Query Complexity of Mastermind Variants

Aaron Berger Christopher Chute Matthew Stone

January 28, 2016

Abstract

We study variants of Mastermind, a popular board game in which the objective is sequence reconstruction. In this two-player game, the so-called *codemaker* constructs a hidden sequence $H = (h_1, h_2, \ldots, h_n)$ of colors selected from an alphabet $\mathcal{A} = \{1, 2, \ldots, k\}$ (*i.e.,* $h_i \in \mathcal{A}$ for all $i \in \{1, 2, \ldots, n\}$). The game then proceeds in turns, each of which consists of two parts: in turn t, the second player (the *codebreaker*) first submits a query sequence $Q_t = (q_1, q_2, \ldots, q_n)$ with $q_i \in \mathcal{A}$ for all i, and second receives feedback $\Delta(Q_t, H)$, where Δ is some agreed-upon function of distance between two sequences with n components. The game terminates when $Q_t = H$, and the codebreaker seeks to end the game in as few turns as possible. Throughout we let $f(n, k)$ denote the smallest integer such that the codebreaker can determine any H in $f(n, k)$ turns. We prove three main results: First, when H is known to be a permutation of $\{1, 2, \ldots, n\}$, we prove that $f(n, n) \geq n - \log\log(n)$ for all sufficiently large n. Second, we show that Knuth's Minimax algorithm identifies any H in at most nk queries. Third, when feedback is not received until all queries have been submitted, we show that $f(n, k) = \Omega(n \log k)$.

1 Introduction

Mastermind is a game created by Mordechai Meirowitz in 1970. In the original game, the codebreaker's objective is to guess a hidden sequence of 4 colors, each chosen from a pool of 6. Knuth analyzed the original game in 1977, and showed that there exists a strategy that guarantees guessing the hidden vector in no more than 5 turns [10]. Variants of Mastermind are a family of two-player games centered around reconstruction of a hidden sequence. In all variants of the game, one player is given the role of *codemaker*, and the other is denoted the *codebreaker*. The codemaker begins the game by constructing a hidden sequence $H = (h_1, h_2, \ldots, h_n)$ where each component is selected from an alphabet $\mathcal{A} = \{1, 2, \ldots, k\}$ of k colors (that is, $h_i \in \mathcal{A}$ for all $i \in \{1, 2, \ldots, n\}$). The goal of the codebreaker is to uniquely determine the hidden sequence H through a series of queries, which are submissions of vectors of the form $Q_t = (q_1, q_2, \ldots, q_n)$. The codebreaker always seeks to determine H with as few queries as possible, however the nature of these queries, the feedback received after a query, and the restrictions on H differ between variants. Erdos and Rényi, for example, studied a 2-color version of the game, before Mastermind existed, in [5].

The variants of Mastermind which we will study are defined by settings of the tuple (n, k, Δ, R, A). These parameters are defined as follows:

(i) (n) *Length of Sequence.* The parameter n denotes the length of the hidden sequence H created by the codemaker, hence $H = (h_1, h_2, \ldots, h_n)$. The codebreaker is also required to submit query vectors of length n, so the t^{th} query vector takes the form $Q_t = (q_1, q_2, \ldots, q_n)$.

(ii) (k) *Size of Alphabet.* This parameter determines the number of possible values for components of H and Q_t. Associated with each game is an alphabet $\mathcal{A} = \{1, 2, \ldots, k\}$ from which the components of H and Q_t are selected. That is, $h_i \in \{1, 2, \ldots, k\}$ and $q_i \in \{1, 2, \ldots, k\}$.

(iii) (Δ) *Distance Function.* Each variant of Mastermind has an associated distance function Δ, which is employed as follows. For each query sequence $Q_t = (q_1, q_2, \ldots, q_n)$ submitted by the codebreaker, the codemaker gives feedback $\Delta(Q_t, H)$, where Δ is the agreed-upon function of distance between two sequences of length n. The information yielded by $\Delta(Q_t, H)$ clearly influences the number of queries needed to identify the hidden vector H. For example, if Δ is defined by $\Delta(Q_t, H) = H$ then trivially the codebreaker can determine H in one query, or if we define $\Delta(Q_t, H) = 0$ then all sequences of length n must be queried to guarantee a winning query. We study the following distance functions:

a. *"Black-peg and white-peg."* Let Q_t and H be as above. The black-peg and white-peg distance function is defined by $\Delta(Q_t, H) = (b(Q_t, H), w(Q_t, H))$ where

$$b(Q_t, H) = |\{i \in [1, n] \mid q_i = h_i\}|, \tag{1}$$

and

$$w(Q_t, H) = \max_{\sigma} \; b(\sigma(Q_t), H) - b(Q_t, H),$$

where σ iterates over all permutations of Q_t. We note that this is the distance function used in the original game of Mastermind.

b. *"Black-peg-only."* When Δ is the black-peg-only distance function, it is defined by $\Delta(Q_t, H) = b(Q_t, H)$, where b is defined as in equation (1).

We will denote the black-peg-only distance function by $\Delta = b$, and the black-white distance function by $\Delta = bw$.

(iv) (R) *Repetition.* The parameter R is a Boolean restriction on the components of H. If R is true, we say that the variant game is *with repeats* or that repeats are allowed. In this case, the hidden vector H may have repeated colors, that is, we allow $h_i = h_j$ for any $i, j \in \{1, 2, \ldots, n\}$. When R is false, we say that the variant is *no repeats*, and we require $h_i \neq h_j$ when $i \neq j$, and so we necessarily require $k \geq n$. In particular, when R is false and $k = n$, we have that H must be a permutation of $(1, 2, \ldots, n)$, and we refer to this variant as the *Permutation Game.*

(v) (A) *Adaptiveness.* The Boolean parameter A determines whether the codebreaker receives feedback after each query. If A is true, we say that the game is *adaptive*. In this case the game consists of two-part turns: on the t^{th} turn, the codebreaker first submits a query sequence Q_t, and then receives feedback $\Delta(Q_t, H)$. The codebreaker may use the feedback to inform the query Q_{t+1} made in turn $t + 1$, and the game ends in turn s if and only if $Q_s = H$.

When A is false, we say that the game is *non-adaptive*. In this case the codebreaker submits m queries $Q_1, Q_2, \ldots, Q_m$ all at once (where the codebreaker chooses m). The codemaker then reports a feedback vector of the form $(\Delta(Q_1, H), \Delta(Q_2, H), \ldots, \Delta(Q_m, H))$, after which the codebreaker must submit the final query $\overline{Q}$. The codebreaker wins if and only if $\overline{Q} = H$.

Throughout we define $f(n, k, \Delta, R, A)$ to be the smallest integer such that the codebreaker can determine any hidden sequence H in $f(n, k, \Delta, R, A)$ queries during a game with the corresponding assignment of n, k, Δ, R, and A. The case $R = T, A = T$ is the most extensively studied in literature [10, 3, 2, 7]. Doerr, Spöhel, Thomas, and Winzen obtain many significant results in the case $R = T$ in [4], applying techniques from [8, 1]. We will denote true and false by T and F, respectively, for assignment of Boolean variables. For example, Donald Knuth's result that the original game of Mastermind (four positions, six colors, black-peg and white-peg, with repeats, and adaptive) can always be determined

after four turns (and then guessed on the fifth turn) is equivalently stated as $f(4, 6, \Delta = (b, w), R = T, A = T) \leq 4$ (in fact, this bound holds with equality) [10]. We will write simply $f(n, k)$ when the context is clear.

We focus only on analyzing the worst-case performance of query strategies for these variants of Mastermind. That is, we always consider the number of queries necessary to guarantee identification of any hidden vector. We prove three main results, each of which builds on the previous work related to a given Mastermind variant.

Our first main result concerns the Permutation Game, in which $n = k$ and $R = F$, thus restricting the hidden sequence H to be a permutation of the alphabet $\mathcal{A}$.

Theorem 1. *Consider the Permutation Game defined by $n = k$ and $R = F$. Let $\Delta = b$ (black-peg-only distance function) and let A be fixed. Then for all sufficiently large n, we have*

$$f(n, k = n, \Delta = b, R = F, A) \geq n - \log\log(n).$$

Explicit algorithms that take $O(n \log(n))$ turns to solve this variant were developed by Ko and Teng in [11], and Ouali and Sutherland in [12]. Ko and Teng approach the problem with an algorithm akin to binary search. The algorithm queries a sequence Q_t, swaps two components, and queries again, doing so repeatedly until a previously unknown component of H is determined by the values of the distance function across these repeated queries. Ouali and Sutherland improve this algorithm primarily by altering the search routine after a single component of H has been identified. They also extend their results to variants with $k \geq n$. In this way, they achieve an average factor of two reduction in the number of queries needed to identify H. In our notation, these results state that $f(n, k = n, \Delta = b, R = F, A = T) \leq O(n \log(n))$.

Via a basic information-theoretic argument, one can show that the Permutation Game also satisfies $f(n, n) \geq n - n/\log(n) + c$ for some constant $c > 0$. We improve this lower bound to $f(n, n) \geq n - \log(n)$ for small n, and $f(n, n) \geq n - \log\log(n)$ for sufficiently large n. To our knowledge, this constitutes the first improvement over the trivial information-theoretic lower bound for the Permutation Game variant of Mastermind.

Our second result concern's Donald Knuth's Minimax algorithm.

Theorem 2. *Consider the original variant of Mastermind (allowing arbitrary n and k), defined by $\Delta = (b, w)$, $R = T$, $A = T$. The Minimax algorithm, described by Donald Knuth in 1976, identifies any hidden sequence H in at most nk queries.*

As a corollary, we have $f(n, k, \Delta = (b, w), R = T, A = T) \leq nk$. Knuth's Minimax algorithm is empirically the best method for solving small games of Mastermind (n and k less than 10) in as few guesses as possible. However, it has proven difficult to analyze the asymptotic performance of the Minimax algorithm, primarily because its behavior is determined by the distribution of possibilities for H, which is difficult to analyze in general terms [11, 12]. To our knowledge, this is the first upper bound on the worst-case performance of the minimax algorithm, but if it performs near-optimally for large n and k, we would expect this bound to be much smaller. We know, for example, that $f(n, k) = O(n \log k)$ for k not too large [10, 11].

Our third result relates to non-adaptive variants of Mastermind, which correspond to $A = F$ in our notation.

Theorem 3. *Consider non-adaptive Mastermind, defined by $A = F$. Let $\Delta = b$, and let n, $k \geq n$, and R be fixed. Then*

$$f(n, k) = \Omega\left(n \log(k)\right).$$

The case $R = T$ was proved by Doerr, Spöhel, Thomas, and Winzen in [4]. They use an information-theoretic argument based on Shannon entropy by letting the hidden sequence be randomly chosen. Shannon entropy is discussed briefly in Section (3.1.1). We use a similar argument to extend this to $R = F$, and these lower bounds together cover the case $n \geq k$. For neither choice of R do we have provably optimal upper bounds; when $R = T$ a corollary of a result by Doerr et al. in [4] gives an upper bound of $O(k \log k)$ guesses, and for $R = F$ no improvement over the nk bound is known. On the other hand, Doerr et al. are able to extend a result of Chvátal in [3] to provide tight bounds for $n \leq k$ with $R = T$.

2 Adaptive Variants of Mastermind

2.1 The Permutation Game

We begin by performing an analysis of the Permutation Game, defined by $n = k$ and $R = F$. We can then show that the information from white-peg responses is irrelevant: These parameters together imply $H = \overline{\sigma}(\mathcal{A})$ for some permutation $\overline{\sigma} \in S_k$ (recalling that $\mathcal{A} = \{1, 2, \ldots, k\}$). Moreover, recalling the definition of the white-peg distance function

$$w(Q_t, H) = \max_{\sigma \in S_{Q_t}} b(\sigma(Q_t), H) - b(Q_t, H),$$

and letting $Q_t = \tau(\mathcal{A})$ for some $\tau \in S_k$, we have that with $\sigma = \overline{\sigma} \circ \tau^{-1}$ we have $\sigma(Q_t) = \overline{\sigma} \circ \tau^{-1} \circ \tau(\mathcal{A})$ which is just H, and then $b(\sigma(Q_t), H) = b(H, H)$ achieves the maximum possible value of n. Then $w(Q_t, H) = n - b(Q_t, H)$ and is uniquely determined by $b(Q_t, H)$ and so no new information is provided from white-peg responses. As such, for the permutation game we will let $\Delta = b$, as $\Delta = bw$ is precisely the same game.

As stated in the introduction, an upper bound $f(n, n) \leq c \cdot n \log(n)$ was proved in 1986 by Ko and Teng, and again in 2013 for a smaller constant $c > 0$ by Ouali and Sutherland [11, 12]. Here we establish lower bounds for $f(n, n) = f(n, k = n, \Delta = b, R = F, A = T)$ concerning the Permutation Game.

In this section we will reference the derangements function, denoted $D(n)$ in this paper and $!n$ in some sources. It counts the number of permutations σ of $\{1, 2, \ldots n\}$ with no fixed points i where $\sigma(i) = i$. The precise number of such permutations is

$$D(n) = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}, \tag{2}$$

which is the nearest integer to $n!/e$ [9].

2.1.1 Trivial Lower Bound

We begin by establishing the trivial information-theoretic result that

$$f(n, n) \geq \log_n(n!) = n - \frac{n}{\ln(n)} + O(1).$$

Proof. Consider an instance of the Permutation Game with the black-peg-only distance function Δ. To uniquely identify H after m turns, the query response vector $(\Delta(Q_1, H), \Delta(Q_2, H), \ldots, \Delta(Q_m, H))$ must distinguish between all $n!$ permutations of the alphabet $\mathcal{A}$. In the t^{th} turn, the codebreaker submits a

query sequence Q_t, which has n possible associated responses given by $\Delta(Q_t, H) \in \{0, 1, \ldots, n-2, n\}$. We note that it is not possible to get $\Delta(Q_t, H) = n-1$ in the Permutation Game. Therefore there are n^m possible query response vectors of length m, hence to identify H in m queries requires that $n^m \geq n!$. This implies $m \geq \log_n(n!)$, and the result follows immediately. □

2.1.2 Buckets

To improve upon the trivial lower bound, we first introduce terminology with which to analyze the distribution of the possible solutions after feedback $\Delta(Q_t, H)$ from a given query Q_t. Let Δ be the black-peg-only distance function. Given a query vector Q_t, the codebreaker may partition the set S_t of remaining solutions into subsets which we will call *buckets*; one bucket for each possible response given by the codemaker. We now formalize the notion of a bucket.

Consider the t^{th} turn in an instance of the Permutation Game. As described in the introduction, the codebreaker may use the responses $\Delta(Q_1, H), \Delta(Q_2, H), \ldots, \Delta(Q_{t-1}, H)$ to construct a minimal set S_t of sequences such that $H \in S_t$. Suppose the codebreaker submits the query $Q_t = (q_1, q_2, \ldots, q_n)$. Then we define the *bucket* $B_t(s)$ *implied by response* $\Delta(Q_t, H) = s$ to be

$$B_t(s) = \{X \in S_t \mid \Delta(Q_t, X) = s\}.$$

Such a construct is useful in that if $\Delta(Q_t, H) = s$, the codebreaker may deduce that if a sequence $X \in S_t$ satisfies $\Delta(Q_t, X) \neq s$, then $X \neq H$. Therefore the codebreaker may construct the set S_{t+1} of possible solutions after the t^{th} turn is complete by setting $S_{t+1} = B_t(s)$.

We now compute the size of a bucket $B_1(s)$ produced in the first turn of an instance of the Permutation Game. First we choose the s colors that are fixed points with respect to the query sequence Q_1. We then permute the remaining $n-s$ colors without any fixed points, which can be done in $D(n-s)$ ways, with $D(i)$ as defined in (2). Hence we have

$$|B_1(s)| = \binom{n}{s} D(n-s),$$

for the bucket implied by response $\Delta(Q_1, H) = s$. Clearly the buckets $B_1(1), B_1(2), \ldots, B_1(n)$ partition the set S_1 of $n!$ possible hidden vectors.

The above computation tells us the *initial* size of each bucket, which is independent of the initial query Q_1. Now consider the evolution of bucket sizes, *i.e.,* the sequence $|B_1(s)|, |B_2(s)|, |B_3(s)|, \ldots$ over the course of an instance of the Permutation Game. We claim that each of these is bounded by $|B_1(s)|$. Since $|B_1(s)|$ is independent of Q_1, assume that $Q_1 = Q_t$. Then the query Q_t, together with response $\Delta(Q_t, H)$, produces a bucket $B_t(s)$ defined by $B_1(s) \cap S_t$ and so $B_t(s) \subseteq B_1(s)$. Hence bucket sizes are bounded above by their initial size, *i.e.,*

$$|B_t(s)| \leq |B_1(s)| = \binom{n}{s} D(n-s).$$

2.1.3 Improved Lower Bound for the Permutation Game

We now employ the notion of a bucket to improve the trivial lower bound for $f(n, n)$ in the context of the Permutation Game.

Proof of Theorem 2. Let n be fixed. We will construct a sequence $L(t)$ that, for any possible guessing strategy, satisfies $L(t) \leq |S_t|$ for some hidden code H. We recall that S_t is the minimal set of sequences

guaranteed to contain H given the responses $\Delta(Q_1, H), \Delta(Q_2, H), \ldots, \Delta(Q_{t-1}, H)$. From this definition, we can set $L(0) = n!$ because S_0 must contain all $n!$ possible solutions, since no guesses have been made. Next, we construct $L(t)$ inductively from $L(t-1)$ by guaranteeing that when proceeding from $L(t-1)$ to $L(t)$, we eliminate at least as many solutions as any possible guess would eliminate. That is, $L(t-1) - L(t) \geq |S_{t-1}| - |S_t|$. This, combined with the assumption that $L(t-1) \leq |S_{t-1}|$, guarantees $L(t) \leq |S_t|$, as desired. Once we have determined the sequence $L(t)$, we find the largest t such that $L(t) > 1$. This would guarantee that for each guessing strategy, there is some hidden code for which t turns of the strategy would still leave at least 2 codes as possibilities, and so the game would not be solved. This minimal t would therefore be a lower bound on the guessing effectiveness of all strategies.

Consider the t^{th} turn in an instance of the Permutation Game. In the terminology of buckets, the codebreaker constructs the set S_t in turn t by setting $S_t = B_{t-1}(\Delta(Q_{t-1}, H))$. To choose an "optimal" query sequence Q_t (in the sense described above, which maximizes $|S_{t+1}| - |S_t|$), the codebreaker therefore must choose a sequence Q_t which minimizes

$$\max_{s \in \{1,2,\ldots,n\}} |B_{t+1}(s)|.$$

Equivalently, the codebreaker chooses Q_t such that the size of the largest bucket is as small as possible. By the discussion in Section (2.1.2), we know that $|B_t(s)| \leq \binom{n}{s} D(n-s)$. The optimal query Q_t will partition S_t into buckets as evenly as possible, and some buckets will have size equal to their upper bound, while others will be partially filled. This best-possible partition will partition S_t into two sets X and Y of buckets, such that $A \in X$ implies bucket A is incompletely filled, $B \in Y$ implies bucket B is completely filled, and $A \in X$, $B \in Y$ implies that the upper bound on $|A|$ is greater than the upper bound on $|B|$. Let $x = |X|$, so that x is the number of incompletely filled buckets.

We now give a recurrence which bounds $L(t)$. At the beginning of the t^{th} turn, we have at least $L(t-1)$ possible solutions remaining, so $|S_t| \geq L(t-1)$. Suppose that the codebreaker has selected the optimal query vector Q_t for submission in turn t. We then have

$$\begin{aligned}
L(t-1) &\leq \sum_{i=0}^{n} |B_i(t)| \\
&= \sum_{i=0}^{x-1} |B_i(t)| + \sum_{i=x}^{n} |B_i(t)| \\
&\leq \sum_{i=0}^{x-1} L(t) + \sum_{i=x}^{n} \binom{n}{i} D(n-i) \quad \text{for optimal } x \\
&= x \cdot L(t) + \sum_{i=x}^{n} \binom{n}{i} D(n-i). \qquad (3)
\end{aligned}$$

Now we find the x that generates this optimal distribution and solve for $L(t)$. Once we know $L(t)$, we claim that the inequality in equation (3) holds for all x. Picking x larger than its optimal value increases the contribution of some of the entirely filled, smaller buckets up to $L(t)$, thus increasing the total sum. Picking x too small increases the contribution of some of the under-filled, larger buckets up to their full sizes, thereby increasing the total sum. So adjustment of x about its optimal value always preserves the inequality. Thus (3) holds for all x.

We now prove the following lemma.

Lemma 1.

$$\sum_{i=x}^{n} \binom{n}{i} D(n-i) \leq \frac{n!}{x!}.$$

Proof. We give a combinatorial proof: The left-hand side denotes the number of permutations of an n-element vector which have at least x fixed points. Suppose we choose x fixed points and simply permute the rest of the vector. This gives $\binom{n}{x}(n-x)! = \frac{n!}{x!}$ possible permutations. Clearly this includes all vectors with at least x fixed points (and over-counts by some margin), so the inequality holds. □

From Lemma 1 and (3), we have

$$L(t-1) \leq x \cdot L(t) + \frac{n!}{x!},$$

which gives

$$L(t) \geq \frac{1}{x}\left(L(t-1) - \frac{n!}{x!}\right).$$

The following lemma will also be of use.

Lemma 2. *We recall that n is fixed in defining $L(t)$. For any positive integer $C_n < n$, we have*

$$\frac{L(t)}{n!} \geq \frac{C_n! - (H_{C_n+t} - H_{C_n})}{(C_n+t)!}$$

for all $0 \leq t \leq n - C_n$, where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n^{th} harmonic number.

Proof. We prove inductively. With $t = 0$, we have $L(0) = n!$. Then

$$1 = \frac{L(0)}{n!} \geq \frac{C_n! - (H_{C_n} - H_{C_n})}{C_n!} = 1,$$

and the inequality is satisfied. Now we move to the general case. Recalling that (3) holds for all x, we will let $x = t + C_n$. Solving for $L(t)$ gives us

$$\frac{L(t)}{n!} \geq \frac{1}{C_n+t}\left(\frac{L(t-1)}{n!} - \frac{1}{(C_n+t)!}\right).$$

Assuming the lemma inductively for $t-1$, we obtain

$$\begin{aligned}
\frac{L(t)}{n!} &\geq \frac{1}{C_n+t}\left(\frac{C_n! - (H_{C_n+t} - H_{C_n})}{(C_n+t-1)!} - \frac{1}{(C_n+t)!}\right) \\
&\geq \left(\frac{C_n! - (H_{C_n+t-1} - H_{C_n}) - \frac{1}{C_n+t}}{(C_n+t)!}\right) \\
&\geq \left(\frac{C_n! - (H_{C_n+t} - H_{C_n})}{(C_n+t)!}\right),
\end{aligned}$$

which completes the induction. □

Now, for $t = n - C_n$ we have

$$L(n-C_n) \geq n!\left(\frac{C_n! - (H_n - H_{C_n})}{n!}\right) = C_n! - (H_n - H_{C_n}).$$

We choose C_n by the following reasoning: we want to achieve a lower bound of $n - C_n$ turns to solve the game. This would occur if $L(n-C_n)$ were greater than 1, as there would still be at least 2 possible solutions after $n - C_n$ turns, and thus the game would not yet be solved. Thus, we want

$$C_n! - (H_n - H_{C_n}) > 1.$$

Noting that $H(n)$ is asymptotic to $\log(n)$, as long as $\log(n) = o(C_n!)$ we will eventually have that $C_n! - H_n > 1$. Since we have, for example, that $\log x = o((\log\log x)!)$, we have that with $C_n = \lceil \log\log(n) \rceil$ the above inequality will eventually be satisfied.

In conclusion, when n is sufficiently large, the minimum number of remaining possible solutions after $n - \lceil \log\log(n) \rceil$ guesses is at least

$$S_{n-\lceil \log\log(n) \rceil} \geq L(n - \lceil \log\log(n) \rceil) \geq (\log\log(n))! - (H_n - H_{\log\log(n)}) > 1.$$

Thus there is no strategy that can identify any hidden sequence in fewer than $n - \log\log(n)$ turns, and so in the Permutation Game variant of Mastermind, we have $f(n, n) \geq n - \log\log(n)$ for all sufficiently large n. □

2.2 Linear Algebra and the Minimax Algorithm

We now move on to an analysis of the Minimax algorithm. To perform this analysis, we first change our notation. We can represent all guess vectors and hidden vectors as $(0, 1)$-vectors in $\mathbb{R}^{n\times k}$, which, for convenience, we write as matrices. These matrices are constructed in the following manner: a matrix with a 1 as the $(i, j)^{\text{th}}$ entry represents a guess where the i^{th} position is the j^{th} color. As such, each row will have exactly one 1, as each position is exactly one color. We refer to these matrices as *representations* of the guess and hidden vectors.

With this notation, the black-peg distance becomes the dot product of the representations of the guess and the hidden vector, as there will be a contribution to the dot product exactly when both representations have a one in the same column in the same row, i.e. there is a color in the same spot of both vectors.

The goal of Mastermind is then to find the (necessarily unique) valid representation such that its dot product with the hidden vector is n. By linearity of the dot product, once we know the value of the dot product of the hidden vector with some set of guess vectors, we know the value of the dot product of the hidden vector with any vector in the span of these guesses. Therefore, it is an immediate consequence that a winning strategy can be found by querying a basis for the span of the valid guess vectors, which, as a subspace of $\mathbb{R}^{n\times k}$, must be of size at most nk. Note that this strategy does not make use of adaptive feedback, of white-peg responses, or the condition on repeated colors. Thus, for any choice of Δ, A, and R, we have $f(n, k) \leq nk$.

This representation also allows us to bound the minimax algorithm of [10]. The bulk of the proof is the following lemma:

Lemma 3. *At each turn, the minimax algorithm either guesses the hidden vector or guesses a vector whose representation is linearly independent of the previous guesses' representations.*

Theorem 2 follows as an immediate corollary from this lemma: Since these representations are in $\mathbb{R}^{n\times k}$, the minimax algorithm can make at most nk linearly independent guesses. After that, the minimax cannot make a linearly independent guess and by the above lemma must then guess the hidden vector, for an upper bound of nk turns to determine the hidden vector and one more to guess it.

Proof of Lemma 3. If the hidden vector is known, the minimax algorithm will guess it. This will certainly occur when the hidden vector's representation is a linear combination of the previous guess vectors' representations, as by the logic above we can compute the dot products of the hidden vector with every possibility in the span of the previous guesses, and we would find the guess with a dot product of n.

Otherwise, the hidden vector is linearly independent from the previous guess vectors. Assume this is the case. If it is still uniquely determined, the algorithm will guess it. Regardless, guessing a vector q that is a linear combination of the previous guesses returns no new information, as the response to that guess can already be computed. Since q is linearly dependent on the previous guesses, it cannot, by our assumption, be the hidden vector, and so guessing it eliminates zero vectors and doesn't solve the game. If we can find a new query r that is guaranteed to either eliminate at least one vector or solve the game, the minimax algorithm will choose r over q. Since the hidden vector must exist and by our assumption is linearly independent of the guess vectors, there is at least one vector linearly independent of the previous guesses. Guessing such a vector will win the game with a black-peg response of n, and eliminate at least one vector (itself) with any other response. Thus, according to the minimax algorithm, guessing any vector linearly independent of the previous guesses will always be strictly better than guessing a linearly dependent vector, and so if the hidden vector is unknown the best guess will be necessarily independent of the previous guesses. □

3 Non-Adaptive Variants

This section follows closely the reasoning of Doerr et al. [4] as they analyze non-adaptive games. We perform an analysis of the Mastermind variant in which Δ is the black-peg-only distance function, $R = F$ and $A = F$ (*i.e.,* no repeats and non-adaptive). We first introduce Shannon entropy, which we then employ to give a lower bound for $f(n, k)$ when no repeats are allowed. We conclude with a small extension of one of the proofs due to Doerr et al. to provide an upper bound on $f(n, k)$ for non-adaptive variants when $k \geq n$.

3.1 Non-Adaptive, No Repeats, and Black-peg-only Distance Function

3.1.1 Shannon Entropy

First we give a brief definition of entropy in the context of information theory, where it is often called *Shannon entropy.* Our presentation borrows from [6]. Shannon entropy is intuitively the "expected level of surprise" of a random variable, and hence is based upon the definition of a surprise function S. Let X and Y be a random variables and let $X = x$ and $Y = y$ be events. Then we will define a surprise function S which satisfies $S(X = x \wedge Y = y) = S(X = x) + S(Y = y \mid X = x)$. It turns out that, with a few added assumptions, the only natural choice for S is $S(x) = -\log_2(\mathbb{P}[X = x])$ [6].

Let D be the domain of the random variable X. Then the *Shannon entropy* $H(X)$ of the random variable X is defined to be

$$\begin{aligned} H(X) &= \mathbb{E}[S(x)] \\ &= \sum_{x \in D} \mathbb{P}[X = x] \cdot (-\log_2(\mathbb{P}[X = x])). \end{aligned}$$

One can easily show that Shannon entropy is *subadditive* [6]. That is, if $X_1, X_2, \ldots, X_n$ are random variables, then

$$H(X_1, X_2, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i). \tag{4}$$

We will use Shannon entropy in the following proof.

3.1.2 Lower Bound: $f(n, k, \Delta = b, R = F, A = F)$

Proof of Theorem 3. We prove Theorem 3 in the case $R = F$, noting that [4] have already proved the case $R = T$. We will be drawing substantially from the techniques they use in their proof. Consider a set $\{Q_1, Q_2, \ldots, Q_s\}$ of s query sequences such that all $k!/(k-n)!$ possible hidden sequences are uniquely determined by the responses $\Delta(Q_1, H), \Delta(Q_2, H), \ldots, \Delta(Q_s, H)$. That is, assume that if two possible hidden vectors H and H' satisfy $\Delta(Q_i, H) = \Delta(Q_i, H')$ for all i, then $H = H'$.

Let the hidden vector Z be sampled uniformly at random from the set of $k!/(k-n)!$ possibilities. Then the responses $Y_i = b(Z, q_i)$ are now random variables, and the vector $Y = (Y_1, Y_2, \ldots, Y_s)$ is also a random variable. By our assumptions, Y always uniquely determines, and is uniquely determined by, Z. Therefore $H(Z) = H(Y)$ (which follows from the bijection of events $Z = z$ with events $Y = y$). Since Z is a random variable with $k!/(k-n)!$ outcomes of equal probability, we compute

$$H(Z) = \log_2 \left(\frac{k!}{(k-n)!} \right). \tag{5}$$

By (4), we have

$$H(Y) \leq \sum_{i=1}^{s} H(Y_i). \tag{6}$$

Now we bound $H(Y_i)$. We recall that by assumption, $\Delta = b$ so Y_i is precisely the black-peg distance between Q_i and Z. By definition,

$$H(Y_i) = -\sum_{x=0}^{n} \mathbb{P}[Y_i = x] \cdot \log_2(\mathbb{P}[Y_i = x]). \tag{7}$$

We now compute $\mathbb{P}[Y_i = x]$, namely the probability that X is a solution vector with x fixed points with respect to the query q_i. Using our previous terminology, this is the probability that X is in bucket $B(x)$. Using the same logic as in Lemma 1 to bound $|B(x)|$, we have

$$\begin{aligned}
\mathbb{P}[Y_i = x] &= \frac{|B(x)|}{\left(\frac{k!}{(k-n)!}\right)} \\
&\leq \frac{\binom{n}{x}\frac{(k-x)!}{(k-n)!}}{\left(\frac{k!}{(k-n)!}\right)} \\
&= \frac{1}{x!} \cdot \frac{n(n-1)\cdots(n-x+1)}{k(k-1)\cdots(k-x+1)} \\
&\leq \frac{1}{x!}.
\end{aligned}$$

It turns out that we can't substitute this upper bound into (7) for all x, because $f(\alpha) = -\alpha \log_2 \alpha$ is an increasing function only when $\alpha < 1/e$. So we can plug in the upper bound of $\Pr[X = x] \leq 1/x!$ when $x \geq 3$ and still have an upper bound. For the first three values of x, we will instead use the trivial

upper bound $f(\alpha) \leq 1/(e \log 2)$. Then

$$\begin{aligned} H(Y_i) &\leq \frac{3}{e \log 2} + \sum_{x=3}^{n} -\frac{1}{x!} \cdot \log_2 \left(\frac{1}{x!}\right) \\ &\leq \frac{3}{e \log 2} + \sum_{x=3}^{\infty} \frac{\log_2(x!)}{x!} \\ &< 3. \end{aligned}$$

Combining this with (6) gives $H(Y) \leq 3s$, and since $H(Z) = H(Y)$ we have $H(Z) \leq 3s$. Substituting in (5) as a lower bound for $H(Z)$ and solving for s gives

$$s \geq \frac{1}{3} \log_2 \left(\frac{k!}{(k-n)!}\right)$$

We can show that the right-hand side is $\Omega(n \log k)$. This is immediate for large k relative to n by bounding the ratio by $(k-n)^n$; for small k (e.g. $k \leq 2n$) we bound the ratio by $n!$ and the claim follows from Stirling's approximation. So this gives us a lower bound of $\Omega(n \log k)$ turns for any non-adaptive strategy for Mastermind with no repeats and black-peg responses. □

3.1.3 Extension to white-peg responses

We can also extend the $R = T$ case of Theorem 3, as proved in [4], to include $\Delta = bw$. We do this by showing that any black-white strategy must submit $\Omega(k)$ queries, and that we can convert a black and white-peg strategy into a black-peg-only strategy by adding $O(k)$ queries. Then changing a black-white strategy to a black-only strategy changes the number of queries by at most some absolute constant factor. So we have $f(\Delta = b) = O(f(\Delta = bw))$. Combining this with Theorem 3 implies the theorem for black-white responses as well.

First we describe the conversion. Take a black-white strategy and append, for each color, a query where every spot is that color. As a black-peg only strategy, we can deduce from these queries exactly how many times each color appears in the hidden vector, from which we can determine the white-peg responses to any of the original queries. Therefore, if the old strategy had enough information to determine the hidden vector with black-white responses, this new strategy can determine the hidden vector with just black-peg responses. Since there are k colors, we have appended k queries in this conversion.

Lastly, we just need that any non-adaptive black-white strategy must submit $\Omega(k)$ queries. This follows from the following reasoning: assume that there are two colors a and b such that a non-adaptive strategy guesses neither a nor b in either spot 1 or 2. Then this strategy would not be able to distinguish between a hidden vector starting a, b and one starting b, a. Then by contradiction any non-adaptive strategy must guess at least $k-1$ colors in these two spots. It can guess 2 colors in these spots per turn, for a total of at least $(k-1)/2 = \Omega(k)$ turns required to solve the game. Thus, by the above logic, we conclude $f(n, k) = \Omega(n \log k)$ in this case as well.

3.2 An Upper Bound for Non-Adaptive Variants with Repeats

Consider non-adaptive variants of Mastermind in which repetitions are allowed. First we note that when $k = n$, [4] showed the existence of a set of $O(n \log(n))$ queries which uniquely identify any hidden sequence H. When $k > n$, one can simply extend H and all queries Q_t by $k-n$ "auxiliary" positions until

$n = k$. We fill these auxiliary positions with arbitrary colors, and we adjust the responses accordingly to account for the auxiliary positions. By the results of [4], there exists a set of $O(k \log k)$ queries which will uniquely identify any hidden sequence.

4 Acknowledgments

We would like to thank Daniel Montealegre for supervising and assisting our work throughout the summer, and Nathan Kaplan for both creating and guiding our project and for his invaluable assistance in editing this paper. We would also like to thank Sam Payne and the Summer Undergraduate Math Research at Yale program for organizing, funding, and supporting this project. SUMRY is supported in part by NSF grant CAREER DMS-1149054.

References

[1] Bshouty, N. H. Optimal algorithms for the coin weighing problem with a spring scale. *COLT 1* (2009).

[2] Chen, Z., Cunha, C., and Homer, S. Finding a hidden code by asking questions. *Lecture Notes in Computer Science 1* (2005), 50–55.

[3] Chvátal, V. Mastermind. *Combinatorica 3* (1983), 325–329.

[4] Doerr, B., Spohel, R., Thomas, H., and Winzen, C. Playing mastermind with many colors. *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (2013), 695–704.

[5] Erdos, P., and Renyi, A. On two problems of information theory. 229–243.

[6] Galvin, D. Three tutorial lectures on entropy and counting. *First Lake Michigan Workshop on Combinatorics and Graph Theory* (3 2014).

[7] Goodrich, M. T. On the algorithmic complexity of the mastermind game with black-peg results. *Information Processing Letters 109* (2009), 675–678.

[8] Grebinsky, V., and Kucherov, G. Optimal reconstruction of graphs under the additive model. *Algorithmica 1* (2000), 104–124.

[9] Hassani, M. Derangements and applications. *Journal of Integer Sequences 6* (2 2003).

[10] Knuth, D. The computer as master mind. *Journal of Recreational Mathematics 9*, 1 (2 1976), 2–7.

[11] Ko, K.-I., and Teng, S.-C. On the number of queries necessary to identify a permutation. *Journal of Algorithms 7* (7 1986), 449–462.

[12] Ouali, M. E., and Sauerland, V. Improved approximation algorithm for the number of queries necessary to identify a permutation. *arXiv 1* (3 2013), 1–415.