{\text{odd}}(\mathbf{y})$. Let $\mathbf{M}^{\min}$ be the vector with $M_i^{\min}$ in the ith coordinate. Note that we can easily compute $\boldsymbol{\tau} \cdot \mathbf{M}^{\min}$. Thus, we are able to compute dot products of arbitrary binary vectors with the vector $(\mathbf{h}_{\text{odd}}(\mathbf{y}) - \mathbf{M}^{\min})$. By dividing by Δ, we have dot products of arbitrary binary vectors with $\frac{1}{\Delta}(\mathbf{h}_{\text{odd}}(\mathbf{y}) - \mathbf{M}^{\min})$. We now define this as

$$\begin{aligned} \varphi_i(y) &:= \frac{1}{\Delta}\left((h_i)_{\text{odd}}(y) - M_i^{\min}\right) \\ \boldsymbol{\varphi}(\mathbf{y}) &:= \frac{1}{\Delta}\left(\mathbf{h}_{\text{odd}}(\mathbf{y}) - \mathbf{M}^{\min}\right) \end{aligned} \tag{19}$$

Note then that $0 \leq \varphi_i \leq d_i - 1$ (see equation (15)) and that $y_1 \neq y_2 \implies |\varphi(y_1) - \varphi(y_2)| \geq 1$.

Now consider the detecting matrix construction of Theorem 4 in [3]. Recall that we may extract the Fourier coefficient of χ_a for some maximal a in our unknown vector $\boldsymbol{\varphi}(\mathbf{y})$ viewed as a function, which gives us

$$\lambda_{r+1} + \lambda_{r+2} d_{r+1} + \lambda_{r+3} d_{r+1} d_{r+2} + \cdots + \lambda_{r+\ell_a+1} d_{r+1} d_{r+2} \ldots d_{r+\ell_a} \tag{20}$$

which in our case we set $\lambda_j = \varphi_j(y_j)$. Now let $\mathcal{X} \coloneqq \prod_{j=r+1}^{r+\ell_a+1} \varphi_j(\{-k, -k+1, \dots, k-1, k\})$ be the image of our original points in a subset of $\ell_a + 1$ coordinates starting at $r+1$ under the corresponding φ_j. Consider the function $\psi : \mathcal{X} \to \mathbb{R}^+$ defined via

$$\psi(\mathbf{z}) = \sum_{i=0}^{\ell_a} z_{i+1} \prod_{j=1}^{i} d_{r+j}. \tag{21}$$

It is easy to see that when we endow $\mathcal{X}$ with the lexicographical ordering, then ψ is increasing. Thus, given the Fourier coefficient as in equation (20), we can do binary search on the at most k^n values in $\mathcal{X}$ to extract the values λ_{r+i} in time $O(n \log k)$. Given this step of recovering ℓ_a of the coordinates, we can proceed as in the rest of [3] by subtracting these coordinates of the unknown vector and recursing. Hence, we conclude that we may recover $\boldsymbol{\varphi}(\mathbf{y})$ efficiently and thus $\mathbf{h}(\mathbf{y})$, as claimed. ◀

3.2.3 Reconstruction with ℓ_p queries

As a corollary of the above result, we obtain an algorithm for recovering $\mathbf{y} \in \{-k, -k+1, \dots, k-1, k\}^n$ from $s = O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right)$ distance queries in ℓ_p.

▶ **Corollary 10** (Algorithm for ℓ_p queries). *Let $\mathbf{y} \in \{-k, -k+1, \dots, k-1, k\}^n$ be an unknown vector, and suppose that we receive answers to s queries of the form $\|\mathbf{x} - \mathbf{y}\|_p$ for p a constant. Then, there is a polynomial time algorithm that recovers $\mathbf{y}$ in $s = O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right)$ queries.*

Proof. We are in the setting to use Theorem 9, with $g_i(x) = x^p$ for all $i \in [n]$ and $h_{\text{odd}}(x) = \frac{1}{2}((k-x)^p - (k+x)^p)$. Recall that we have efficient algorithms with the desired guarantees when $p \in \{1, 2\}$ so we dismiss these cases. In the remaining range of p, we just need to compute $\prod_{i=1}^n d_i$.

Note that

$$h'_{\text{odd}}(x) = -\frac{p}{2}\left((k-x)^{p-1} + (k+x)^{p-1}\right) < 0 \tag{22}$$

on $x \in [-k, k]$ so h_{odd} is decreasing on this interval. Then, $(2k)^p/2 = h_{\text{odd}}(-k) \geq h_{\text{odd}}(x) \geq h_{\text{odd}}(k) = -(2k)^p/2$. Furthermore, note that

$$h''_{\text{odd}}(x) = \frac{p(p-1)}{2}\left((k-x)^{p-2} - (k+x)^{p-2}\right). \tag{23}$$

If $p > 2$, then this is negative on $x \geq 0$, so $|h'_{\text{odd}}(x)|$ is smallest at $x = 0$ and thus $|h'_{\text{odd}}(x)| \geq |h'_{\text{odd}}(0)| = pk^{p-1}$ for all x. If $1 < p < 2$, then this is positive on $x \geq 0$, so $|h'_{\text{odd}}(x)|$ is smallest at $x = k$ and thus $|h'_{\text{odd}}(x)| \geq |h'_{\text{odd}}(k)| = (p/2)(2k)^{p-1}$ for all x. In either case, we have that $\Delta = \Omega(pk^{p-1})$ and the range is $M^{\max} - M^{\min} = O(k^p)$ and thus $d_i = O((M^{\max} - M^{\min})/\Delta) = O(k)$. Thus, the query complexity is

$$O\left(\min\left\{n, \frac{\log \prod_{i=1}^n d_i}{\log n}\right\}\right) = O\left(\min\left\{n, \frac{n \log k}{\log n}\right\}\right) \tag{24}$$

as desired. ◀

4 Lower Bounds

In this section, we complement our algorithms with matching lower bounds, for integer p. Our lower bounds work even for the problem of approximating the hidden vector and for adaptive randomized algorithms with constant success probability.

▶ **Theorem 11** (Lower bound for integer ℓ_p)**.** *Let $1 \le p < \infty$ be a constant integer and let $R \in (0, kn^{1/p}]$ be an approximation radius. Suppose there exists an algorithm $\mathcal{A}$ such that for all unknown vectors $\mathbf{y} \in \{-k, -k+1, \dots, k-1, k\}^n$, $\mathcal{A}$ outputs a vector $\mathbf{y}' \in \{-k, -k+1, \dots, k-1, k\}^n$ such that*

$$\|\mathbf{y}' - \mathbf{y}\|_p \le R \tag{25}$$

in s possibly adaptive ℓ_p queries with probability at least $2/3$ over the algorithm's random coin tosses. Then

$$s = \Omega\left(\frac{n \log(kn^{1/p}/R)}{\log k + \log n}\right). \tag{26}$$

In particular, if $R \le k^{1-\varepsilon} n^{1/p}$ for some constant $\varepsilon > 0$, then

$$s = \Omega\left(\frac{n \log k}{\log k + \log n}\right), \tag{27}$$

which is $\Omega\left(\frac{n \log k}{\log n}\right)$ if $k < n$ and $\Omega(n)$ if $k \ge n$.

Proof. By Yao's minimax principle [9], it suffices to show the lower bound for all deterministic algorithms $\mathcal{A}$ that correctly approximates a uniformly random $\mathbf{y} \in \{-k, -k+1, \dots, k-1, k\}^n$ with probability at least $2/3$.

Note that each query $\|\mathbf{x} - \mathbf{y}\|_p^p$ results in a nonnegative integer that is at most $(2k)^p n$. Thus, there are at most $((2k)^p n + 1)^s$ possible sequences of answers. Now let Q be the set of all sequence of answers that $\mathcal{A}$ can observe, and for each sequence of answers $\mathbf{q} \in Q$, let $S_{\mathbf{q}}$ denote the set of vectors $\mathbf{y} \in \{-k, -k+1, \dots, k-1, k\}^n$ such that the deterministic algorithm $\mathcal{A}$ observes $\mathbf{q}$ on input $\mathbf{y}$. Then, $S_{\mathbf{q}}$ partitions the unknown vectors $\mathbf{y}$ into $|Q|$ disjoint sets. Then, the probability that $|S_{\mathbf{q}}|$ has size at most $\frac{1}{100}\frac{(2k+1)^n}{|Q|}$ is

$$\begin{aligned}
\Pr_{\mathbf{y}}\left(|S_{\mathbf{q}}| \le \frac{1}{100}\frac{(2k+1)^n}{|Q|}\right) &= \sum_{\substack{\mathbf{q} \in Q \\ |S_{\mathbf{q}}| \le \frac{1}{100}\frac{(2k+1)^n}{|Q|}}} \Pr_{\mathbf{y}}(\mathcal{A} \text{ queries the sequence } \mathbf{q}) \\
&= \sum_{\substack{\mathbf{q} \in Q \\ |S_{\mathbf{q}}| \le \frac{1}{100}\frac{(2k+1)^n}{|Q|}}} \frac{|S_{\mathbf{q}}|}{(2k+1)^n} \\
&\le \sum_{\substack{\mathbf{q} \in Q \\ |S_{\mathbf{q}}| \le \frac{1}{100}\frac{(2k+1)^n}{|Q|}}} \frac{1}{100}\frac{(2k+1)^n}{|Q|}\frac{1}{(2k+1)^n} \\
&\le \sum_{\mathbf{q} \in Q} \frac{1}{100|Q|} = \frac{1}{100}.
\end{aligned} \tag{28}$$

Thus with probability at least $99/100$, $|S_{\mathbf{q}}|$ has size at least $\frac{1}{100}\frac{(2k+1)^n}{|Q|}$.

Note that by [8], the volume of a unit ℓ_p ball is $2^n \Gamma(1+1/p)^n / \Gamma(1+n/p)$, so the volume of a ball of radius R in ℓ_p is

$$V := R^n 2^n \frac{\Gamma(1+1/p)^n}{\Gamma(1+n/p)} = \left(\Theta\left(\frac{R}{n^{1/p}}\right)\right)^n. \tag{29}$$

Now suppose that $\mathbf{q}$ is a sequence of queries such that $|S_{\mathbf{q}}| > 2V$ and let $\mathbf{z}$ be the output of the deterministic algorithm $\mathcal{A}$ on the sequence of queries $\mathbf{q}$. Then, at most V of the points in S can be in the ℓ_p ball of radius R centered at $\mathbf{z}$. Thus, with probability at least $1/2$ over the random hidden vector $\mathbf{y}$, we output a point $\mathbf{z}$ such that $\|\mathbf{z}-\mathbf{y}\|_p \geq R$. Thus, if

$$\frac{1}{100}\frac{(2k+1)^n}{|Q|} > 2V, \tag{30}$$

then our probability of success is at most $1/2+1/100$ and thus we do not have a correct algorithm. Thus, it must be that

$$\frac{1}{100}\frac{(2k+1)^n}{|Q|} \leq 2V \implies \frac{(2k+1)^n}{200V} \leq |Q| \leq ((2k)^p n+1)^s. \tag{31}$$

Rearranging, we have that

$$s \geq \frac{\log\frac{(2k+1)^n}{200V}}{\log((2k)^p n+1)} = \Omega\left(\frac{n\log(kn^{1/p}/R)}{\log k+\log n}\right), \tag{32}$$

as claimed. ◀

For $p=\infty$, we have a lower bound of $\Omega(n)$ regardless of k.

▶ **Theorem 12** (Lower bound for ℓ_∞). *Let $R \in (0,k]$ be an approximation radius. Suppose there exists an algorithm $\mathcal{A}$ such that for all unknown vectors $\mathbf{y} \in \{-k,-k+1,\ldots,k-1,k\}^n$, $\mathcal{A}$ outputs a vector $\mathbf{y}' \in \{-k,-k+1,\ldots,k-1,k\}^n$ such that*

$$\|\mathbf{y}'-\mathbf{y}\|_\infty \leq R \tag{33}$$

in s possibly adaptive ℓ_∞ queries with probability at least $2/3$ over the algorithm's random coin tosses. Then

$$s = \Omega\left(\frac{n\log(k/R)}{\log k}\right). \tag{34}$$

In particular, if $R \leq k^{1-\varepsilon}$ for a constant $\varepsilon > 0$, then $s = \Omega(n)$.

Proof. By the same argument as the finite ℓ_p case, we use Yao's minimax principle to reduce the argument to a lower bound for all deterministic algorithms $\mathcal{A}$ on uniformly random inputs $\mathbf{y}$ succeeding with probability at least $2/3$. Furthermore, by the same partition argument as before, we have that $|S_{\mathbf{q}}|$ is at least $\frac{1}{100}\frac{(2k+1)^n}{|Q|}$ with probability at least $99/100$.

The volume of an ℓ_∞ ball of radius R is $(2R)^n$, so as before, we must have

$$\frac{1}{100}\frac{(2k+1)^n}{|Q|} \leq 2(2R)^n. \tag{35}$$

When $p=\infty$, there are only $(2k+1)^s$ possible sequences of answers, so we instead have the bound

$$\frac{(2k+1)^n}{(2R)^n} \leq 200(2k+1)^s \tag{36}$$

By rearranging, we obtain the bound $s = \Omega\left(\frac{n\log(k/R)}{\log k}\right)$ as desired. ◀

4.1 Lower bound for the noisy problem

Finally, we show that in the noisy version of the problem, i.e., the setting where the codemaker is allowed to answer the queries $\mathbf{x}$ with any $q = (1 \pm \varepsilon)\|\mathbf{y} - \mathbf{x}\|_p$, there is no good algorithm.

▶ **Theorem 13** (Lower bound for the noisy problem). *Let $1 \le p < \infty$ be a constant and let $0 < R < kn^{1/p}$ be an approximation radius. Suppose there exists an algorithm $\mathcal{A}$ such that for all unknown vectors $\mathbf{y} \in \{-k, -k+1, \ldots, k-1, k\}^n$, $\mathcal{A}$ outputs a vector $\mathbf{y}' \in \{-k, -k+1, \ldots, k-1, k\}^n$ such that*

$$\|\mathbf{y}' - \mathbf{y}\|_p \le R \tag{37}$$

in s possibly adaptive $(1 \pm \varepsilon)$-noisy ℓ_p queries, i.e., answers with adversarially chosen $q_{\mathbf{x}} = (1 \pm \varepsilon)\|\mathbf{y} - \mathbf{x}\|_p$, with probability at least $2/3$ over the algorithm's random coin tosses. Then

$$s = \Omega\big(\exp\big(\varepsilon^2 \Theta(k^p n)\big)\big). \tag{38}$$

Proof. By Yao's minimax principle, we can take the algorithm to be deterministic by taking our hidden vector $\mathbf{y}$ to be drawn uniformly from $\{-k, -k+1, \ldots, k-1, k\}^n$. Now fix any query $\mathbf{x} \in \{-k, -k+1, \ldots, k-1, k\}^n$ and let $\mu = \mathbf{E}_{\mathbf{z}}\left(\|\mathbf{x} - \mathbf{z}\|_p^p\right) = \Theta(k^p n)$. Then by Chernoff bounds,

$$\Pr_{\mathbf{y}}\left(\left|\|\mathbf{x} - \mathbf{y}\|_p^p - \mu\right| \ge \varepsilon\mu\right) \le 2\exp\big(-\varepsilon^2\mu\big). \tag{39}$$

Thus, if the number of queries s is less than $\exp\big(\varepsilon^2\Theta(k^p n)\big)/200$, then by the union bound over the s queries, with probability at least $99/100$ over the choice of $\mathbf{y}$, the codemaker can just return $\mathbf{E}_{\mathbf{z}}\left(\|\mathbf{x} - \mathbf{z}\|_p^p\right)$ for any query $\mathbf{x}$. Thus, the deterministic codebreaker algorithm sees the same sequence of answers with probability at least $99/100$ and so the algorithm cannot be correct. Hence, we conclude that $s = \Omega(\exp(\varepsilon^2\Theta(k^p n)))$. ◀

References

1 Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of a permutation-based variant of Mastermind. *Discrete Applied Mathematics*, 2019.

2 Aaron Berger, Christopher Chute, and Matthew Stone. Query complexity of mastermind variants. *Discrete Mathematics*, 341(3):665–671, 2018.

3 Nader H. Bshouty. Optimal Algorithms for the Coin Weighing Problem with a Spring Scale. In *COLT 2009 - The 22nd Conference on Learning Theory, Montreal, Quebec, Canada, June 18-21, 2009*, 2009. URL: `http://www.cs.mcgill.ca/%7Ecolt2009/papers/004.pdf#page=1`.

4 Vasek Chvátal. Mastermind. *Combinatorica*, 3(3):325–329, 1983. `doi:10.1007/BF02579188`.

5 Benjamin Doerr, Carola Doerr, Reto Spöhel, and Henning Thomas. Playing Mastermind With Many Colors. *J. ACM*, 63(5):42:1–42:23, 2016. `doi:10.1145/2987372`.

6 David Ginat. Digit-distance Mastermind. *The Mathematical Gazette*, 86(507):437–442, 2002.

7 Donald E. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 9:1–6, 1977.

8 Xianfu Wang. Volumes of generalized unit balls. *Mathematics Magazine*, 78(5):390–395, 2005.

9 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227. IEEE, 1977.